Der JAVA^{TM}–COACH

— Modellieren mit UML, Programmieren mit Java^{TM} 2 Plattform (J2SE & J2EE), Dokumentieren mit XHTML —

Hinrich E. G. Bonin[1]

5-Oct-1997 – 30-May-2005

[1]Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. Bonin, University of Applied Sciences, Fachhochschule Nordostniedersachsen, Volgershall 1, D-21339 Lüneburg, Germany.

mailto:bonin@uni-lueneburg.de

Inhaltsverzeichnis

Abbildungsverzeichnis

*

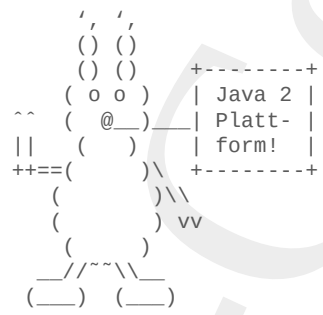
Tabellenverzeichnis

*

Kapitel 1

Vorspann

1.1 Zusammenfassung



Unstrittig gilt in jeder Softwareentwicklungsumgebung die Aussage *Programmieren bleibt schwierig!* — insbesondere bei der Entwicklung großer Systeme, die, verteilt auf viele Computer, parallel und asynchron agieren. Dies gilt auch für Java^{TM} von Sun Microsystems, Inc. USA. Klar ist, mit Java 2 Standard Edition (J2SE) und darauf aufbauend Java 2 Enterprise Edition (J2EE) lassen sich bewährte Softwarekonstruktionen (≈ Muster & Rahmenwerke) direkt nutzen.

So können einige Schwierigkeiten durch „Abkupfern“ gelungener Konstruktionen leicht gemeistert werden. Klar ist aber auch, die komplexe Web-geprägte Mehrschichtenarchitektur und ihre Halbfertigprodukte wie *Enterprise JavaBeans* verlangen ein fundiertes Verständnis der Objekt-Orientierung und ihrer Realisierung in und mit Java.

Der JAVA^{TM}–COACH[1] versucht ein solches Verständnis Schritt

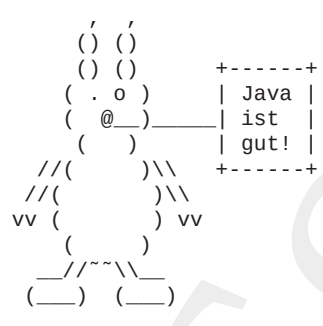
[1] Hinweis: Dieses Dokument ist ein Entwurf und wird laufend fortgeschrieben. Die aktuelle Version befindet sich unter: `http://as.fhnon.de/publikation/anwdall.pdf`. Anmerkungen und

für Schritt aufzubauen. Deshalb werden auch Themen wie beispielsweise anonyme Klasse und Reflektion behandelt. Bei den Beispielen, Übungen und Musterlösungen geht es primär um ein Begreifen und Umgehen mit der Komplexität, die einer Software innewohnt. Plakativ formuliert möchte der JAVA^{TM}–COACH Ihnen helfen Java^{TM} als ein Akronym für *Just a valid application* zu verstehen.

Kommentare schicken Sie bitte an: mailto:bonin@fhnon.de (Remark: This is a **draft document** and continues to be revised. The latest version can be found at http://as.fhnon.de/publikation/anwdall.pdf. Please send comments to mailto:bonin@fhnon.de)

1.2 Vorwort

Sie wollen und/oder müssen sich mit der Softwareentwicklung in und mit Java^{TM} befassen? Das ist gut so! Schreiben von Programmen in Java^{TM} macht Freude. Als alter Fan von *List Processing* (LISP) (↪ [Bonin91b]) ergeht es mir seit Oktober 1997 ähnlich. Ob nun Java^{TM} noch im Jahre 2010 relevant ist oder sich in einer Rolle wie heute LISP befindet, ist unerheblich. Es geht um die Frage wie kann effektiv ein Java-Begreifen ermöglicht werden.



Ziel ist es, das breite Spektrum der Java^{TM} Möglichkeiten aufzuzeigen und eine hohe Qualität bei der Programmierung sicher zu stellen. Die Qualität eines Programms hängt primär von der Qualität der Modelle zur Abbildung von Benutzeranforderungen und deren Umsetzung in die Welt der Objekte ab. Für diesen Zweck vermittelt der JAVA^{TM}–COACH das Modellieren in der Standardsprache *Unified Modeling Language* (UML).

Ohne eine Dokumentation ist ein Programm werder vollständig noch verstehbar. In allen Phasen der Entwicklung entstehen die vielfältigsten Dokumente. Um sich in dieser Menge erfolgreich bewegen zu können, wird die Sprache des Webs *HyperText Markup Language* (HTML[2]) zum Dokumentieren genutzt. Dabei werden Erfahrungen zur Sicherung einer Einheitlichkeit (Links, Namensvergabe, Layoutvorgabe) vermittelt. Die Bausteine und -Konstruktionen werden stets im Wechselspiel zur Fachwelt (≈ Anforderungen) und zur Dokumentation verdeutlicht. Der JAVA^{TM}–COACH umfaßt daher folgende Aspekte:

UML – Von unscharfen Vorgaben zum fachlichen Modell in Form von Klassendiagrammen mit Darstellung der Beziehungen

[2] Präzise formuliert: XHTML Version 1.0 (Exentsible Hypertext Markup Language — Eine Spezifikation von HTML 4.0 in XML —) ↪ `http://www.w3.org/TR/2000/REC-xhtml1-20000126/` (Zugriff: 23-Oct-2001)

zwischen den Objekten (Vererbung, Assoziation, Aggregation und Komposition).

- Umsetzung des fachlichen Modells in Java-Bausteine
- Tipps zur Vorgehensweise

J2SE & J2EE Java 2 Plattform

- Paradigma der Objekt-Orientierung
- Konzeption und Nutzung der Java 2 Plattform (Application, Applet, Bytecode, 80/20-Modell, mobiles Code-System)
- Erläuterung von Grundbausteinen (primitiven Datentypen, Operatoren, Parameterbindung, Kontrollstrukturen (Alternative, Iteration) und Rekursion).
- Erörterung von mächtigen Konstruktionen (Nebenläufigkeit, Delegationsmodell, persistente Objekte, innere Klassen, *Reflection*, *Cloning*, verteilte Objekte und Komponentenmodelle.

XHTML

- Tipps zur praxisgerechten Dokumentation
- in einem einheitlichen Stil (*Cascading Style Sheets*).

Der JAVA^{TM}–COACH wendet sich an alle, die auf einer fundierten Theoriebasis Schritt für Schritt anhand von praxisnahen Beispielen Java^{TM} gründlich verstehen wollen. Dabei spielt Ihr Alter keine Rolle, denn nach den neueren Erkenntnissen der Hirnforschung verfügt das Hirn über eine hohe Plastizität und die Fähigkeit der Neurogenese bei der neue Nervenzellen in bestehende Verschaltungen eingefügt werden. Dank dieser Hirneigenschaften *kann Hans also durchaus noch lernen, was Hänschen nicht gelernt hat* — auch wenn es mit den Jahren deutlich schwerer fällt.

Im Mittelpunkt steht das objekt-orientierte Programmieren. Dazu muß der Beginner viele Konstrukte erlernen und bewährte Konstruktionen nachbauen. Im Rahmen der Modellierung wird die „Benutzermaschine“ mit Hilfe von UML spezifiziert und die „Basismaschine“ in J2SE bzw. J2EE implementiert. Die Dokumentation der Software erfolgt als verknüpfte Hypertextdateien (in XHTML, also in der „*Reformulation*“ von HTML 4.0 in XML).

Ebensowenig wie zum Beispiel Autofahren allein aus Büchern erlernbar ist, wird niemand zum „Java-Wizard“ (deutsch: Hexenmeister) durch das Nachlesen von erläuterten Beispielen. Das intensive Durchdenken der Beispiele im Dialog mit einer laufenden Java 2 Plattform vermittelt jedoch, um im Bild zu bleiben, unstrittig die Kenntnis der Straßenverkehrsordnung und ermöglicht ein erfolgreiches Teilnehmen am Verkehrsgeschehen — auch im Großstadtverkehr. Für diesen Lernprozeß wünsche ich der „Arbeiterin“ und dem „Arbeiter“ viel Freude.

Der JAVA^{TM}–COACH ist konzipiert als ein Buch zum Selbststudium und für Lehrveranstaltungen. Mit den umfassenden Quellenangaben und vielen Vor- und Rückwärtsverweisen dient es auch als Nachschlagewerk und Informationslieferant für Spezialfragen.

Der JAVA^{TM}–COACH vermittelt mehr als das üblichen Java-Einführungsbuch mit vielen Web-Seiten-Beispielen. Er befaßt sich eingehend mit mächtigen Konstruktionen, die in klassischen Sprachen[3], kaum oder gar nicht machbar sind, wie zum Beispiel *Multithreading*, *Inner Classes*, *Serialization*, *Reflection* und *Remote Method Invocation*. Erläutert wird die Integration eines objekt-orientierten Datenbankmanagementsystems am Beispiel der Software `FastObjects t7` der Firma Poet[4]. Vertieft werden Konzeptionen und Möglichkeiten von Komponentenmodellen. Dazu werden JavaBeans^{TM} und EJB (*Enterprise JavaBeans*^{TM}) erklärt.

Der JAVA^{TM}–COACH folgt nicht den üblichen Einführungen in eine (neue) Programmiersprache. Diese beginnen meist unmittelbar mit dem obligatorischen Beispiel „Hello World“[5]. Zunächst werden eingehend die prägenden Ideen und Konzepte von Java^{TM} erläutert. Damit die Programme überhaupt in der gewünschten Qualität erzeugbar sind, müssen die „**richtigen**“ **Objekte** und deren „**richtigen**“ **Beziehungen** aus der Anwendungswelt erkannt und präzise notiert werden. Deshalb erklärt der JAVA^{TM}–COACH vorab die „**UML-Boxologie**“ ehe das Buch von Quellcodebeispielen bestimmt wird.

Der JAVA^{TM}–COACH wurde 1997 begonnen und in einigen Etappen den jeweiligen Java^{TM} Entwicklungen angepasst. Im Jahr 1997 war die Welt noch recht einfach. Einen *Component Transaction Monitor* in

1997: JDK 1.x

[3] Exemplarisch ist hier die Programmiersprache COBOL zu nennen.

[4] ↪ `http://www.fastobjects.de` (Zugriff 18.12.2001)

[5] Dieses Beispiel befindet sich im JAVA^{TM}–COACH erst im Abschnitt 6.1.1 S. 84.

2002:
J2EE

der Form von EJB kannte das damals verfügbare *Java Development Kit* (JDK 1.0) noch nicht. Im Dezember 2002 wurde dieses Komponentenmodell der Java^{TM} 2 Plattform eingebaut. Während dieser Fortschreibung lernt man erfreulicherweise stets dazu. Das hat jedoch auch den Nachteil, daß man laufend neue Unzulänglichkeiten am Manuskript erkennt. Schließlich ist es trotz solcher Schwächen der Öffentlichkeit zu übergeben. Ich bitte Sie daher im voraus um Verständnis für Unzulänglichkeiten. Willkommen sind Ihre konstruktiven Vorschläge, um die Unzulänglichkeiten Schritt für Schritt weiter zu verringern. Ihre Vorschläge werden mit Ihrer Zustimmung über den Web-Server:

`http://as.fhnon.de`

verfügbar gemacht. Dort finden Sie auch alle aktuellen Ergänzungen.

Danksagung

Für das Interesse und die Durchsicht einer der ersten Fassungen danke ich meinem Kollegen Prof. Dr. Fevzi Belli (Universität Paderborn). Ohne die kritischen Diskussionen mit Studierenden im Rahmen der Lehrveranstaltungen *Anwendungsentwicklung* und die Beiträge von Ehemaligen, die auf aktuellen Praxiserfahrungen basieren, wäre der JAVA^{TM}–COACH nicht in dieser Form entstanden. Ihnen möchte ich an dieser Stelle ganz besonders danken. In diesem Kontext möchte ich exemplarisch für viele die beiden Diplom-Wirtschaftsinformatiker Sven Hohlfeld und Stephan Wiesner erwähnen.

Lüneburg, 30. Mai 2005

```
<Erfasser>
     <Verfasser>
          Hinrich E. G. Bonin
     </Verfasser>
</Erfasser>
```

1.3 Notation

In diesem Buch wird erst gar nicht der Versuch unternommen, die weltweit übliche Informatik-Fachsprache Englisch zu übersetzen. Es ist daher teilweise „mischsprachig“: Deutsch und Englisch. Aus Lesbarkeitsgründen sind nur die männlichen Formulierungen genannt; die Leserinnen seien implizit berücksichtigt. So steht das Wort „Programmierer“ hier für Programmiererin und Programmierer.

Für die Notation des („benutzernahen“) Modells einer Anwendung wird in *Unified Modeling Language* (UML) genutzt. Ursprünglich ist UML eine Zusammenführung der Methoden von Grady Booch, James Rumbaugh und Ivar Jacobson. Jetzt ist UML die Standardsprache für die Spezifikation, Visualisierung, Konstruktion und Dokumentation von „Artifakten“[6] eines Softwaresystems. Der UML-Standard wird von der OMG[7] betreut ↪ Abschnitt 4.7 S. 69. Leider ist UML noch immer eine bloße Sammlung von Konzepten – so wie in den 60igern `PL/1` bei den Programmiersprachen — und noch keine konsequente Synthese dieser Konzepte (z. B. ↪ [Broy/Siedersleben02] S. 5). **UML**

Für die Notation von („maschinennahen“) Modellen beziehungsweise Algorithmen wird auch im Text Java^{TM} verwendet. Beispielsweise wird zur Kennzeichnung einer Prozedur (Funktion) — also eines „aktivierbaren“ (Quellcode-)Teils — eine leere Liste an den Bezeichner angehängt, zum Beispiel `main()`.

Zur Beschreibung der Strukturen in Dokumenten werden XHTML[8]-Konstrukte verwendet — soweit möglich. Zum Beispiel generiert `javadoc` noch kein valides XHTML. Die Layout-Spezifikation erfolgt mit Hilfe von *Cascading Style Sheets* (CSS). **XHTML**

Ein Programm (Quellcode) ist in der Schriftart `Typewriter` dargestellt. Ausgewiesene Zeilennummern in einer solchen Programmdarstellung sind kein Bestandteil des Quellcodes. Sie dienen zur Vereinfachung der Erläuterung.

PS: Ab und zu werden zur Aufmuterung und zum Schmunzeln im Text *Internet Smileys*

[6] Als Artifakt wird das durch menschliches Können Geschaffene, das Kunsterzeugnis bezeichnet. Artifakt ist auch das Werkzeug aus vorgeschichtlicher Zeit, das menschliche Bearbeitung erkennen läßt.

[7] OMG ≡ Object Management Group

[8] XHTML ≡ Extensible Hypertext Markup Language

`:-)`	Your basic smiley. This smiley is used to inflect a sarcastic or joking statement since we can't hear voice inflection over e-mail.
`;-)`	Winky smiley. User just made a flirtatious and/or sarcastic remark. More of a ¨don't hit me for what I just said¨ smiley.
`:-(`	Frowning smiley. User did not like that last statement or is upset or depressed about something.
`:-I`	Indifferent smiley. Better than a `:-(` but not quite as good as a `:-)`.
`:->`	User just made a really biting sarcastic remark. Worse than a `;-)`.
`>:->`	User just made a really devilish remark.
`>;->`	Winky and devil combined. A very lewd remark was just made.

Legende:
Quelle ↪ `http://members.aol.com/bearpage/smileys.htm`
(online 21-Nov-2003)

Tabelle 1.1: Internet Smileys

benutzt. Ihre Bedeutung erläutert Tabelle 1.1 S. 24.

Kapitel 2

Java^{TM}-Training — Mehr als Web-Seiten entwickeln

Just a valid application

Unter den Fehlleistungen der Programmierung wird ständig und überall gelitten: Zu kompliziert, zu viele Mängel, nicht übertragbar, zu teuer, zu spät und so weiter. *The Java Factor*[1] soll es besser machen. Der Hoffnungsträger basiert auf einer beinahe konsequent objektorientierten Programmierung. Sie soll die gewünschte Qualität ermöglichen. Dabei wird Qualität durch die Leistungsfähigkeit, Zuverlässigkeit, Durchschaubarkeit & Wartbarkeit, Portabilität & Anpaßbarkeit, Ergonomie & Benutzerfreundlichkeit und Effizienz beschrieben.

Der JAVA^{TM}–COACH schwärmt wohl vom Glanz der „Java-Philosophie“, ist aber nicht euphorisch eingestimmt. Es wäre schon viel erreicht, wenn Sie, liebe Programmiererin, lieber Programmierer, nach dem Arbeiten mit diesem Buch Java^{TM} als ein Akronym für *Just a valid application* betrachten können.

[1] Titelüberschrift von *Communications of the ACM*, June 1998, Volume 41, Number 6.

Trainingsplan

Das Kapitel „Einführung: Java-Training — Mehr als Web-Seiten entwickeln“ gibt einen Überblick über:

- die Java 2 Standard Edition (J2SE)
  ↪ Seite 26 . . .
- die Java 2 Enterprise Edition (J2EE) und
  ↪ Seite 29 . . .
- einige *Application Programming Interfaces* (APIs) und Standarddienste der Java 2 Plattform.
  ↪ Seite 31 . . .

2.1 J2SE

B. Joy

J. Gosling

Java ist eine wunderschöne Insel Indonesiens mit exotischer Ausstrahlung und großem Kaffeeanbau. In der Softwarewelt ist Java^{TM} ein Begriff für eine objekt-orientierte Softwareentwicklung. Als das Projektteam um Bill Joy und James Gosling (Sun Microsystems, Inc. USA) Java^{TM} konzipierten ging es zunächst um die Steuerung von Haushaltsgeräten und danach um dynamische Web[2]-Seiten und um eine allgemeine Programmiersprache mit vielen guten Eigenschaften (↪ Tabelle 2.1 S. 27). In dieser Zeit wurde Java^{TM} primär als eine von `C++` abgeleitete, bessere objekt-orientierte Programmiersprache angesehen und wurde daher scherzhaft als „C plus-plus-minus-minus“ (Bill Joy zitiert nach [Orfali/Harkey97]) bezeichnet.

[2] Web ≡ *World Wide Web*

Java^{TM} ist eine:

- „einfache,
- objektorientierte,
- verteilte,
- interpretierte,
- robuste,
- sichere,
- architektur-unabhängige,
- portable,
- hochperformante,
- multithread-fähige und
- dynamische Sprache“

[Arnold/Gosling96, JavaSpec].

Tabelle 2.1: Java-Beschreibung von Sun Microsystems, Inc. USA

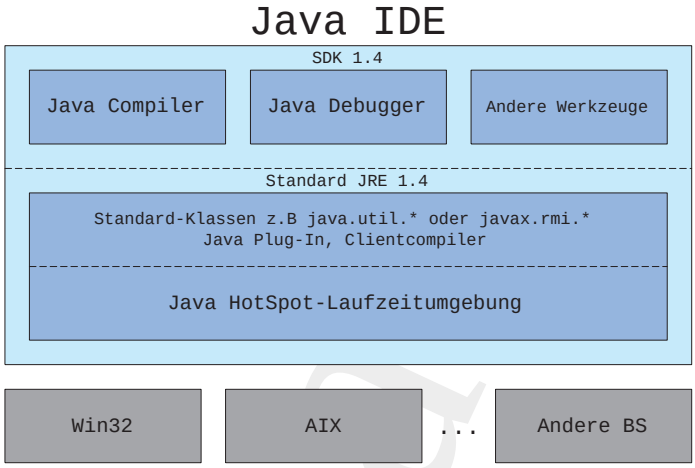
Legende: Quelle ↪ [ITS97], Seite 134.

Abbildung 2.1: Java — wunderschöne Insel Indonesiens

Kontinuierlich reifte Java^{TM} zu einem leistungsfähigen Werkzeug (*Java Development Kit*, kurz: JDK) für die Entwicklung objekt-orientierter Software in einem breiten Anwendungsspektrum. Von der Chip-Karte über den Personalcomputer und die Hochleistungsworkstation (Server) bis hin zum Großrechner (*Host*) ist Java^{TM} nutzbar; sei es für neue Projekte oder zur Anpassung von „Altsystemen“ an die gestiegenen Benutzerforderungen wie beispielsweise Ortsunabhängigkeit und graphische Benutzerführung.

Aus dem ursprünglichen *Object Application Kernel* (OAK), der urheberrechtlich bedingt einfach in Java^{TM} umgetaufte wurde [Hist97], ist eine ganze Java-Technologie entstanden. Die derzeitige Basisplattform zur Entwicklung für betriebswirtschaftlich relevante Software, also besispielsweise für sogenannte *Enterprise Infomation Systems* (EIS) und/oder *Enterprise Resource Planing Systems* (ERP) besteht im Wesentlichen aus zwei Teilen:

J2SE SDK Java 2 SDK (*Software Developer's Kit*), Standard Edition
Zum Beispiel: `java version "1.4.0_01"`



Legende:

IDE	*Integrated Development Environment*
JRE	*Java Runtime Environment*
SDK	Software Developer's Kit

Abbildung 2.2: J2SE: Skizze der Komponenten

JRE Java 2 Runtime Environment, Standard Edition
Zum Beispiel: `Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_01-b03) und Java HotSpot (TM) Client VM (build 1.4.0_01-b03, mixed mode)`

`J2SE SDK`, früher als JDK bezeichnet, bietet die Sprachfunktionalität von Java^{TM} und die Kernbibliotheken, die für die „übliche" Anwendungsentwicklungen notwendig sind (↪ Abbildung 2.2 S. 29). Diese Kernklassen sind in den Paketen `java.*` enthalten. Zusätzlich bietet `J2SE SDK` Bibliotheken für Erweiterungen. Diese stehen als `javax.*`-Pakete zur Verfügung (↪ Tabelle 2.2 S. 30).

2.2 J2EE

Das `J2SE SDK` liegt der `J2EE`-Architektur zugrunde. Diese besteht aus den folgenden Schichten:

Pakete	Aufgabe
`java.io.*`	Ein-/Ausgabe
`java.awt.*` `java.swing.*`	Graphische Benutzungs-oberfläche
`java.sql.*`	Datenbankzugriff
`java.security.*`	Sicherheit
`javax.naming.*`	Verzeichniszugriff
`javax.rmi.CORBA.*`	* CORBA

Legende:
Grober Überblick über `java version "1.4.0_01"`

Tabelle 2.2: J2SE: Pakete und ihre Aufgabe

Client-**Schicht** Sie präsentiert die Daten und realisiert die Benutzerinteraktion. Sie wird daher auch als Präsentationsschicht bezeichnet. Unterstützt werden verschiedene Typen, beispielsweise Java-Applets, Java-Applications und HTML-Clients.

Web-**Schicht** Sie nimmt die Interaktionen der Präsentationsschicht entgegen und generiert die Darstellungslogik. Einmalanmeldung, Sitzungsverwaltung, Inhaltserstellung, -formatierung und -bereitstellung sind Aufgaben dieser Sicht. Softwaretechnisch sind *Java Server Pages* (JSP) und *Java Servlets* bedeutsam.

Geschäfts-**Schicht** Sie behandelt die Geschäftslogik und dient als Schnittstelle zu den Geschäftskomponenten, die üblicherweise als Komponenten mit Unterstützung durch einen EJB-Container (*Enterprise JavaBeans*) implementiert sind, der den Lebenszyklus der Komponenten unterstützt und die Persistenz, Transaktionen und Ressourcenzuweisung verwaltet.

EIS-**Schicht** Die *Enterprise Information System*-Schicht verknüpft J2EE-Anwendungen mit „Altsystemen“(NICHT-J2EE-Anwendungen), den sogenannten *Legacy*-Systemen. Softwaretechnisch sind *Java Message Service* (JMS), *Java Database Connectivity* (JDBC) und Connector-Komponenten bedeutsam.

Ressourcen-**Schicht** Diese Schicht umfaßt die *Data Base Manangement Systems* (DBMS),

das heißt die Datenbanken, Daten und externe Dienste. Sie gewährleistet die Persistenz von Daten.

2.3 APIs & Standarddienste

J2EE umfasst folgende wichtige *Application Programming Interfaces* (APIs) und Standarddienste:

HTTP(S) Clients können mit dem Paket `java.net.*` das Standardprotokoll *Hypertext Transfer Protocol* der Web-Kommunikation nutzen. Auch das Protokoll *Secure Socket Layer* (SSL) ist verfügbar. Damit ist analog zu HTTP auch HTTPS nutzbar.

JDBC *Java Database Connectivity* ist ein API um herstellerunabhängig auf *Data Base Management Systems* (DBMS) zugreifen zu können.

JMS Mit dem *Java Message Service* ist eine asynchrone Kommunikation zwischen Anwendungen möglich und zwar in einem Netzwerk, das auf *Message-Oriented Middleware* (MOM) Produkten basiert.

JNDI Das *Java Naming and Directory Interface* ermöglicht es auf unterschiedliche Typen von Namens- und Verzeichnisdienste zuzugreifen. JNDI dient zum Registieren und Suchen von (Geschäfts-)-Komponenten. Dazu dienen das *Lightweigth Directory Access Protocol* (LDAP), der *CORBA*[3] *Object Service* (COS) und die RMI-Registrierung.

Java RMI-IIOP Die *Remote Methode Invocation* (RMI) in Verbindung mit CORBA-IIOP (*Internet Inter-Operability Protocol*) ermöglicht mit CORBA-kompatiblen Clients, die nicht in Java^{TM} geschrieben sein müssen, zu kommunizieren.

JTA Das *Java Transaction API* ermöglicht Transaktionen zu starten, erfolgreich zu beenden oder abzubrechen. Dabei kommuniziert der Transaktionsmanager mit dem Ressourcenmanager.

[3] CORBA ≡ *Common Object Request Broker Architecture*

Kapitel 3

Eine Welt voller Objekte

Die objekt-orientierte Denkwelt von Java^{TM} schöpft ihre Ideen aus schon lange bekannten Konzepten. Zum Beispiel aus dem Konzept des Polymorphismus (generische Funktion) und der daten- oder muster-gesteuerten Programmierung. Stets geht es dabei um das Meistern von komplexen Systemen mit angemessenem wirtschaftlichen Aufwand.
Die Objekt-Orientierung zielt auf zwei Aspekte:

1. Komplexe Software soll erfolgreich konstruierbar und betreibbar werden.
   ↪ Qualität, Validität

2. Die Erstellungs- und Wartungskosten von komplexer Software sollen gesenkt werden.
   ↪ Wirtschaftlichkeit

Dies soll primär durch eine bessere Verstehbarkeit der Software erreicht werden. Transparenter wird die Software, weil sie unmittelbar die Objekte aus dem Anwendungsfeld in Objekte des Quellcodes abbildet.

Trainingsplan

Das Kapitel „Eine Welt voller Objekte“ erläutert:

- das Paradigma der Objekt-Orientierung,
  ↪ Seite 34 . . .
- die Wurzelen der Objekt-Orientierung und
  ↪ Seite 38 . . .
- die Ausrichtung von objekt-orientierten Programmiersprachen.
  ↪ Seite 41 . . .

3.1 Denkwelt der Objekt-Orientierung

Objekt-Orientierung[1] verkörpert
viel mehr als eine Programmierungstechnik.
Sie ist eine Denkwelt der gesamten Softwareentwicklung!

Bei der Entwicklung einer Anwendung ist Software in einer ausreichenden Qualität zu erstellen. Dabei wird die Qualität von Software bestimmt durch ihre:

1. *Leistungsfähigkeit*,
   das heißt, das Programm erfüllt die gewünschten Anforderungen.
2. *Zuverlässigkeit*,
   das heißt, das Programm arbeitet auch bei ungewöhnlichen Bedienungsmaßnahmen und bei Ausfall gewisser Komponenten weiter und liefert aussagekräftige Fehlermeldungen (Robustheit),

[1] Zur umfassenden Bedeutung der Objekt-Orientierung siehe z. B. ↪ [Broy/Siedersleben02, Jähnichen/Herrmann02].

3. *Durchschaubarkeit & Wartbarkeit*,
   das heißt, das Programm kann auch von anderen Programmierern als dem Autor verstanden, verbessert und auf geänderte Verhältnisse eingestellt werden,

4. *Portabilität & Anpaßbarkeit*,
   das heißt, das Programm kann ohne großen Aufwand an weitere Anforderungen angepaßt werden,

5. *Ergonomie & Benutzerfreundlichkeit*,
   das heißt, das Programm ist leicht zu handhaben,

6. *Effizienz*,
   das heißt, das Programm benötigt möglichst wenig Ressourcen.

Aufgrund der langjährigen Fachdiskussion über die Innovation *Objekt-Orientierung*[2], wollen wir annehmen, daß dieses Paradigma[3] (≈ Denkmodell), etwas besseres ist, als das *was die Praktiker immer schon gewußt und gemacht haben*. Damit stellt sich die Kernfrage: Wenn das objekt-orientierte Paradigma die Lösung ist, was ist eigentlich das Problem? Des Pudels Kern ist offensichtlich das Unvermögen, komplexe Systeme mit angemessenem wirtschaftlichen Aufwand zu meistern. Objekt-Orientierung verspricht deshalb (↪ [Kim/Lochovsky89]),

Komplexität

- einerseits komplexere Systeme erfolgreich konstruieren und betreiben zu können und
- andererseits die Erstellungs- und Wartungskosten von heutigen Systemen zu senken.

Bewirken soll diesen Fortschritt primär eine wesentliche Steigerung der Durchschaubarkeit der Modelle in den Phasen: Anforderungsanalyse, Systemdesign, Programmierung und Wartung. Die Grundidee ist:

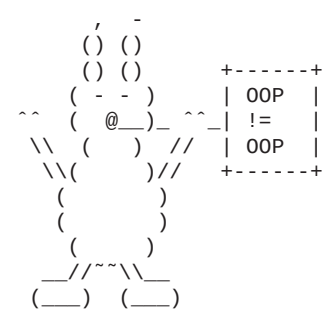
[2] Das Koppelwort *Objekt-Orientierung* ist hier mit Bindestrich geschrieben. Einerseits erleichtert diese Schreibweise die Lesbarkeit, andererseits betont sie Präfix-Alternativen wie zum Beispiel Logik-, Regel- oder Muster-Orientierung.

[3] Ein Paradigma ist ein von der wissenschaftlichen Fachwelt als Grundlage der weiteren Arbeiten anerkanntes Erklärungsmodell, eine forschungsleitende Theorie. Es entsteht, weil es bei der Lösung von als dringlich erkannten Problemen erfolgreicher ist (zu sein scheint) als andere, bisher „geltende“ Ansätze (Kritik am klassischen Paradigma der Softwareentwicklung ↪ [Bonin88]).

Ein „Objekt“ der realen (oder erdachten) Welt bleibt stets erhalten. Es ist über die verschiedenen Abstraktionsebenen leicht verfolgbar. Das gewachsene Verständnis über die Objekte der realen Welt verursacht eine größere Modelltransparenz.

Modell

Was ist ein Objekt im Rahmen der Erarbeitung eines objekt-orientierten Modells? Der Begriff „Modell“ ist mit divergierenden Inhalten belegt. In der mathematischen Logik ist „Modell“ das Besondere im Verhältnis zu einem Allgemeinen: Das Modell eines Axiomensystems ist eine konkrete Inkarnation (≈ Interpretation) dieses Systems. In der Informatik wird der Begriff in der Regel im umgekehrten Sinne verwendet: Das Modell ist das Allgemeine gegenüber einem Besonderen.



Wie können die Objekte in einem konkreten Anwendungsfall erkannt, benannt und beschrieben werden? Das objekt-orientierte Paradigma gibt darauf keine einheitliche, unstrittige Antwort. Es kennt verschiedene bis hin zu konkurrierenden Ausprägungen (zum Beispiel Klassen versus Prototypen). Die jeweiligen Ansätze spiegeln sich in charakteristischen Programmiersprachen wider. Zum Beispiel steht Smalltalk [Goldberg/Robson83, Goldberg83] für die Ausrichtung auf den Nachrichtenaustausch[4] oder CLOS [Bobrow/Moon88] für die inkrementelle Konstruktion von Operationen mittels generischer Funktionen (*Polymorphismus*).

Zur Modellbeschreibung nach einem beliebigen Paradigma dienen in der Praxis Bilder aus beschrifteten Kästchen, Kreisen und Pfeilen (≡ gerichteten Kanten eines Graphen). Die gemalten Kästchen (≡ Boxen) sind Schritt für Schritt zu präzisieren. Boxen sind durch neue Boxen zu verfeinern, das heißt es entstehen weitere Kästchen kombiniert mit Kreisen und Pfeilen. Überspitzt formuliert: Die Lehre von der Boxen-Zeichnerei ist wesentlicher Bestandteil der Modellierung.

Boxologie

Die Informatik kennt schon lange bewährte *Boxologien*[5] im Zu-

[4] engl. message passing

sammenhang mit anderen Modellierungstechniken. Die aus dem griechischen stammende Nachsilbe ... *logie* bedeutet *Lehre, Kunde, Wissenschaft* wie sie zum Beispiel beim Wort Ethnologie (≡ Völkerkunde) zum Ausdruck kommt. Boxologie ist einerseits spassig-provokativ gemeint. Andererseits soll damit die Wichtigkeit aussagekräftiger Zeichnungen hervorgehoben werden.

Im imperativen Paradigma umfaßt die Boxologie zum Beispiel Ablaufpläne oder Struktogramme. Geht es um nicht-prozedurale Sachverhalte (zum Beipiel Kausalitäten oder Prädikatenlogik), dann bietet die allgemeine Netztheorie vielfältige Darstellungsmöglichkeiten. So sind zum Beispiel mit (Petri-)Netzen nebenläufige Vorgänge gut visualisierbar.

Sollen die Vorteile einer objekt-orientierten Modellierung in der Praxis zum Tragen kommen, dann gilt es, auch ihre spezifische Boxologie im Hinblick auf Vor- und Nachteile zu analysieren, das heißt die Fragen zu klären: Welche Erkenntnis vermitteln ihre vielfältigen Diagramme, beispielsweise Diagramme vom Typ:

Diagramme

- Anwendungsfalldiagramm,
- Klassendiagramm,
- Verhaltensdiagramm und
- Implementationsdiagramm.

Die Vorteile der objekt-orientierten Modellierung sind nicht in allen Anwendungsfeldern gleich gut umsetzbar. Soll Objekt-Orientierung nicht nur eine Marketing-Worthülse sein, sondern der erfolgreiche Modellierungsansatz, dann bedarf es zunächst einer Konzentration auf dafür besonders prädestinierte Automationsaufgaben. Wir nennen daher Kriterien, um das Erkennen solcher bevorzugten OO-Aufgaben zu unterstützen (↪ Abschnitt 3.3 S. 45). Als Kriterien dienen einerseits die prognostizierte Menge der Objekte und andererseits ihre Komplexität und Struktur.

[5] Angemerkt sei, daß diese Boxologie nichts mit den Boxern im Sinne von Faustkämpfern oder im Sinne des chinesischen Geheimbundes um 1900 zu tun hat. Auch die Box als eine einfache Kamera oder als Pferdeunterstand sind hier keine hilfreiche Assoziation.

3.2 Wurzeln der Objekt-Orientierung

Java^{TM}, ein (beinahe)[6] strikter objekt-orientierter Ansatz, schöpft viele Ideen aus den drei klassischen Konzepten:

1. **Polymorphismus** (generische Funktion)
2. **Daten-gesteuerte Programmierung** (explizites „dispatching“)
3. **Muster-gesteuerter Prozeduraufruf**[7]

Stets geht es dabei um die Technik einer separat, inkrementell definierten Implementation und automatischen Wahl von Code.

3.2.1 Polymorphismus

Ein polymorpher Operator ist ein Operator, dessen Verhalten oder Implementation abhängt von der Beschreibung seiner Argumente.[8] Java^{TM} unterstützt separate Definitionen von polymorphen Operationen (Methoden), wobei die einzelnen Operationen denselben Namen haben, aber Argumente mit verschiedenen Typen abarbeiten. Die Operationen selbst werden als unabhängig voneinander behandelt. Zum Beispiel kann man eine Methode `groesse()` konstruieren, die definiert ist für ein Argument vom Typ `String` (Zeichenkette) und eine weitere ebenfalls mit dem Namen `groesse()` für ein Argument vom Benutzer-definierten Typ `Jacke`. Das Java-System wählt dann die passende Implementation basierend auf den deklarierten oder abgeleiteten Typen der Argumente im aufrufenden Code.

[6] Nur beinahe, weil beispielsweise die arithmetischen Grundoperationen, wie $1 + 1$, nicht objekt-orientiert notiert werden. Objekt-orientiert wäre dann zu notieren: `1.add(1)`. Mehr dazu siehe Abschnitt 10 S. 391.

[7] engl. pattern directed procedure invocation

[8] Schon die alte Programmiersprache FORTRAN kennt polymorphe arithmetische Operatoren. In FORTRAN hängt der Typ des Arguments vom ersten Buchstaben des Namens der Variablen ab, wobei die Buchstaben $I, \ldots, N$ auf Fixpunktzahlen, während die anderen auf Gleitkommazahlen verweisen. Zur Compilierungszeit wird die passende Operation anhand der Typen der Argumente ausgewählt. Während FORTRAN dem Programmierer nicht ermöglicht, solche polymorphen Operationen selbst zu definieren, ist dies in modernen objekt-orientierten Sprachen, wie zum Beispiel in Java^{TM} oder `C++` (↪ [Stroustrup86, Stroustrup89]) möglich.

[Hinweis: Die folgende Java-Notation mag für Java-Neulinge noch unverständlich sein. Sie kann von ihnen sorglos überlesen werden. Die „Java-Berührten“ verstehen mit ihr die polymorphen Operationen jedoch leichter.]

```
...
public int groesse(String s) {...}
...
public int groesse(Jacke j)  {...}
...
foo.groesse(myString);
...
foo.groesse(myJacke);
...
```

Entscheidend für objekt-orientierte Ansätze ist die Frage, zu welchem *Zeitpunkt* die Auswertung der Typen der Argumente vollzogen wird. Prinzipiell kann es zur Zeit der Compilierung (*statisches Konzept*) oder zur Laufzeit (*dynamisches Konzept*) erfolgen. Java^{TM} unterstützt den Compilierungszeit-Polymorphismus. Smalltalk ist beispielsweise ein Vertreter des Laufzeit-Polymorphismus.

Zeitpunkt

3.2.2 Daten-gesteuerte Programmierung

Daten-gesteuerte Programmierung ist keine leere Worthülse oder kein Pleonasmus[9], weil offensichtlich Programme von Daten gesteuert werden. Zumindest aus der Sicht der Mikroprogramme eines Rechners sind ja alle Programme selbst Daten. Hier geht es um die Verknüpfung von passiven Systemkomponenten (≡ anwendungsspezifischen Daten) mit aktiven Systemkomponenten (≡ anwendungsspezifischen Operationen).

Insbesondere bei Anwendungen im Bereich der Symbolverarbeitung haben sich im Umfeld der Programmierung mit LISP (LISt Processing) verschiedene Ausprägungen der sogenannten *daten-gesteuerten Programmierung*[10] entwickelt (↪ [Abelson85]). Der Programmierer definiert selbst eine *dispatch*-Funktion[11], die den Zugang zu den da-

[9] Als Pleonasmus wird eine überflüssige Häufung sinngleicher oder sinnähnlicher Ausdrücke bezeichnet.

[10] engl. „data-directed programming“ oder auch „data-driven programming“

[11] engl. dispatch ≡ Abfertigung

tentypabhängigen Operationen regelt. Die Idee besteht darin, für jeden anwendungsspezifischen Datentyp ein selbstdefiniertes Symbol zu vergeben. Jedem dieser Datentyp-Symbole ist die jeweilige Operation als ein Attribut zugeordnet.[12] Quasi übernehmen damit die „Datenobjekte" selbst die Programmablaufsteuerung. Erst zum Zeitpunkt der Evaluation („Auswertung") eines Objektes wird die zugeordnete Operation ermittelt (↪ [Bonin91b]).

3.2.3 Muster-gesteuerter Prozeduraufruf

Ziel: Paßt!

Ist die Auswahl der Operation abhängig von mehreren Daten im Sinne eines strukturierten Datentyps, dann liegt es nahe, das Auffinden der Prozedur als einen Vergleich zwischen einem Muster und einem Prüfling mit dem *Ziel: Paßt!* zu konzipieren. Ein *allgemeingültiger Interpreter*, der dieses Passen feststellt, wird dann isoliert, das heißt aus dem individuellen Programmteil herausgezogen.

Da eine Datenbeschreibung (zum Beispiel eine aktuelle Menge von Argumenten) prinzipiell mehrere Muster und/oder diese auf mehr als einem Wege entsprechen könnte, ist eine Abarbeitungsfolge vorzugeben. Die *Kontrollstruktur* der Abarbeitung ist gestaltbar im Sinne eines Beweises, das heißt sie geht aus von einer Zielthese und weist deren Zutreffen nach, oder im Sinne einer Zielsuche, das heißt sie verfolgt einen Weg zum zunächst unbekannten Ziel. Anhand der Kontrollstruktur kann die Verknüpfung von Daten und Prozedur mittels der Technik des Mustervergleichs[13] als Vorwärtsverkettung[14] und/oder als Rückwärtsverkettung[15] erfolgen.

Aus der Tradition des muster-gesteuerten Prozeduraufrufs sind Schritt für Schritt leistungsfähige regel- und logik-orientierte Sprachen mit Rückwärtsverkettung[16], wie zum Beispiel PROLOG[17], entstanden.

[12] LISP ermöglicht diese Attributzuordnung über die Eigenschaftsliste, die LISP für jedes Symbol automatisch führt.

[13] engl. „pattern matching"

[14] engl. „forward chaining"

[15] engl. „backward chaining"

[16] engl. backward chaining rule languages

[17] *PROgramming in LOGic* ↪ [Belli88, Clocksin/Mellish87]

Schwerpunkte der Objekt-Orientierung			
		I *Klassen-/ (Daten)Typ-Betonung*	II *Prototyp-/ Einzelobjekt-Betonung*
A	*Betonung der Operations-konstruktion*	CLOS (**Java**) (C++)	(Daten-gesteuerte Programmierung) PROLOG
B	*Betonung des Nachrichten-austausches*	Smalltalk (**Java**) (C++)	SELF (Actor-Sprachen)

Legende:
Actor-Sprachen ↪ [Lieberman81]
C++ ↪ [Stroustrup86, Stroustrup89]
CLOS (*C*ommon *L*isp *O*bject *S*ystem) ↪ [Bobrow/Moon88]
Daten-gesteuerte Programmierung ↪ [Abelson85]
Java^{TM} ↪ [Arnold/Gosling96]
SELF ↪ [Ungar/Smith91]
Smalltalk ↪ [Goldberg/Robson83, Goldberg83]
PROLOG (*PRO*gramming in *LO*gic) ↪ [Belli88, Clocksin/Mellish87]

Tabelle 3.1: Ausrichtung von objekt-orientierten Ansätzen

3.3 Ausrichtung objekt-orientierter Sprachen

Bei einem groben, holzschnittartigen Bild von objekt-orientierten Sprachen ist die Zusammenfassung der passiven Komponente (≡ Daten) mit der aktiven Komponente (≡ zugeordnete Operationen) zum Objekt verknüpft mit dem Konzept des Nachrichtenaustausches. Dabei enthält die Nachricht, die an ein Objekt, den Empfänger, gesendet wird, den Namen der Operation und die Argumente. Entsprechend der *daten-gesteuerten Programmierung* dient der Empfänger und der Name der Operation zur Auswahl der auszuführenden Operation, die üblicherweise Methode genannt wird. So gesehen ist die Objekt-Orientierung, wie folgt, definierbar:

passiv

aktiv

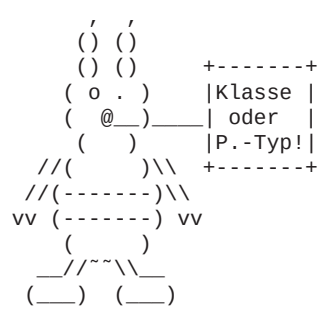
Objekt-Orientierung ≡ Datentypen
+ Nachrichtenaustausch

Das Bild von Objekten, die miteinander Nachrichten austauschen, verdeutlicht nur einen Schwerpunkt der Objekt-Orientierung. Verfolgt man die Polymorphismuswurzel, dann rückt nicht der Transfer von Nachrichten, sondern die Konstruktion von generischen Funktionen in den Fokus. Kurz: Die Konstruktion der Operation wird betont.

Es scheint trivial zu sein, festzustellen, daß bei objekt-orientierten Sprachen neue Objekte aus existierenden Objekten konstruiert werden. Die Konzepte der „Ableitung“ der neuen, also der anwendungsspezifischen Objekte unterscheiden sich jedoch. Gemeinsame Eigenschaften einzelner Objekte können zu einer Klasse oder zu einem charakteristischen Einzelobjekt zusammengefasst werden. Oder umgekehrt: Ein konkretes Objekt kann sich aus *Klasseneigenschaften* ergeben oder es kann bezugnehmen auf einen *Prototyp* (ein anderes Einzelobjekt).

Klasse

Das Klassenkonzept unterstellt von Anfang an eine größere Zahl von Objekten, die gleiche Eigenschaften (Struktur und Verhalten) aufweisen. Als Beispiel nehmen wir mehrere Ausgabekonten an, wobei jedes einzelne Ausgabekonto einen SOLL-Wert aufweist. Die Klasse `Ausgabekonto` beschreibt das für alle Ausgabekonten gleiche Merkmal SOLL-Wert. Wird ein einzelnes Ausgabekonto erzeugt, so „erbt“ es aus der Klasse `Ausgabekonto` die Definition SOLL-Wert. Da die Klassendefinition selbst wiederum auf abstraktere Klassen zurückgreift, zum Beispiel auf eine Klasse `Haushaltskonto`, umfaßt das Klassenkonzept mehrere Abstraktionsebenen. Das Objekt entsteht aus einer Hierarchie von Beschreibungen auf verschiedenen Abstraktionsebenen.



Beim Konzept mit Prototypen unterstellt man ein charakteristisches konkretes Objekt mit möglichst umfassend vordefinierten Eigenschaften. Beim Definieren der einzelnen Konten nimmt man Bezug auf diese Beschreibung des charakteristischen Einzelobjekts. Ein übliches Ausgabekonto ist zunächst detailliert zu beschreiben. In unserem Kontenbeispiel wäre zunächst ein Ausgabekonto, vielleicht ein Konto 1203/52121, zu beschreiben und zwar einschließlich seiner Eigenschaft SOLL-Wert.

Prototyp

Die einzelnen Ausgabekonten enthalten den Bezug auf und gegebenenfalls die Abweichung von diesem Prototypen 1203/52121. Dieses Prototypmodell unterstützt vergleichsweise weniger das Herausarbeiten von mehreren Abstraktionsebenen. Einerseits entfällt die Suche nach Eigenschaften und Namen der höheren Abstraktion für unser Kontensystem. Andererseits ist die Wiederverwendbarkeit geringer.

Die Tabelle 3.1 S. 41 zeigt die vier skizzierten Schwerpunkte als die zwei Dimensionen: Klassen / Prototyp und Operation / Nachrichtenversand (ähnlich [Gabriel91]). Eine schlagwortartige Übersicht (↪ Tabelle 3.2 S. 44) vergleicht Java^{TM} mit , Smalltalk und CLOS. Ausgewählt wurden diese OO-Sprachen aufgrund ihrer unterschiedlichen OO-Konzepte.

Die Option eines Benutzer-definierten Polymorphismus zur Compilierungszeit ist in Zukunft sicherlich auch in weit verbreitete Programmiersprachen wie COBOL, BASIC, Pascal oder FORTRAN einbaubar. Damit können jedoch solche ursprünglich imperativ-geprägten Sprachen mit dieser Form aufgepfropfter Objekt-Orientierung nicht die volle Flexibilität ausschöpfen, wie sie zum Beispiel von CLOS geboten wird mit *Multiargument Polymorphismus zur Laufzeit*.

CLOS

Objekt-orientierte Modelle sind geprägt durch eine (Un)Menge von Definitionen, die Abstraktionen (≡ Klassen) von „realen“ Einheiten (≡ Objekten) beschreiben. Da Objektklassen von anderen Klassen Eigenschaften (interne Variablen, Methoden) *erben* können, lassen sich aus allgemeinen Objekten spezielle anwendungsspezifische konstruieren. Ver-

Terminologie (Leistungen) bei der Objekt-Orientierung				
	Aspekte[1]	I **Java**	II **Smalltalk**	III **CLOS**
1	Abstraktion	class		
2	Objekt	member of class	instance	
3	Ver-erbung	superclass		direct superclasses
		subclass		
4	Objekt-struktur	member data elements	instance variables	slots
5	Struktur-beschrei-bung (*Slot*)	name		
		type initial value forms		initial value forms, accessor functions, initi-alizer keywords
6	Opera-tionsaufruf	calling a method	sending a message	calling a generic function
7	Opera-tionsimple-mentation	method		
8	Verknüpfung Klasse mit Operation	defined in class scope	through class browser	specializers in parameter list
9	Multiargument „Dispatch“	chained dispatch		multi-methods
10	Referenz zum Objekt	this	self	name in parameter list
11	Auruf der verdeckten Operation	call method with qualified name	message to super	call-next-method
12	Direktzu-griff auf interen Objektzustand	by name within class scope	by name within method scope	slot-value
13	Allgemeiner Zugriff auf internen Objektzustand	user methods		
		public declaration		accessor functions
14	Operations-kombination	Nein		standard method combination

Legende:
[1] ähnlich [Kczales91] S. 253.

Tabelle 3.2: Java im Vergleich mit Smalltalk und CLOS

erbungsgraphen, gezeichnet mit beschrifteten Kästchen, Kreisen und Pfeilen, bilden dabei das Modell ab. (Ein Beispiel zeigt ↪ Abbildung 4.11 S. 68.) Die Erkenntnisgewinnung und -vermittlung geschieht primär über Bilder, die aus vielen verknüpften „Boxen“ bestehen.

Entscheidungsträger in der Praxis fordern Handlungsempfehlungen. Welche Anwendungsfelder sind für die Objekt-Orientierung in der Art und Weise von Java^{TM} heute besonders erfolgversprechend? Offensichtlich ist eine Antwort darauf risikoreich. Die Einflußfaktoren sind vielfältig, und die Prioritäten bei den Zielen divergieren.

Wir begrenzen unsere Antwort auf den Fall, daß die Objekt-Orientierung den gesamten Softwarelebenszyklus umfaßt. Es soll kein Paradigmenwechsel zwischen Analyse-, Entwurfs- und Implementationsphase geben. Eine vielleicht erfolgversprechende Kombination zwischen objekt-orientiertem Entwurf und imperativer Programmierung bleibt hier unberücksichtigt.

Effizienz

Da auch der Programmcode objekt-orientiert ist, ist die erreichbare (Laufzeit-)Effizienz ebenfalls ein Entscheidungskriterium. Müssen wir mangels heute schon bewährter objekt-orientierter Datenbank-Managementsysteme erst die Objekte aus den Daten bei jedem Datenbankzugriff erzeugen und beim Zurückschreiben wieder in Tabellen (Daten) konvertieren, dann entstehen zumindest bei komplex stukturierten Objekten Laufzeitprobleme. Die Objektmenge und die Komplexität der Objektstruktur sind relevante Kriterien, um sich für die Objekt-Orientierung entscheiden zu können (Tabelle 3.3 S. 46). Die Laufzeit-Effizienz betrifft zwei Aspekte:

1. den **Nachrichtenaustausch** zwischen den Objekten und

2. das **Propagieren von Modifikationen** der Objekt-Eigenschaften, des Vererbungsgraphen und gegebenenfalls der Vererbungsstrategie.

Rüstzeit

Beide Aspekte sind nur bei Objektmengen einigermaßen effizient beherrschbar, bei denen alle betroffenen Objekte sich im Arbeitsspeicher des Rechners befinden. Ist die Menge der Objekte jedoch so groß, daß Objekte auf externen Speichern zu führen sind, dann ist die *Konvertierungszeit* von Objekten in Daten(relation)en und umgekehrt ein erheblicher Zeitfaktor. Erfreulicherweise kann bei manchen Anwendungen die-

Aspekte für die Beurteilung von OO-Anwendungsfelder				
„Daten“-**Struktur**			„Daten“-**Menge** (Datenvolumen)	
		gleich-artige Fälle	I *relativ beschränkt* (ladbar in Arbeits-speicher)	II *sehr groß* (DBMS erforder-lich)
einfach	A_1	wenige	[0] OO → Effi-zienz-ver-lust	[-] Objekt ⇔ Datensatz Konver-tierung
	A_2	viele		
komplex	B_1	wenige	[+] OO plus XPS	[–] ohne OO-DBMS nicht mach-bar
	B_2	viele	[++] **OO**	

Legende:

Bewertungsskala:

[–]	ungeeignet
[-]	kaum geeignet
[0]	machbar
[+]	geeignet
[++]	sehr gut geeignet

DBMS	≡	Datenbank-Managementsystem
OO	≡	Objekt-Orientierung
XPS	≡	Expertensystem-Techniken (z.B. Regel-Orientierung)

Tabelle 3.3: Aspekte Menge und Struktur

se Konvertierung im Zusammenhang mit der Start- und Beendigungsprozedur erfolgen. Solche terminierbaren Rüstzeiten können häufig akzeptiert werden.

Bei wirklich großen Objektmengen ist auch das Propagieren der Modifikationen auf alle betroffenen Objekte zeitkonsumptiv, insbesondere wenn der Zugriff auf Objekte erst die Konvertierung von Daten in Objekte umfaßt. Daher ist zum Beispiel ein objekt-orientiertes Einwohnerwesen für eine Großstadt nur bedingt zu empfehlen, solange der Nachrichtenaustausch und das Propagieren von Modifikationen mit Konvertierungsvorgängen belastet sind. In diesem Kontext reicht es nicht aus, wenn wir die Laufzeit-Effizienz durch eine Typprüfung zur Compilierungszeit steigern und zum Beispiel auf einen Laufzeit-Polymorphismus (↪ Abschnitt 3.2.1 S. 38) ganz verzichten. Nicht nur der Konvertierungsaufwand ist offensichtlich abhängig von der Komplexität der Objektstruktur, sondern auch die Konstruktion und Abarbeitung eines Vererbungsgraphens.

Stück-Kosten

Ist ein umfangreicher, tiefgeschachtelter Graph mit relativ wenigen „Blättern“ (Instanzen) pro Knoten zu erwarten, dann ist der Konstruktions- und Abarbeitungsaufwand pro Instanz hoch (Stückkosten!). Alternativen mit geringerem Abbildungsaufwand kommen in Betracht. Neben dem Prototyp-Ansatz ist der Übergang auf die Strukturierung im Sinne der Expertensystem-Technik (XPS) eine zweckmäßige Alternative. Die Modularisierung erfolgt primär in handhabbaren „Wissensbrocken“, die ein allgemeingültiger „Interpreter“ auswertet. Verkürzt formuliert: In Konkurrenz zur Welt aus kommunizierenden Objekten treten XPS mit Regel-Orientierung.

Kapitel 4

Modellieren mit UML

Die Qualität eines Programms hängt von der Qualität der Modelle ab, die die Objekte der Anwendungswelt sowie die (Benutzer-)-Anforderungen abbilden. Erst die „richtigen“ Objekte mit den „richtigen“ Beziehungen führen zum gelungenen Programm. Kurz: Fehler im Modell gleich Fehler im Programm! Für den Transfer der „richtigen“ Objekte und Beziehungen in den Quellcode werden die erforderlichen Modelle in *Unified Modeling Language* (UML) notiert.

UML ist eine Sprache zum Spezifizieren, Visualisieren, Konstruieren und Dokumentieren von „Artifakten eines Softwaresystems“. Mit UML können Geschäftsvorgänge gut modelliert werden. Zusätzlich stellt UML eine Sammlung von besten Ingenieurpraktiken und Mustern dar, die sich erfolgreich beim Modellieren großer, komplexer Systeme bewährt haben.

Trainingsplan

Das Kapitel „Modellieren mit UML“ erläutert:

- die verschiedenen Arten von UML Diagrammen,
  ↪ Seite 50 . . .
- die verschiedenen Typen von Klassen mit ihren Varaibalen und Methoden,
  ↪ Seite 52 . . .
- die Verbindung zwischen Klassen in Form der Assoziation,
  ↪ Seite 57 . . .
- die Beziehungen zwischen dem Ganzen und seinen (Einzel-)Teilen,
  ↪ Seite 61 . . .
- die Vererbung von Eigenschaften (Variablen und Methoden),
  ↪ Seite 64 . . .
- gibt Empfehlungen für die Namensvergabe für UML-Elemente und
  ↪ Seite 68 . . .
- skizziert die Weiterentwicklung von UML.
  ↪ Seite 69 . . .

4.1 UML-Boxologie

Für die objekt-orientierte Modellierung, also für Analyse und Design, wurden zu Beginn der 90-Jahre eine Menge von Methoden mit ihren speziellen Notationen (Symbolen) bekannt. Exemplarisch sind beispielsweise zu erwähnen:

- G. Booch; *Objekt-oriented Analysis and Design with Applications* ↪ [Booch94]
- D. W. Embley u. a.; *Object-Oriented Systems Analysis – A Model-Driven Approach* ↪ [Embley92]
- I. Jacobsen u. a.; *Object-oriented Software Engineering, A Use Case Driver Approach* ↪ [Jacobsen92]
- W. Kim u. a.; *Object-Oriented Concepts, Databases, and Applications* ↪ [Kim/Lochovsky89]
- J. Rumbaugh u. a.; *Object-oriented Modelling and Design* ↪ [Rumbaugh91]

Rational

Die *Unified Modeling Language* (UML) wurde von der *Rational Software Corporation* (↪ [Rational97]) aus solchen Methoden und Notationen entwickelt. UML ist eine Sprache zum

- **Spezifizieren**,
- **Visualisieren**,
- **Konstruieren** und
- **Dokumentieren**

von Artifakten eines Softwaresystems. UML ist besonders gut geeignet für die Modellierung von Geschäftsvorgängen (*business modeling*). Zunehmend gwinnt UML auch Bedeutung für die Modellierung von anderen Nicht-Software-Systemen. Die große Verbreitung von UML hat zu einer umfangreichen Sammlung von besten Ingenieurpraktiken und Mustern geführt, die sich erfolgreich beim Modellieren auch sehr komplexer Systeme bewährt haben. Im Kern umfaßt UML die folgenden Diagramme (*Orginalnamen*):

Diagramme

1. Anwendungsfalldiagramm (*use case diagram*)
2. Klassendiagramm (*class diagram*)
3. Verhaltensdiagramme (*behavoir diagrams*):
   (a) Zustandsdiagramm (*statechart diagram*)

   (b) Aktivitätsdiagramm (*activity diagram*)
   (c) Interaktionsdiagramme (*interaction diagrams*):
      i. Sequenzdiagramm (*sequence diagram*)
      ii. Kollaborationsdiagramm (*collaboration diagram*)
4. Implementierungsdiagramme (*implementation diagrams*):
   (a) Komponentendiagramm (*component diagram*)
   (b) Einsatzdiagramm (Knoten[1]) (*deployment diagram*)

Diese Diagramme ermöglichen verschiedene Sichten auf das System und zwar besonders aus den Perspektiven der Analyse und des Designs. Aufgrund der konsequenten Objekt-Orientierung unterstützt UML beispielsweise keine Datenflußdiagramme, weil nicht Daten und deren Fluß durch das Programm sondern Objekte und deren Kommunikation zur objekt-orientierte Denkwelt gehören.

4.2 Basiselement: Klasse

4.2.1 Beschreibung

class

Eine Klasse definiert die Eigenschaften ihrer Objekte mit Hilfe von Variablen (Attributen) und Methoden (Operationen). Darüberhinaus kann diese Definition auch Zusicherungen, Merkmale und Stereotypen umfassen. Tabelle 4.1 S. 53 zeigt die UML-Notation einer Klasse.

[Hinweis: Ein verwandter Begriffe für die Klasse ist der Begriff (Daten-)Typ. In Java^{TM} ist ein einfacher Datentypen wie zum Beispiel `float` von einem „zusammengesetzten“ Datentype (*ReferenceType*) wie zum Beispiel `FahrzeugProg` (↪ Abschnitt 6.1.3 S. 94) zu unterscheiden.]

Beschreibung: Abstrakte Klasse

Eine abstrakte Klasse ist eine Klasse, die die Basis für weitere Unterklassen bildet. Eine abstrakte Klasse hat keine Mitglieder (*member*), al-

[1] Ein Knoten ist ein Computer, also ein zur Laufzeit vorhandenes Gerät (Objekt), daß über Speicher und Prozessorleistung verfügt.

Klassensymbol:

Klassensymbol mit Variablen und Methoden:

≪stereotyp≫ **paket**::**Klasse** {merkmal}
variable : Typ=initialwert {merkmal} {zusicherung}
methode(param : Typ=initialwert) : Rückgabetyp {merkmal} {zusicherung}

Beispiel: `Fahrzeug` (↪ Abschnitt 6.1.3 S. 94)

≪fachklasse≫ **de.fhnon.fahrzeug**::**Fahrzeug** {public, Version=1.0}
geschwindigkeit : double=0 {geschwindigkeit ≥ 0} fahrtrichtung : long=0 {fahrtrichtung ≥ 0} eigentuemer : String=" " anzahl : long=0 {anzahl ≥ 0}
≪getMethode≫ getAnzahl() : long getGeschwindigkeit() : double getFahrtrichtung() : long getEigentuemer() : String ≪konstuktor≫ Fahrzeug() Fahrzeug(eigentuemer: String) Fahrzeug(geschwindigkeit: double, fahrtrichtung : long, eigentuemer: String) ≪bewegung≫ wohin(g : double, f : long) : Fahrt wohin(fahrzeugFahrt : Fahrt) beschleunigt(g : double) : double

Tabelle 4.1: UML-Basiselement: Klasse

so keine Instanzen (Objektexemplare). Sie wird als eine (normale) Klasse mit dem Merkmal {`abstract`} notiert[2].

Beschreibung: Metaklasse

Eine Metaklasse dient zum Erzeugen von Klassen. Sie wird wie eine (normale) Klasse notiert und erhält den Stereotyp ≪metaclass≫[3].

Beschreibung: Variable (Attribut)

Variable

Ein Variable benennt einen Speicherplatz in einer Instanz (≡ Instanzvariable) oder in der Klasse selbst (≡ Klassenvariable).

variable : Typ=initialwert {merkmal} {zusicherung}

[Hinweis: Verwandte Begriffe für die Variable sind die Begriffe Attribut, *Member*, *Slot* und Datenelement.]

Mit Hilfe eines der Sonderzeichen „+“, „−“ und „#“, das dem Namen der Variablen vorangestellt wird, kann eine Variable in Bezug auf ihre Sichtbarkeit besonders gekennzeichnet werden. Ein vorangestellter Schrägstrich (*slash*) gibt an, daß der Wert der Variable von anderen Variablen abgeleitet ist. Handelt es sich um eine Klassenvariable, dann wird der Name unterstrichen. Tabelle 4.2 S. 55 zeigt diese Möglichkeiten einer Kennzeichnung.

Beschreibung: Methode

Methode

Methoden sind der „aktiv(ierbar)e“ Teil (Algorithmus) der Klasse. Eine Methode wird durch eine Nachricht an eine Instanz aktiviert. Eine Methode kann sich auch auf die Klasse selbst beziehen (≡ Klassenmethode (`static`)). Dann wird sie durch eine Nachricht an die Klasse aktiviert.

```
methode(parameter : ParameterTyp=standardWert) :
        RückgabeTyp {merkmal} {zusicherung}
```

[2] Alternativ zu dieser Kennzeichnung kann der Klassenname auch *kursiv* dargestellt werden.

[3] Näheres zur Programmierung mit Metaobjekten ↪ [Kczales91].

UML-Notation	Erläuterung
+publicVariable	allgemein zugreifbare Variable
−privateVariable	private, nicht allgemein zugreifbare Variable
#protectedVariable	geschützte, bedingt zugreifbare Variable
/abgeleitetesVariable	hat einen Wert aus anderen Variablen
klassenVariable	Variable einer Klasse (`static`)

Hinweis: Sichtbarkeit (Zugriffsrechte) im Java-Kontext ↪ Tabelle 6.7 S. 155.

Tabelle 4.2: Kennzeichnung einer Variablen in UML

UML-Notation	Erläuterung
+publicMethode()	allgemein zugreifbare Methode
−privateMethode()	private, nicht allgemein zugreifbare Methode
#protectedMethode()	geschützte, bedingt zugreifbare Methode
klassenMethode()	Methode einer Klasse (`static`)

Hinweis:
Sichtbarkeit (Zugriffsrechte) im Java-Kontext ↪ Tabelle 6.7 S. 155.

Tabelle 4.3: Kennzeichnung einer Methode in UML

Beispiel:

```
setPosition(x : int=10, y : int=300) :
  boolean {abstract} {(x ≥ 10) ∧ (y ≥ 300)}
```

[Hinweis: Verwandte Begriffe für Methode sind die Begriffe Funktion, Prozedur und Operation.]

Mit Hilfe eines der Sonderzeichen „+“, „−“ und „#“, das dem Namen der Methode vorangestellt wird, kann eine Methode in Bezug auf ihre Sichtbarkeit besonders gekennzeichnet werden. Handelt es sich um eine Klassenmethode, dann wird der Name unterstrichen. Tabelle 4.3 S. 55 zeigt diese Möglichkeiten einer Kennzeichnung.

Beschreibung: Merkmal und Zusicherung

Merkmal Ein Merkmal ist ein Schlüsselwort aus einer in der Regel vorgegebenen Menge, das eine charakteristische Eigenschaft benennt. Ein Merkmal steuert häufig die Quellcodegenerierung.
Beispiele: {`abstract`}, {`readOnly`} oder {`old`}

Zusicherung Eine Zusicherung definiert eine Integritätsregel (Bedingung). Häufig beschreibt sie die zulässige Wertmenge, eine Vor- oder Nachbedingung für eine Methode, eine strukturelle Eigenschaft oder eine zeitliche Bedingung.
Beispiel: `Rechnung.kunde = Rechnung.vertrag.kunde`

Die Angaben von {`zusicherung`} und {`merkmal`} überlappen sich. So kann jedes Merkmal auch als eine Zusicherung angegeben werden.
Merkmal als Zusicherung angegeben — Beispiele:
{`abstract=true`}, {`readOnly=true`} oder {`old=true`}

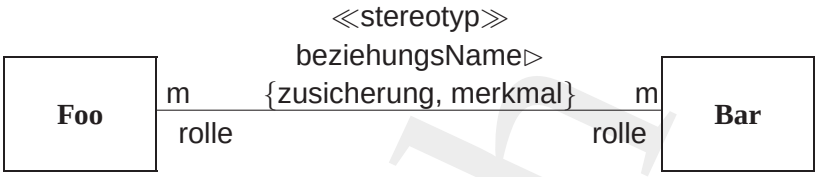
Beschreibung: Stereotyp

Stereotyp Ein Stereotyp ist eine Möglichkeit zur Kennzeichnung einer Gliederung auf projekt- oder unternehmensweiter Ebene. Ein Stereotyp gibt in der Regel den Verwendungskontext einer Klasse, Schnittstelle, Beziehung oder eines Paketes an.
Beispiele: {`fachklasse`}, {`präsentation`} oder {`vorgang`}

[Hinweis: Verwandte Begriffe für den Stereotyp sind die Begriffe Verwendungskontext und Zusicherung.]

4.2.2 Paket von Elementen

Paket Ein Paket beinhaltet eine Ansammlung von Modellelementen beliebigen Typs. Mit Hilfe von Paketen wird ein (komplexes) Gesamtmodell in überschaubarere Einheiten gegliedert. Jedes Modellelement gehört genau zu einem Paket. Ein Paket wird mit dem Symbol eines Aktenregisters dargestellt. Innerhalb des Symbols steht der Name des Paketes. Werden innerhalb des Symbols Aktenregisters Modellelemente genannt,



Legende:

beziehungsName	Name der Assoziation
{zusicherung}	↪ Abschnitt 31 S. 56
{merkmal}	↪ Abschnitt 31 S. 56
m	Multiplizität (↪ Tabelle 4.4 S. 59)
rolle	Sichweise durch das gegenüberliegende Objekt
▷	(Lese)-Richtung für die Beziehung; hier: **Foo**beziehungsName**Bar**

Abbildung 4.1: UML-Beziehungselement: Assoziation

dann steht der Paketname auf der Aktenregisterlasche.
Beispiel: `de.fhnon.fahrzeug`

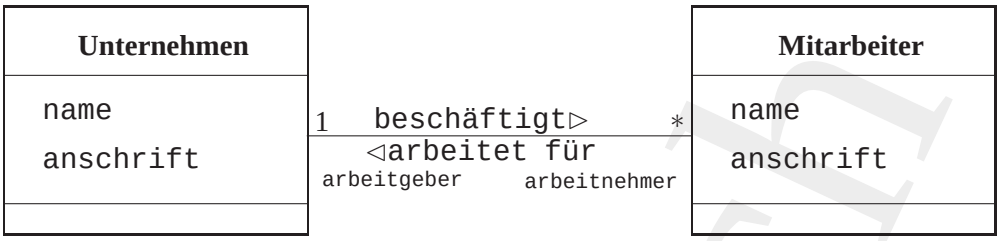
[Hinweis: Verwandte Begriffe für das Paket sind die Begriffe Klassenkategorie, Subsystem und *Package*.]

4.3 Beziehungselement: Assoziation

4.3.1 Beschreibung

Eine Assoziation beschreibt eine Verbindung zwischen Klassen (↪ Abbildung 4.1 S. 57). Die Beziehung zwischen einer Instanz der einen Klasse mit einer Instanz der „anderen“ Klasse wird Objektverbindung (englisch: *link*) genannt. Links lassen sich daher als Instanzen einer Assoziation auffassen. **link**

Häufig ist eine Assoziation eine Beziehung zwischen zwei verschiedenen Klassen (↪ Abbildung 4.2 S. 58). Jedoch kann eine Assoziation auch von rekursiver Art sein, das heißt beispielsweise als Beziehung zwischen zwei Instanzen derselben Klasse formuliert werden (Beispiel ↪ Abbildung 4.3 S. 60) oder eine Assoziation zwischen mehreren Klassen sein. Spezielle Formen der Assoziation sind die Aggregation (↪



Legende:
↪ Abbildung 4.1 S. 57

Abbildung 4.2: Beispiel einer Assoziation: Ein Unternehmen beschäftigt viele Mitarbeiter

Abschnitt 4.4.1 S. 61) und die Komposition (↪ Abschnitt 4.4.2 S. 63).

[Hinweis:Verwandte Begriffe für die Assoziation sind die Begriffe Relation, Aggregation, Komposition, Link und Objektverbindung.]

4.3.2 Multiplizität

min:max

Die Multiplizität m gibt an mit wievielen Instanzen der gegenüberliegende Klasse **Bar** eine Instanz der Klasse **Foo** assoziiert ist[4]. Dabei kann eine Bandbreite durch den Mini- und den Maximumwert angegeben werden (↪ Tabelle 4.4 S. 59).

Ein Assoziation wird in der Regel so implementiert, daß die beteiligten Klassen zusätzlich entsprechende Referenzvariablen bekommen. Im Beispiel „Ein Unternehmen beschäftigt viele Mitarbeiter“ (↪ Abbildung 4.2 S. 58) erhält die Klasse **Unternehmen** die Variable `arbeitnehmer` und die Klasse **Mitarbeiter** die Variable `arbeitgeber`. Aufgrund der angebenen Multiplizität $*$ („viele Mitarbeiter“) muß die Variable `arbeitnehmer` vom Typ einer Behälterklasse sein, damit sie viele Objekte aufnehmen kann. Ein *Set* (Menge ohne Duplizität) oder ein *Bag* (Menge mit Duplizität) sind beispielsweise übliche Behälterklassen.

[4] . . . beziehungsweise assoziiert sein kann.

Multi-plizität	Erläuterung
	keine Angabe entspricht *
*	null oder größer
0..*	null oder größer
1..*	eins oder größer
1	genau eins
0,1	null oder eins
0..3	null oder eins oder zwei oder drei
7,9	sieben oder neun
0..2,5,7	(zwischen null und zwei) oder fünf oder sieben

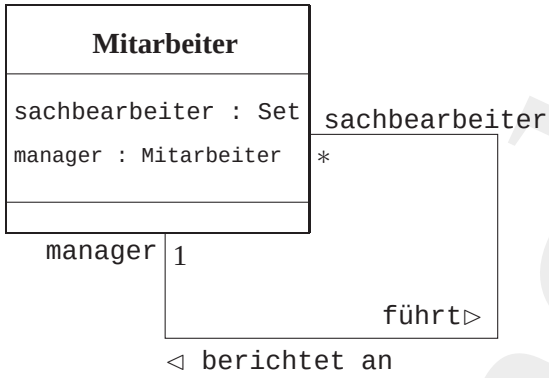
Legende:

`,`	Komma ist Trennzeichen für die Aufzählung
`*`	beliebig viele
`0`	optional

Tabelle 4.4: Angabe der Multiplizität

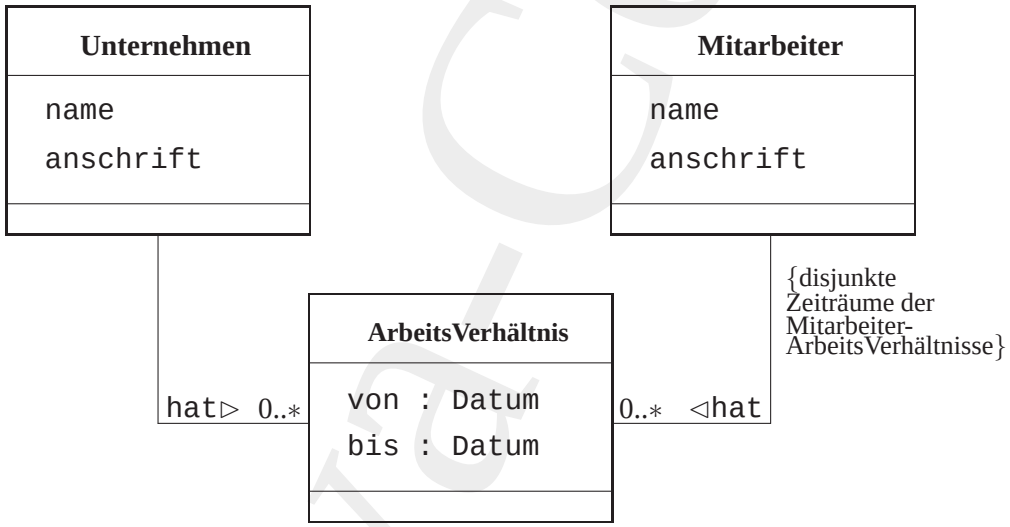
[Hinweis: Moderne Modellierungswerkzeuge verwenden den Rollennamen für die Referenzvariable und generieren entsprechend der Multiplizität die Variable mit ihrem Typ automatisch.]

Eine Assoziation kann selbst Vaiablen haben. Im Beispiel „Ein Unternehmen beschäftigt viele Mitarbeiter“ (↪ Abbildung 4.2 S. 58) kann dies beispielsweise die Historie der Beschäftigungsverhälnisse für einen Mitarbeiter sein, das heißt die `von`- und `bis`-Daten der Assoziation `beschäftigt`. Eine solche Beziehung bezeichnet man als degenerierte Assoziationsklasse. Das Beiwort „degeneriert“ verdeutlicht, daß die Assoziationsklasse keine Instanzen beschreibt und daher keinen eigenen Namen benötigt. In den späteren Phasen der Modellierung wird eine solche degenerierte Klasse in eine vollständige Assoziationsklasse, die dann einen Namen hat und Instanzen aufweisen kann, umgeformt (↪ Abbildung 4.4 S. 60).



Legende:
Assoziation ↪ Abbildung 4.1 S. 57

Abbildung 4.3: Beispiel einer direkten rekursiven Assoziation



Legende:
↪ Abbildung 4.2 S. 58

Abbildung 4.4: Beispiel einer Assoziationsklasse: `ArbeitsVerhältnis`

4.3.3 Referentielle Integrität

Soll eine Assoziation eine Bedingung erfüllen, dann ist diese in Form der {zusicherung} neben der Assoziationslinie zu notieren. Eine {zusicherung} kann auch die referenzielle Integrität beschreiben. Hierzu werden beim Löschen beispielsweise angebenen:

Integrität

- {`prohibit deletion`}
  Das Löschen eines Objektes ist nur erlaubt, wenn keine Beziehung zu einem anderen Objekt besteht.
- {`delete link`}
  Wenn ein Objekt gelöscht wird, dann wird nur die Beziehung zwischen den Objekten gelöscht.
- {`delete related object`}
  Wenn ein Objekt gelöscht wird, dann wird das assoziierte („gegenüberliegende“) Objekt ebenfalls gelöscht.

4.3.4 Schlüsselangabe

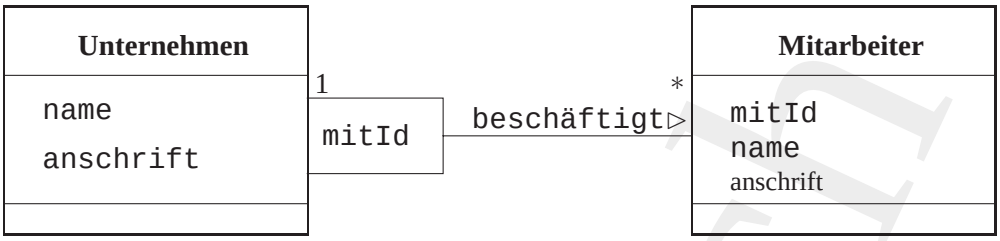
Beziehungen mit der Multiplizität $m = *$ werden in der Regel mittels einer Behälterklasse implementiert (↪ Abschnitt 4.3.2 S. 58). Dabei kann es sinnvoll sein schon im Klassenmodell mit den Assoziationen den Zugriffsschlüssel darzustellen. Ein solcher Schlüssel, auch als qualifizierendes Attribut der Assoziation bezeichnet, wird als Rechteck an der Seite der Klasse notiert, die über diesen Schlüssel auf das Zielobjekt zugreift. Ist ein solcher Schlüssel genannt, dann ist er Bestandteil der Assoziation. Die Navigation erfolgt dann ausschließlich über diesen Schlüssel. Ein Beispiel zeigt Abbildung 4.5 S. 62.

4.4 Beziehungselemente: Ganzes ⇔ Teile

4.4.1 Aggregation

Eine Aggregation beschreibt eine „Ganzes⇔Teile“-Assoziation. (↪ Abbildung 4.6 S. 62) Das Ganze nimmt dabei Aufgaben stellvertretend für seine Teile wahr. Im Unterschied zur normalen Assoziation haben die beteiligten Klassen keine gleichberechtigten Beziehungen. Die Aggre-

Aggregation



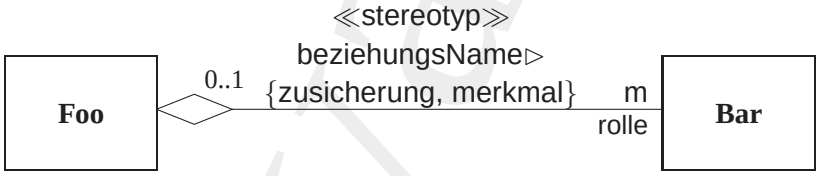
Legende:
↪ Abbildung 4.2 S. 58

Unternehmen		Mitarbeiter		
name	anschrift	mitId	name	anschrift
Otto KG	Nürnberg	Boe	Boesewicht	Bonn
Otto KG	Nürnberg	Gu	Gutknecht	Lüneburg
Emma AG	Berlin	Fr	Freund	Lüneburg

Abbildung 4.5: Beispiel einer qualifizierenden Assoziation (`mitId`)

gationsklasse hat eine hervorgehobene Rolle und übernimmt die„Koordination“ ihrer Teileklassen. Zur Unterscheidung zwischen Aggregationsklasse und Teileklasse(n) wird die Beziehungslinie durch eine Raute auf der Seite der Aggregationsklasse ergänzt. Die Raute symbolisiert das Behälterobjekt, das die Teile aufnimmt.

[Hinweis:Verwandte Begriffe für die Aggregation sind die Begriffe Ganzes-Teile-Beziehung und Assoziation.]



Legende:
↪ Abbildung 4.1 S. 57

Abbildung 4.6: UML-Beziehungselement: Aggregation

Legende:
↪ Abbildung 4.6 S. 62

Abbildung 4.7: Beispiel einer Aggregation: Ein Fahrrad hat zwei Laufräder mit jeweils 36 Speichen

Die Abbildung 4.7 S. 63 zeigt den Fall: „Ein Fahrrad hat zwei Laufräder mit jeweils 36 Speichern“. Dieses Beispiel verdeutlich, daß ein Teil (Laufrad) selbst wieder eine Aggregation sein kann.

4.4.2 Komposition

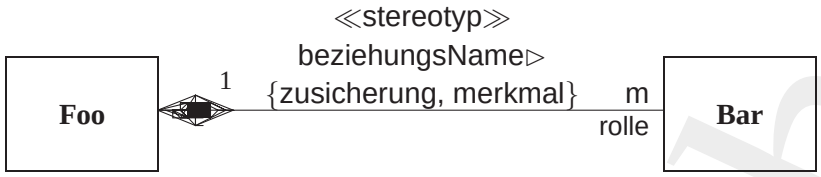
Komposition

Eine Komposition ist eine spezielle Form der Aggregation und damit auch eine spezielle Form der Assoziation. Bei dieser „Ganzes⇔Teile“-Assoziation sind die Teile existenzabhängig vom Ganzen (↪ Abbildung 4.8 S. 64). Die Lebenszeit eines Teils ist abhängig von der Lebenszeit des Ganzen, das heißt, ein Teil wird zusammen mit dem Ganzen (oder im Anschluß daran) erzeugt und wird vor dem Ende des Ganzen (oder gleichzeitig damit) „vernichtet“[5]. Die Kompositionsklasse hat eine hervorgehobene Rolle und übernimmt die „Koordination“ ihrer Teileklassen wie bei der (normalen) Aggregation. Zur Unterscheidung zwischen Kompositionsklasse und Teileklasse(n) wird die Beziehungslinie durch eine ausgefüllte Raute auf der Seite der Kompositionsklasse ergänzt. Auch hier symbolisiert die Raute das Behälterobjekt, das die Teile aufnimmt. Neben der Kennzeichnung durch die ausgefüllte Raute können die Teileklassen auch direkt in den Kasten der Kompositionsklasse geschrieben werden.

[Hinweis:Verwandte Begriffe für die Komposition sind die Begriffe Ganzes-Teile-Beziehung und Assoziation.]

Ein Beispiel aus den Bereich *Graphical User Interface* verdeutlicht, daß ein *Window* aus zwei *Slider*, einem *Header* und einem *Panel* besteht.

[5] In Java^{TM} also nicht mehr referenzierbar.



Alternative Notation:

Legende:
↪ Abbildungen 4.6 S. 62 und 4.1 S. 57

Abbildung 4.8: UML-Beziehungselement: Komposition

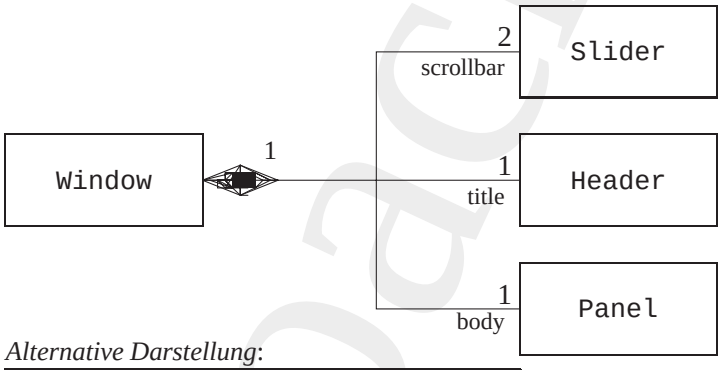
Diese Teile bestehen nicht unabhängig vom Ganzen, dem *Window* (↪ Abbildung 4.9 S. 65).

4.5 Beziehungselement: Vererbung

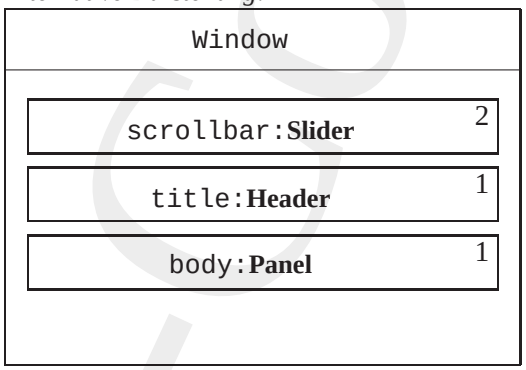
4.5.1 Vererbung

Vererbung

Als Vererbung bezeichnet man einen Mechanismus, der die Eigenschaften (Variablen und Methoden) einer Klasse (≡ Oberklasse) für eine andere Klasse (≡ Unterklasse) zugänglich macht. Aus der Sicht einer Unterklasse sind die Eigenschaften der Oberklasse eine Generalisierung, das heißt, sie sind für die Unterklasse allgemeine (abstrakte) Eigenschaften. Umgekehrt ist die Unterklasse aus der Sicht der Oberklasse eine Spezialisierung ihrer Oberklasseneigenschaften. Mit der Vererbung wird eine Klassenhierarchie modelliert (↪ Abbildung 4.10 S. 66). Welche gemeinsamen Eigenschaften von Unterklassen zu einer Oberklasse zusammengefaßt, also generalisiert werden, und umgekehrt, wel-

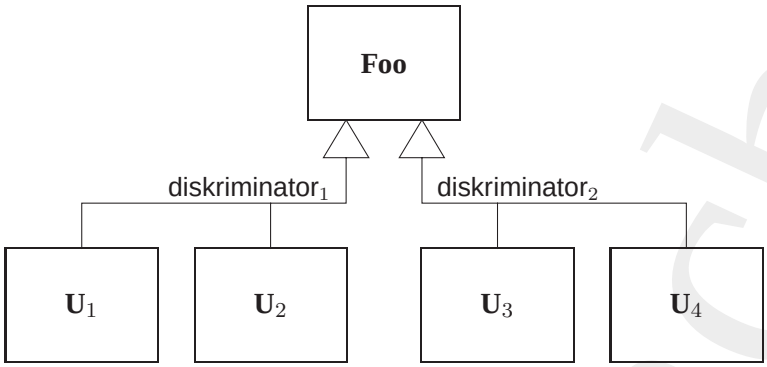


Alternative Darstellung:



Legende:
Ein Window besteht aus zwei Slider, einem Header und einem Panel
↪ Abbildung 4.6 S. 62 (Beispielidee [Rational97])

Abbildung 4.9: Beispiel einer Komposition: Window mit Slider, Header und Panel



Legende:

$\triangle$	$\equiv$	Kennzeichnung der Oberklasse Vererbungsrichtung
dikriminator$_j$	$\equiv$	charakteristisches Gliederungsmerkmal für die Gereralisierungs⇔Spezialisierungs-Beziehung
Foo	$\equiv$	Oberklasse von $\mathbf{U}_1$,$\mathbf{U}_2$,$\mathbf{U}_3$ und $\mathbf{U}_4$ Eigenschaften von **Foo** sind in $\mathbf{U}_i$ zugreifbar
$\mathbf{U}_i$	$\equiv$	$\mathbf{U}_i$ ist Unterklasse von **Foo**

Abbildung 4.10: UML-Beziehungselement: Vererbung

Diskriminator

che Eigenschaften der Oberklasse in Unterklassen genauer beschrieben, also spezialisiert werden, ist abhängig vom jeweiligen charakteristischen Unterscheidungsmerkmal der einzelnen Unterklassen. Ein solches Merkmal wird „Diskriminator“genannt.

Beispielsweise kann eine Oberklasse `Fahrrad` untergliedert werden nach dem Diskriminator `Verwendungszweck` und zwar in die Unterklassen `Rennrad`, `Tourenrad` und `Stadtrad`. Genau so wäre ein Diskiminator `Schaltungsart` möglich. Dieser ergäbe beispielsweise die Unterklassen `KettenschaltungsFahrrad` und `NabenschaltungsFahrrad`. Welcher Diskriminator zu wählen und wie dieser zu bezeichnen ist, hängt von der gewollten Semantik der **Generalisierung⇔Spezialisierung-Relation** ab.

Bei der Modellierung einer Vererbung ist es zweckmäßig den Diskrimitator explizit anzugeben. Dabei ist es möglich, daß eine Oberklasse auf der Basis von mehreren Diskriminatoren Unterklassen hat.
[Hinweis:Verwandte Begriffe für die Vererbung sind die Begriffe Ge-

neralisierung, Sepzialisierung und *Inheritance*.]

4.5.2 Randbedingungen (*Constraints*)

Bei Modellierung einer Vererbung können für die Unterklassen Randbedingungen (*Constraints*) notiert werden. Vordefiniert sind in UML die Randbedingungen:

- {`overlapping`}
  Ein Objekt einer Unterklasse kann gleichzeitig auch ein Objekt einer anderen Unterklasse sein.

  In dem Beispiel „Schiffe“ (↪ Abbildung 4.11 S. 68) könnte ein Objekt `Emma-II` sowohl ein Objekt der Unterklasse `Tanker` wie auch der Unterklasse `ContainerSchiff` sein, wenn {`overlapping`} angegeben wäre.

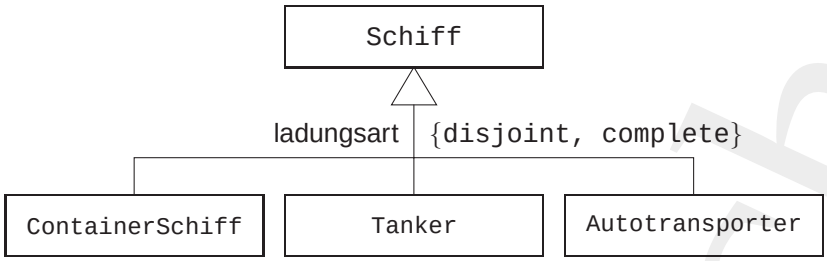
- {`disjoint`}
  Ein Objekt einer Unterklasse kann nicht gleichzeitig ein Objekt einer anderen Unterklasse sein.

  In unserem Schiffsbeispiel könnte das Objekt `Emma-II` nur ein Objekt der Unterklasse `Tanker` sein und nicht auch eines der Unterklassen `ContainerSchiff` und `Autotransporter`, weil {`disjoint`} angegeben ist.

- {`complete`}
  Alle Unterklassen der Oberklasse sind spezifiziert. Es gibt keine weiteren Unterklassen. Dabei ist unerheblich ob in dem Diagramm auch alle Unterklassen dargestellt sind.

  In unserem Schffsbeispiel könnte also keine Unterklasse `KreuzfahrtSchiff` auftauchen, weil {`complete`} angegeben ist.

- {`incomplete`}
  Weitere Unterklassen der Oberklasse sind noch zu spezifizieren. Das Modell ist noch nicht vollständig. Die Aussage bezieht sich auf die Modellierung und nicht auf die Darstellung. Es wird daher nicht {`incomplete`} angebenen, wenn nur aus zeichnerischen Gründen eine Unterklasse fehlt.



Legende:
↪ Abbildung 4.10 S. 66
Eine Oberklasse `Schiff` vererbt an die Unterklassen `ContainerSchiff`, `Tanker` und `Autotransporter`.

Abbildung 4.11: Beispiel einer Vererbung

In unserem Schiffsbeispiel könnte also eine weitere Unterklasse `KreuzfahrtSchiff` später modelliert werden, wenn {`incomplete`} angegeben wäre.

4.6 Pragmatische UML-Namenskonventionen

Setzt sich der Namen aus mehreren ganzen oder abgekürzten Wörtern zusammen, dann werden diese ohne Zwischenzeichen (zum Beispiel ohne „–“ oder „_“) direkt hintereinander geschrieben. Durch Wechsel der Groß-/Kleinschreibung bleiben die Wortgrenzen erkennbar.

Beispiele zum Wechsel der Groß/Kleinschreibung:

```
FahrzeugProg
getGeschwindigkeitFahrrad()
setFahrtrichtung()
```

Die Groß/Kleinschreibung des ersten Buchstabens eines Namens richtet sich (möglichst) nach folgenden Regeln:

package

- *Paket*: Der Name beginnt mit einem kleinen Buchstaben.
  Beispiel: `java.lang`

class

- *Klasse* oder *Schnittstelle*: Der Name beginnt mit einem Großbuchstaben.
  Beispiel: `Fahrzeug`
- *(Daten-)Typ* (zum Beispiel Rückgabetyp): Der Name beginnt in der Regel mit einem Großbuchstaben, weil eine (Daten-)Typangabe eine Klasse benennt. Bei einfachen Java-Datentypen (*Primitive-Type* ↪ Tabelle 6.5 S. 150) beginnen die Namen mit einem kleinen Buchstaben um der Java-Konvention zu entsprechen.
  Beispiele: `int`, `double`, `MyNet`, `HelloWorld`
- *Variable* oder *Methode*: Der Name beginnt mit einem kleinen Buchstaben. Ausnahme bildet eine Konstruktor-Methode. Der Konstruktor muß in Java^{TM} exakt den Namen der Klasse haben. Deshalb beginnt ein Konstruktor stets mit einem Großbuchstaben.
  Beispiel: `Fahrzeug()`, `beschleunigt()`

Variable

Methode

- *Merkmal* oder *Zusicherung*: Der Name beginnt mit einem kleinen Buchstaben.
  Beispiel: {`public`}
- *Stereotyp*: Der Name beginnt mit einem kleinen Buchstaben.
  Beispiel: {`metaclass`}

4.7 OMG & UML

> The OMG claim that
> *"UML is a language for*
> *specifying, visualizing, constructing and documenting*
> *the artifacts of software systems,*
> *as well as for business modeling*
> *and other non-software systems."*

Die *Object Management Group* (OMG) begann im Jahre 1989 als Entwickler des Standards *Common Object Request Broker Architecture* (CORBA). Im Jahre 1997 standartisierte sie UML1 als eine umfassende Modellierungssprache, die bewährte Modellierungstechniken integrierte und dies auf der Grundlage einer einheitlichen graphische Notation. Derzeit arbeitet die OMG an einer wesentlichen Überarbeitung und Fortschreibung des Standards zu einem UML2.

Dabei geht es um Lösungsvorschläge für eine Menge neuer Anforderungen (52 *Requirements*, ↪ [Miller02]). In diesem Kontext haben bis Oktober 2002 fünf Gruppen *Proposals*[6] eingereicht:

U2P: Bran Selic / Guus Ramackers / Cris Kobryn; *UML 2.0 Partners* (U2P)

DSTC: Keith Duddy; *Distributed Systems Technology Center* (DSTC)

2U: Stephen J. Mellor; *Umambiguous UML* (2U)

3C: William Frank / Kevin P. Tyson; *Clear, Clean, Concise* (3C)

OPM: Dov Dori; *Object Process Methodology* (OPM)

MDA

OMG's *Model Driven Architecture* (MDA) Initiative ist ein Ansatz für Industriestandards, die geprägt sind von der Überzeugung, dass anstatt der Programme die *primären Artifacte* der Softwareentwicklung zu kreieren sind. Mit Werkzeugen sind die Programme dann mehr oder weniger automatisch generierbar.

[6] UML2 Proposals ↪ `http://www.community-ML.org/`

Kapitel 5

Java^{TM} $\approx$ mobiles Code-System

Write Once, Run Everywhere.
Slogan der Sun Microsystems, Inc. USA

Klassen mit Variablen und Methoden, Assoziationen, Aggregationen, Kompositionen und Vererbung sind (nun bekannte) Begriffe der Objekt-Orientierung. Java^{TM} ist jedoch mehr als eine objektorientierte Programmiersprache. Java^{TM} ist (fast) ein *mobiles Code-System*.

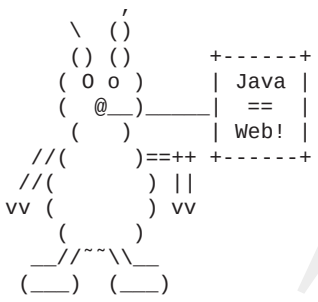
Java^{TM} ermöglicht es, Code einschließlich Daten über Netzknoten, also über Computer in den Rollen eines Clients und eines Servers, problemlos zu verteilen und auszuführen. Ein Stück mobiler Java-Code (*Applet*) wird dynamisch geladen und von einem eigenständigen („standalone“) Programm ausgeführt. Ein solches Programm kann ein Web-Browser, Appletviewer oder Web-Server sein.

Trainingsplan

Das Kapitel „Java^{TM} ≈ mobiles Code-System“ erläutert:

- das Zusammenspiel von Java^{TM} und dem Web,
  ↪ Seite 72 . . .
- die Portabilität aufgrund des Bytecodes,
  ↪ Seite 75 . . .
- das Sicherheitskonzept und
  ↪ Seite 77 . . .
- skizziert den Weg zur Softwareentwicklung mit Java.
  ↪ Seite 79 . . .

5.1 Java^{TM} im Netz



Java^{TM} ist auch ein System, um im Internet ausführbaren Code auszutauschen. Eine Anwendung im *World Wide Web* (ursprünglich WWW; heute kurz Web) kann zusätzlich zum gewohnten Laden von Texten, Graphiken, Sounds und Videos den Java^{TM} Bytecode laden und diesen direkt ausführen. Über einen vorgegebenen Fensterausschnitt des Browsers kann dann dieses Bytecodeprogramm mit dem Benutzer kommunizieren.

Java^{TM} ist daher auch ein *mobiles Code-System*.[1] Ein solches System

[1] Ein anderes Beispiel für ein mobiles Code-System ist *Safe-Tcl* ↪ [Orfali/Harkey97].

ermöglicht es, Code einschließlich Daten über Netzknoten, also über Computer in den Rollen eines Clients und eines Servers, zu verteilen. Ein mobiles Objekt, in der Java-Welt als *Applet* bezeichnet, ist selbst ein Stück ausführbarer Code. Ebenso wie traditionelle Software enthält auch ein mobiles Objekt eine Sequenz von ausführbaren Instruktionen. Anders jedoch als bei traditioneller Software wird ein mobiles Objekt, also ein Applet, dynamisch geladen und von einem eigenständigen („standalone") Programm ausgeführt. Ein solches Programm kann ein Web-Browser, Appletviewer oder Web-Server sein.

Das Web-Szenario der Client⇔Server-Interaktionen läßt sich mit folgenden Schritten skizzieren:

1. **Anfordern des Applets** (*request*-Schritt)
   Request
   Ein Java-fähiger Browser (Client) fordert das Applet vom Server an, wenn er im empfangenen HTML-Dokument ein `<applet>`-Konstrukt feststellt. Das Attribut `CODE` der `<applet>`-Marke hat als Wert den Namen des Applets, also den Dateinamen der Java-Bytecodedatei mit dem Suffix `class`. Typischerweise befindet sich diese Java-Bytecodedatei auf demselben Server wie das angefragte HTML-Dokument.

2. **Empfangen des Applets** (*download*-Schritt)
   Download
   Der Browser initiiert eine eigene TCP/IP-Session um das Applet vom Server herunterzuladen (*download*). Der Browser behandelt dabei das Applet wie andere HTML-Objekte, zum Beispiel wie eine Video- oder Sounddatei.

3. **Laden und ausführen des Applets** (*execute*-Schritt)
   Execute
   Der Browser läd das Applet in den Arbeitsspeicher des Client und stößt seine Ausführung an. Typischerweise kreiert ein Applet graphische Ausgaben und reagiert auf Eingaben (Keyboard und Maus). Dies geschieht alles in einer festgelegten Bildschirmfläche der angezeigten HTML-Seite. Die Größe dieser Fläche wird durch die Werte der Attribute `width` und `height` bestimmt.

4. **Stoppen und löschen des Applets** (*delete*-Schritt)
   Delete
   Der Browser stoppt die Ausführung des Applet und gibt den Arbeitsspeicher des Client wieder frei. Dies geschieht beim „Verlasssen" des HTML-Dokumentes.

Jedes mobile Code-System, also auch Java^{TM}, sollte die beiden Kernforderungen *Portabilität* und *Sicherheit* möglichst gut erfüllen. Dafür muß es (mindestens) folgende Aspekte abdecken:

Portabilität

1. Eine Plattformunabhängigkeit der gesamten Leistungen
   Ein mobiles Code-System stellt ein plattform-übergreifendes Management des Arbeitsspeichers bereit. Parallel ablaufende Prozesse (*threads*) und ihre Kommunikation inclusive ihrer Synchronisation sind unabhängig vom jeweiligen Betriebssystem der Plattform realisiert. Die gleiche Plattformunabhängigkeit wird auch für die graphische Benutzungsschnittstelle (GUI) gewährleistet.

2. Ein Kontrollsystem für den ganzen Lebenszyklus
   Ein mobiles Code-System stellt die Laufzeitumgebung für das Laden, Ausführen und das „Entladen“ des Codes bereit.

Sicherheit

1. Eine kontrollierbare Ausführungsumgebung für den mobilen Code (*safe environment*)
   Bei einem mobilen Code-System ist der Anwender in der Lage, die Ausführungsumgebung des Codes präzise zu steuern, das heißt, den Zugriff auf den Arbeitsspeicher und auf das Dateisystem, den Aufruf von Systemroutinen und das Nachladen von Servern zu kontrollieren.

2. Eine sichere Code-Verteilung über das Netz
   Ein mobiles Code-System gestaltet den Transfer des Codes über das Netz sicher, also unverfälscht. Dazu ist die Authentifikation sowohl auf Client- wie auf Server-Seite erforderlich. Es gewährleistet, daß der Client beziehungsweise der Server wirklich derjenige ist, der er vorgibt zu sein. Zusätzlich ist der Code zu zertifizieren. Pointiert formuliert: Es tut alles, damit der Code nicht von „Viren“ infiziert werden kann.

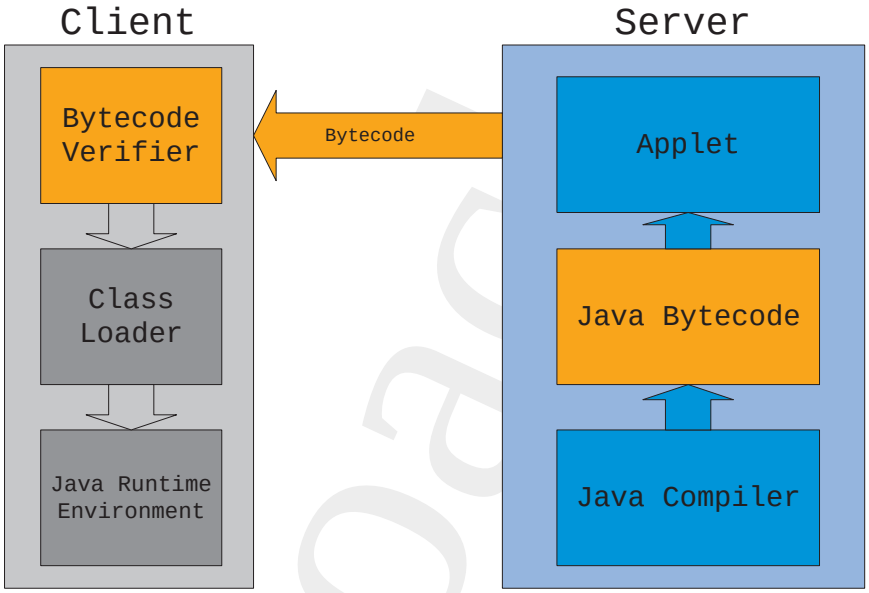


Abbildung 5.1: Von der Bytecode-Produktion bis zur Ausführung

5.2 Bytecode: Portabilität ⇔ Effizienz

Java^{TM} realisiert die Portabilität indem der Java-Quellcode übersetzt wird in primitive Instruktionen eines virtuellen Prozessors. Diese maschinennäheren, primitiven Instruktionen nennt man Bytecode. Das Compilieren bezieht sich bei Java^{TM} nicht auf den Befehlssatz eines bestimmten, marktüblichen Prozessors, sondern auf die sogenannte *Java Virtual Machine*. Der Bytecode bildet eine möglichst maschinennahe Code-Ebene ab, ohne jedoch, daß seine einzelnen Instruktionen wirklich maschinenabhängig sind. Darüberhinaus legt Java^{TM} die Größe seiner einfachen Datentypen (*PrimitiveType*↪ Tabelle 6.5 S. 150) und das Verhalten seiner arithmetischen Operatoren präzise fest. Daher sind Rechenergebnisse stets gleich, also unabhängig davon, ob die jeweilige Plattform 16-, 32- oder 64-Bit-basiert ist. **Bytecode**

Der Bytecode macht Java^{TM} zu einer sogenannten „partiell-compilierten“ Sprache (↪ Abbildung 5.1 S. 75).

Um den Bytecode aus dem Java-Quellcode zu erzeugen, ist ungefähr

80% des gesamten Compilationsaufwandes notwendig; die restlichen 20% entfallen auf Arbeiten, die das Laufzeitsystem übernimmt. So kann man sich Java^{TM} als 80% compiliert und 20% interpretiert vorstellen. Dieser 80/20-Mix führt zu einer exzellenten Code-Portabilität bei gleichzeitig relativ guter Effizienz, da der Java-Bytecode eine gelungene, recht maschinennahe Abstraktion über viele Plattformen darstellt. Trotz alledem ist der Bytecode beim Interpretieren über 15mal[2] langsamer als maschinenspezifisch compilierter Code (≡ *native code*). Um diesen Nachteil an Effizienz auszuräumen gibt es auch für Java^{TM} heute *Just-In-Time*-Compiler[3] (JIT) und reguläre, maschinenabhängige Compiler.

80/20-Mix

JIT

Bei genauer Betrachtung läuft jedes Java-Programm nur einen sehr geringen Prozentsatz seiner Zeit wirklich in Java. Java^{TM} schafft nur den Eindruck über jede Plattform-Architektur Alles exakt zu kennen. Wie soll Java^{TM} bespielsweise wissen wie eine Linie auf dem Bildschirm für jede möglich Plattform gezogen wird. Jedes Betriebssystem in dem Java^{TM} heute üblicherweise läuft nutzt dafür Routinen, geschrieben in anderen Sprachen, zum Beispiel in C oder C++. Egal ob nun Etwas auf dem Bildschirm auszugeben ist oder ein *Thread*[4] oder eine TCP/IP-Verbindung zu meistern sind, das was Java^{TM} tun kann, ist das jeweilige Betriebssystem zu beauftragen diese Dinge zu tun. So wird letztlich eine Java^{TM}-Anwendung auch über die Abarbeitung von Routinen in anderen Programmiersprachen, beispielsweise in C-Code, realisiert.

[2] ↪ [Orfali/Harkey97] page 32.

[3] Ein JIT-Compiler konvertiert Java's Stack-basierte Zwischenrepräsentation in den benötigten (*native*) Maschinencode und zwar unmittelbar vor der Ausführung. Die Bezeichnung „Just-In-Time" vermittelt den Eindruck einer rechtzeitigen (schnellen) Programmausführung. Aber der JIT-Compiler erledigt seine Arbeit erst nachdem man der Anwendung gesagt hat: „run". Die Zeit zwischen diesem Startkommando und dem Zeitpunkt, wenn das übersetzte Programm wirklich beginnt das Gewünschte zu tun, ist Wartezeit für den Anwedner. Ein mehr passender Name wäre daher „*Wait to the Last Minute Holding Everybody Up Compiler*" [Tyma98] p. 42.

[4] Näheres dazu ↪ Abschnitt 7.1 S. 161.

5.3 Sicherheit

5.3.1 Prüfung des Bytecodes (*Bytecode Verifier*)

Für die Sicherheit ist in den Bytecode-Zyklus von der Erzeugung über das Laden bis hin zur Ausführung ein Schritt der Code-Verifizierung eingebaut. Zunächst wird der Java-Quellcode zu Bytecode compiliert. Danach wird dieser Bytecode üblicherweise über das Netz zum nachfragenden Client transferiert. Bevor der Bytecode dort ausgeführt wird, durchläuft er den Bytecode-Verifizierer. Dieser prüft den Bytecode in vielerlei Hinsicht, beispielsweise auf nachgemachte Zeiger, Zugriffsverletzungen, nicht passende Parametertypen und auf Stack-Überlauf.

Verifier

Man kann sich den Verifizierer als einen Türkontrolleur vorstellen, der aufpaßt, daß kein unsicherer Code von außerhalb oder auch von der lokalen Maschine Eintritt zur Ausführung hat. Erst nach seinem OK wird das Laden der Klassen aktiviert. Dieses übernimmt der Klassenlader (*class loader*). Er übergibt den Bytecode an den Interpreter. Dieser ist das Laufzeitelement, das die Bytecode-Instruktionen auf der Arbeitsmaschine in die dortigen Maschinenbefehle umsetzt und zur Ausführung bringt (↪ Abbildung 5.1 S. 75).

5.3.2 Traue Niemandem!

Die Sicherheitsphilosophie ist geprägt von der Annahme, daß Niemandem zu trauen ist. Dieses Mißtrauen hat zu einem Konzept der Rundumverteidigung geführt. Diese beschränkt sich nicht nur auf den Bytecode-Verifizierer, sondern setzt bei der Sprache selbst an und bezieht selbst den Browser mit ein. Im folgenden sind einige Aspekte dieser Rundumverteidigung skizziert.

Sicherheit durch das *Memory Layout* zur Laufzeit Ein wichtiger Sicherheitsaspekt liegt in der Entscheidung über die Bindung von Arbeitsspeicher (*Memory*). Im Gegensatz zu den Sprachen `C` und `C++` wird vom Java^{TM}-Compiler nicht das Memory-Layout entschieden. Es wird erst abgeleitet zur Laufzeit. Dieser Mechanismus einer späten Bindung verhindert es, aus der Deklaration einer Klasse auf ihr physikalisches Memory-Layout zu schließen. Eine solche Kenntnis war stets ein Tor für „Einbrüche“.

keine Zeiger

Sicherheit durch Verzicht auf Zeiger Java^{TM} verzichtet auf Zeiger (*Pointer*) in der Art wie sie in den Sprachen `C` und `C++` vorkommen und dort auch häufig im Sinne schwerdurchschaubarer Codezeilen genutzt werden. Java^{TM} kennt keine Speicherzellen, die ihrerseits wieder Adressen zu anderen Zellen speichern. Java^{TM} referenziert Arbeitsspeicher nur über symbolische „Namen“, deren Auflösung in konkrete Speicheradressen erst zur Laufzeit durch den Java-Interpreter erfolgt. Es gibt daher keine Gelegenheit, Zeiger zu „verbiegen“, um hinterrücks etwas zu erledigen.

Namen

Sicherheit durch eigene Namensräume Der Klassenlader unterteilt die Menge der Klassen in unterschiedliche Namensräume. Eine Klasse kann nur auf Objekte innerhalb ihres Namensraumes zugreifen. Java^{TM} kreiert einen Namensraum für alle Klassen, die vom lokalen Dateisystem kommen, und jeweils einen unterschiedlichen Namensraum für jede einzelne Netzquelle. Wird eine Klasse über das Netz importiert, dann wird sie in einen eigenen Namensraum plaziert, der mit ihrer Quelle (Web-Server) assoziiert ist. Wenn eine Klasse `Foo` die Klasse `Bar` referenziert, dann durchsucht Java^{TM} zuerst den Namensraum des lokalen Dateisystems (*built-in classes*) und danach den Namensraum der Klasse `Foo`.

Sicherheit durch Zugriffs-Kontroll-Listen Die Dateizugriffskonstrukte implementieren die sogenannten Kontroll-Listen. Damit lassen sich Lese- und Schreib-Zugriffe zu Dateien, die vom importierten Code ausgehen oder von ihm veranlaßt wurden, benutzungsspezifisch kontrollieren. Die Standardwerte (*defaults*) für diese Zugriffs-Kontroll-Listen sind äußerst restriktiv.

Sicherheit durch Browser-Restriktionen Moderne Browser unterscheiden verschiedene Sicherheitsstufen. So lassen sich Netzzugriffe eines Applets unterbinden oder auf den Bereich einer Sicherheitszone (*Firewall-Bereich*) begrenzen.

PGP

Sicherheit durch zusätzliche Applet-Zertifizierung Mit Hilfe der Kryptologie läßt sich die Sicherheit wesentlich steigern. So kann beispielsweise über das Verfahren *Pretty Good Privacy* (PGP) ein Applet

unterschrieben (signiert) werden. Veränderungen am Bytecode auf dem Transport werden sofort erkannt. Der liefernde Server und der nachfragende Client können einer Authentifikation unterzogen werden, das heißt, sie müssen sich ordungsgemäß ausweisen.

5.4 The Road To Java

Sun Microsystems formliert unter dem Motto „The Road to Java“[5] fünf Meilensteine, die den Weg hin zu einer Java^{TM} Computing Architektur markieren: **Umstellung**

1. **Erhebung** (*Investigate*):
   Sammlung von Informationen über Java^{TM} und über die Geschäftsauswirkungen von Java^{TM} Computing.

2. **Bewertung** (*Evaluate*):
   Bewertung von Technologien und Geschäftsauswirkungen im Rahmen des jeweiligen Unternehmens.

3. **Gestaltung** (*Architect*):
   Entwicklung einer um Java^{TM} Computing erweiterten Architektur der bisherigen Informationstechnologie.

4. **Pilotierung** (*Pilot*):
   Initiierung von Pilotprojekten, um Erfahrungen zu sammeln.

5. **Betrieb** (*Implement*):
   Implementierung und unternehmensweite Umsetzung der Java^{TM} Computing Architektur.

Integrated Development Environments (IDEs) für Java^{TM} (z. B. Forte^{TM} for Java^{TM}, Community Edition von Sun Microsystems, Inc. oder Visual Age for Java^{TM} von IBM) unterstellen zunächst einzelnen Systementwickler statt ein Team von Konstrukteuren. Sie enthalten beispielsweise Versionsmanagement oder Test- und Freigabeunterstützung nur in Ansätzen. Jedoch werden zunehmende von modernen IDEs die Anforderungen für sehr große Projekte („Millionen Zeilen Quellcode“) mit mehreren Entwicklungsteams abgedeckt.

[5] ↪ [Sun97] S. 4

Entwicklungsumgebung für die dargestellten Beispiele Die Beispiele wurden auf den beiden folgenden Plattformen entwickelt:

AIX

- *AIX-Plattform*:
  IBM[6] RISC[7]-Workstation RS[8]/6000, Typ 250 und Typ 43p, Betriebssystem AIX[9] 4.1

NT & XP

- *NT-Plattform*:
  Intel PC, Betriebssystem Microsoft Windows NT[10] und Microsoft Windows XP [Version 5.1.2600] (C) Copyright 1985-2001 Microsoft Corp.

Aufgrund der langen Fortschreibungsdauer wurden für JAVA^{TM}–COACH eine Vielzahl von Java-Produkten genutzt. An dieser Stelle sind beispielsweise zu nennen:

1. die *Integrierten Entwicklungsumgebungen* wie Sun ONE Studio 4 (update 1) oder IBM Eclipse (Version 2) mit ihren Basisprodukten:
   (a) `appletviewer` — Java-Appletviewer
   (b) `java` — Java-Interpreter
   (c) `javac` — Java-Compiler
   (d) `javadoc` — Java-Dokumentations-Generator
   (e) `javah` — native Methoden, C-Dateigenerator
   (f) `javap` — Java-Klassen-Disassembler
   (g) `jdb` — Java-Debugger

Emacs

2. **Editor** und **Shell**:
   Emacs[11] in verschiedene Versionen zum Beispiel `XEmacs`, GNU[12] Emacs und `jEdit` Version 4.1 pre 5 (↪`http://www.jedit.org`).

[6] IBM ≡ International Business Machines Corporation
[7] RISC ≡ Reduced Instruction Set Computer
[8] RS ≡ RISC System
[9] AIX ≡ Advanced Interactive Executive — IBM's Implementation eines *UNIX-Operating System*
[10] NT ≡ New Technology, 32-Bit-Betriebssystem mit Multithreading und Multitasking
[11] Emacs ≡ Editing Macros — gigantic, functionally rich editor
[12] GNU ≡ Free Software Foundation: GNU stands for „GNU's Not Unix"

3. **Browser**:
   Netscape Communicator Version $\geq$ *4.03* & *Microsoft Internet Explorer Version* $\geq$ *4.0*

4. **Datenaustausch** im Netz:
   File Transfer Program (FTP) auf Basis von TCP/IP[13]

5. **Prozeß- und Produkt-Dokumentation**:
   XHTML-Dateien auf einem Web-Server, erstellt mit ↪ Punkt 2 S. 80.

[13] TCP/IP $\equiv$ Transmission Control Protocol / Internet Protocol — communications protocol in UNIX environment

Kapitel 6

Konstrukte (Bausteine zum Programmieren)

Es werden Applikationen (≡ „eigenständige“ Programme) von Applets (≡ Programm eingebettet in eine HTML-Seite) unterschieden. Beide benutzen die Bausteine (*Konstrukte*[1]) aus der Java 2 Plattform. Praxisrelevante Konstrukte werden anhand von Beispielen eingehend erläutert.

[1]Der lateinische Begriff Konstrukt (Construct, Constructum) bezeichnet eine Arbeitshypothese für die Beschreibung von Phänomenen, die der direkten Beobachtung nicht zugänglich sind, sondern nur aus anderen beobachteten Daten erschlossen werden können. In der Linguistik ist zum Beispiel Kompetenz ein solches Konstrukt. Im JAVA^{TM}–COACH wird jeder verwendbare „Baustein“, der bestimmte Eigenschaften hat, als Konstrukt bezeichnet.

Trainingsplan

Das Kapitel „Java^{TM}-Konstrukte“ erläutert:

- Klassen, Variablen, Methoden, Parameter, Konstruktoren, Internetzugriff, Threads und GUI-Bausteine anhand von ersten Kostproben,
  ↪ Seite 84 . . .
- die Aufgaben von Applet und Applikation,
  ↪ Seite 132 . . .
- das Einbinden eines Applets in ein XHTML-Dokument,
  ↪ Seite 132 . . .
- die Syntax, die Semantik und die Pragmatik.
  ↪ Seite 148 . . .

6.1 Einige Java-Kostproben

Die folgenden Kostproben enthalten zum Teil Konstrukte, die erst später eingehender erläutert werden. Der Quellcode der Beispiele dient primär zum schnellen Lernen der Syntax, Semantik und Pragmatik von Java^{TM}. Er ist nicht im Hinblick auf Effizienz optimiert oder entsprechend eines einheitlichen (Firmen-)Standards formuliert.
[Hinweis: Die Zeilennummerierung ist kein Quellcodebestandteil.]

6.1.1 Kostprobe `HelloWorld`

Jeder Einstieg in eine formale (Programmier-)Sprache beginnt mit der Ausgabe der Meldung „Hello World“ auf dem Bildschirm. Dies entspricht einer „alten“ Informatik-Gepflogenheit. Die Abbildung 6.1 S. 85 zeigt das Klassendiagramm für die Applikation `HelloWorld`.

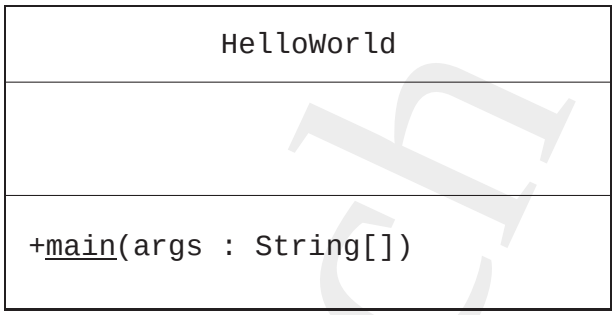


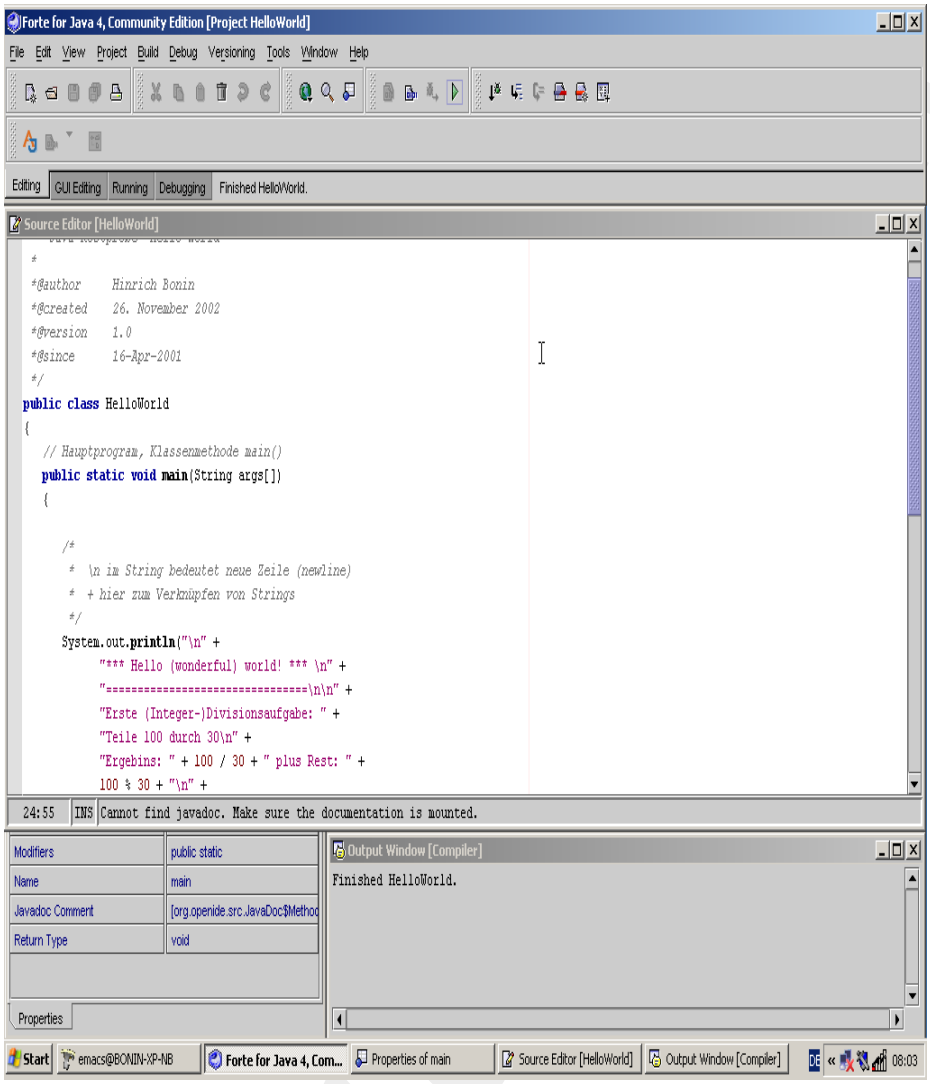
Abbildung 6.1: Klassendiagramm für `HelloWorld`

Die Startmethode einer Applikation ist `main()`. Sie hat folgende Eigenschaften:

- Sie ist von „Außen“ zugreifbar ↪ Kennwort `public`. `public`
- Da ihr Klassenname beim Aufruf der Java^{TM} 2 Runtime Environment zugeben ist (— und nicht eine Instanz ihrer Klasse —) ist `main()` selbst eine Klassenmethode ↪ Kennwort `static`. `static`
- Da keine Rückgabe eines Wertes organisiert ist (— Wohin damit? —), ist `main()` ohne Rückgabewert definiert werden ↪ Kennwort `void`). `void`
- Die Methode `main()` hat nur einen Parameter. Dieser wird üblicherweise mit `argv` (*argument value*) oder `args` (*arguments*) angegeben. Er ist als ein Feld von Zeichenketten (`String`) deklariert. `args` `argv`

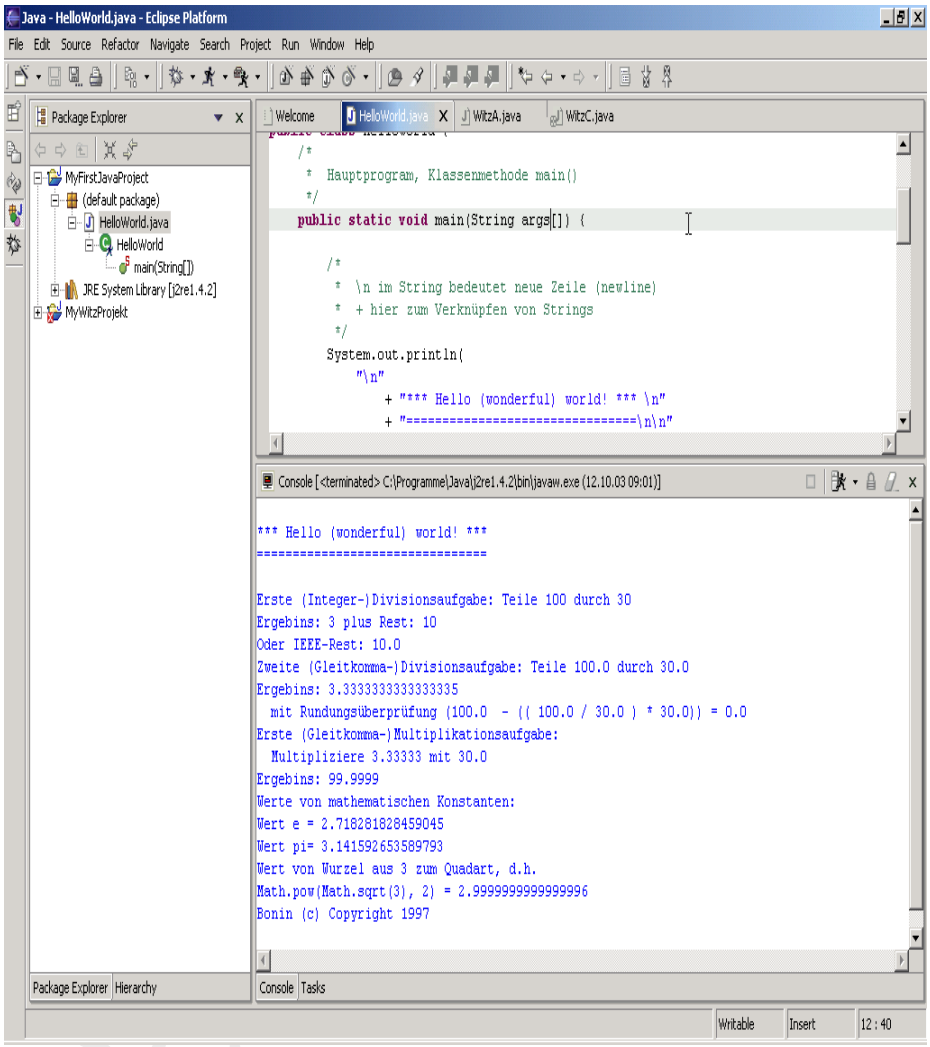
Die Quellcodedatei `HelloWorld.java` abgebildet in der IDE *Forte*^{TM} *for Java*^{TM} *4* von Sun Microsystems, Inc. USA, ↪ Abblidung 6.2 S. 86, abgebildet in der IDE *Eclipse* von IBM ↪ Abblidung 6.3 S. 87. Näheres zur IDE *Eclipse* ↪ Abschnitt 8.1 S. 321.

Klasse `HelloWorld`



Legende:
IDE *Forte*^{TM} *for Java*^{TM}*, Community Edition*, Sun Microsystems, Inc. USA
↪ `http://forte.sun.com/`

Abbildung 6.2: `HelloWorld.java` in *Forte*^{TM} *for Java*^{TM} *4*



Legende:
IDE *Eclipse* Version: 2.1.1, (c) Copyright IBM Corp. 2003. ↪ `http://www.eclipse.org/platform` (online 12-Oct-2003). Das Produkt enthält Software der *Apache Software Foundation* ↪ `http://www.apache.org/` (online 12-Oct-2003). Näheres zur IDE *Eclipse* ↪ Abschnitt 8.1 S. 321.

Abbildung 6.3: `HelloWorld.java` in *Eclipse Platform*

```
/**
 *  Java-Kostprobe "Hello World"
 *
 *@author     Hinrich Bonin
 *@version    1.1 16-Apr-2001
 */
public class HelloWorld
{
   /*
    *  Hauptprogram, Klassenmethode main()
    */
   public static void main(String[] args)
   {

      /*
       *  \n im String bedeutet neue Zeile (newline)
       *  + hier zum Verknüpfen von Strings
       */
      System.out.println(
            "\n"
             + "*** Hello (wonderful) world! *** \n"
             + "================================\n\n"
             + "Erste (Integer-)Divisionsaufgabe: "
             + "Teile 100 durch 30\n"
             + "Ergebins: "
             + 100 / 30
             + " plus Rest: "
             + 100 % 30
             + "\n"
             + "Oder IEEE-Rest: "
             + Math.IEEEremainder(100, 30)
             + "\n"
             + "Zweite (Gleitkomma-)Divisionsaufgabe: "
             + "Teile 100.0 durch 30.0 \n"
             + "Ergebins: "
             + 100.0 / 30.0
             + "\n"
             + "  mit Rundungsüberprüfung "
             + "(100.0  - (( 100.0 / 30.0 ) * 30.0)) = "
             + (100.0 - ((100.0 / 30.0) * 30.0))
             + "\n"
             + "Erste (Gleitkomma-)Multiplikationsaufgabe: \n"
```

```
                + "  Multipliziere 3.33333 mit 30.0\n"
                + "Ergebins: "
                + (3.33333 * 30.0)
                + "\n"
                + "Werte von mathematischen Konstanten: \n"
                + "Wert e = "
                + Math.E
                + "\n"
                + "Wert pi= "
                + Math.PI
                + "\n"
                + "Wert von Wurzel aus 3 zum Quadart, d.h. \n"
                + "Math.pow(Math.sqrt(3), 2) = "
                + Math.pow(Math.sqrt(3), 2)
                + "\n"
                + "Bonin (c) Copyright 1997 \n");
    }
}
```

Compilation und Ausführung von `HelloWorld`:

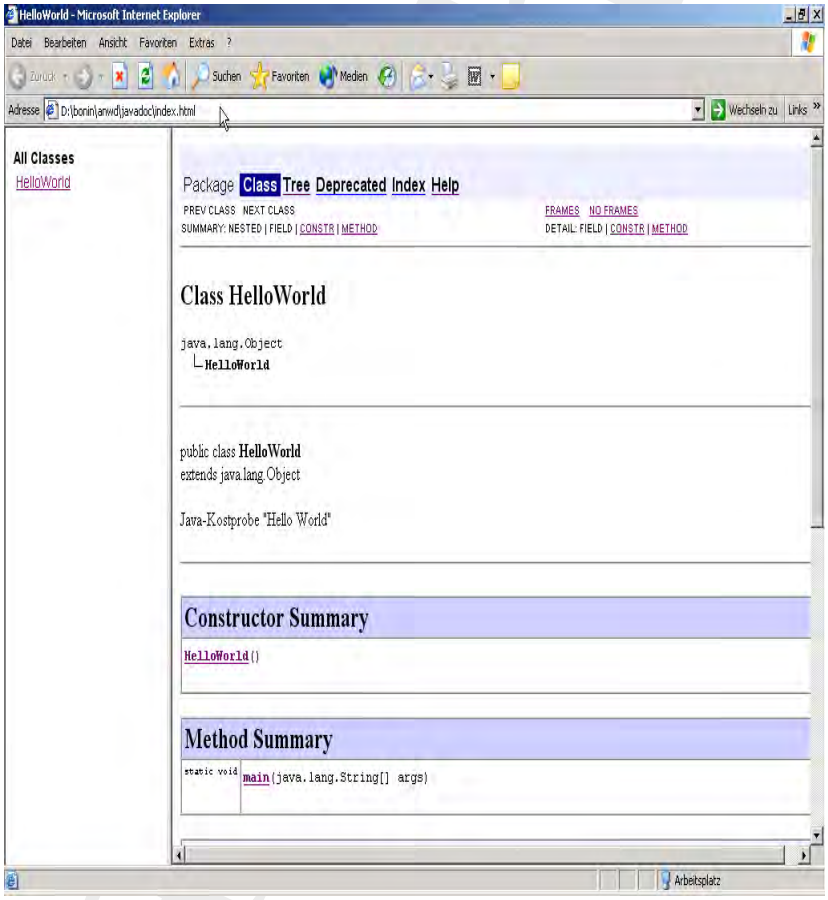
```
d:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

d:\bonin\anwd\code>javac HelloWorld.java

d:\bonin\anwd\code>java HelloWorld

*** Hello (wonderful) world! ***
================================

Erste (Integer-)Divisionsaufgabe: Teile 100 durch 30
Ergebins: 3 plus Rest: 10
Oder IEEE-Rest: 10.0
Zweite (Gleitkomma-)Divisionsaufgabe: Teile 100.0 durch 30.0
Ergebins: 3.3333333333333335
  mit Rundungsüberprüfung (100.0  - (( 100.0 / 30.0 ) * 30.0)) = 0.0
Erste (Gleitkomma-)Multiplikationsaufgabe:
  Multipliziere 3.33333 mit 30.0
```

```
Ergebins: 99.9999
Werte von mathematischen Konstanten:
Wert e = 2.718281828459045
Wert pi= 3.141592653589793
Wert von Wurzel aus 3 zum Quadart, d.h.
Math.pow(Math.sqrt(3), 2) = 2.9999999999999996
Bonin (c) Copyright 1997


d:\bonin\anwd\code>javadoc -d D:\bonin\anwd\javadoc HelloWorld.java
Loading source file HelloWorld.java...
Constructing Javadoc information...
Standard Doclet version 1.4.2
Generating D:\bonin\anwd\javadoc\constant-values.html...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating D:\bonin\anwd\javadoc\overview-tree.html...
Generating D:\bonin\anwd\javadoc\index-all.html...
Generating D:\bonin\anwd\javadoc\deprecated-list.html...
Building index for all classes...
Generating D:\bonin\anwd\javadoc\allclasses-frame.html...
Generating D:\bonin\anwd\javadoc\allclasses-noframe.html...
Generating D:\bonin\anwd\javadoc\index.html...
Generating D:\bonin\anwd\javadoc\packages.html...
Generating D:\bonin\anwd\javadoc\HelloWorld.html...
Generating D:\bonin\anwd\javadoc\package-list...
Generating D:\bonin\anwd\javadoc\help-doc.html...
Generating D:\bonin\anwd\javadoc\stylesheet.css...

d:\bonin\anwd\code>
```

Die Abbildung 6.4 S. 91 zeigt einen Ausschnitt der Dokumentation von `HelloWorld`, die mit `javadoc` generiert wurde.

6.1.2 Kostprobe `Foo` — Parameterübergabe der Applikation

Das Beispiel `Foo` zeigt wie Argumente beim Aufruf der Java-Applikation übergeben werden. Die Werte der Argumente werden als Zeichenkette (Datentyp `String`) an den einen Parameter der Methode `main()` gebunden. Dieser Parameter ist vom Datentyp `Array` und umfaßt soviele Felder wie es Argumente gibt. Die Adressierung dieser Felder beginnt mit dem Wert 0; das heißt, der Wert des erste Arguments „steht“ im nullten Feld (*zero based*). Die Abbildung 6.5 S. 92 zeigt das



Legende:
Diese Abbildung im Browser *Microsoft Internet Explorer Version: 6.0* zeigt einen Ausschnitt aus der Dokumentation, die mit `javadoc` generiert wurde.

Abbildung 6.4: Beispiel `HelloWorld` — `javadoc`

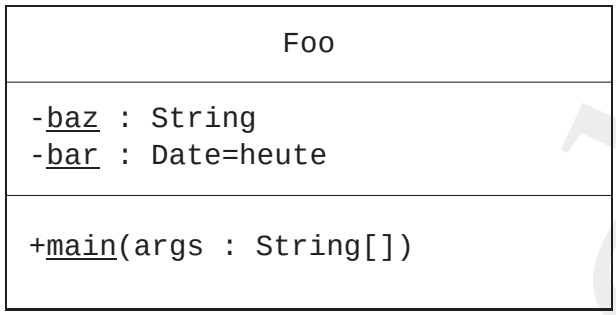


Abbildung 6.5: Klassendiagramm für Foo

Klassendiagramm für die Applikation Foo.

args

```
public static void main(String[] args) { ... }
```

Klasse Foo

```
/**
 *  Kleiner Java Spaß: Demonstration der Bindung des
 *  Parameters an die Argumente ("call by value")
 *
 *@author     Hinrich Bonin
 *@version    2
 *@since      16-Apr-2001
 */
import java.util.*;

public class Foo
{

   private static String baz;
   private static Date bar = new Date();


   public static void main(String[] args)
   {
```

```
      for (int i = 0; i < args.length; i = i + 1)
      {
         System.out.println(
               "Eingabeteil:" + i + "\n" +
               "*" + args[i] + "*");
      }
      if (args.length != 0)
      {
         args[args.length - 1] = "Neuer Wert: ";
         baz = args[args.length - 1];
      } else
      {
         baz = "Kein Argument: ";
      }
      System.out.println(
            baz +
            "Java ist interessant! " +
            bar.toString());
   }
}
```

Nach dem Aufruf von:

`>java Foo %PROCESSOR_IDENTIFIER%`
ist der Wert des Arguments `PROCESSOR_IDENTIFIER` nicht verändert, weil die Shell des Betriebssystems die Variable `PROCESSOR_IDENTIFIER` als Wert übergibt.

Compilation und Ausführung von `Foo`:

```
C:\bonin\anwd>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd>javac code/Foo.java

C:\bonin\anwd>dir Foo.class
```

```
Datei nicht gefunden

C:\bonin\anwd>cd code

C:\bonin\anwd\code>dir Foo.class
 1.009 Foo.class

C:\bonin\anwd\code>java Foo
Kein Argument: Java ist interessant! Sat Dec 14 19:43:58 CET 2002

C:\bonin\anwd\code>echo %PROCESSOR_IDENTIFIER%
x86 Family 15 Model 2 Stepping 7, GenuineIntel

C:\bonin\anwd\code>java Foo %PROCESSOR_IDENTIFIER%
java Foo %PROCESSOR_IDENTIFIER%
Eingabeteil:0
*x86*
Eingabeteil:1
*Family*
Eingabeteil:2
*15*
Eingabeteil:3
*Model*
Eingabeteil:4
*2*
Eingabeteil:5
*Stepping*
Eingabeteil:6
*7,*
Eingabeteil:7
*GenuineIntel*
Neuer Wert: Java ist interessant! Sat Dec 14 20:00:43 CET 2002

C:\bonin\anwd\code>echo %PROCESSOR_IDENTIFIER%
x86 Family 15 Model 2 Stepping 7, GenuineIntel

C:\bonin\anwd\code>
```

6.1.3 Kostprobe `FahrzeugProg` — Konstruktor

In diesem Beispiel einer Applikation sind drei Klassen definiert, um die zwei Fahrzeuge `myVolo` und `myBianchi` zu konstruieren:

- `class Fahrzeug`
  Sie ist die eigentliche „fachliche“ Klasse und beschreibt ein Fahrzeug durch die drei Attribute (↪ Abschnitt 4.1 S. 53):
  - Geschwindigkeit
  - Fahrtrichtung
  - Eigentümer
- `class FahrzeugProg`
  Sie enthält die Methode `main()`. Diese Klasse entspricht dem „Steuerungsblock“ eines (üblichen) imperativen Programmes.
- `class Fahrt`
  Sie dient zum Erzeugen eines „Hilfsobjektes“. Ein solches Objekt wird einerseits als Argument und andererseits als Rückgabewert der Methode `wohin()` genutzt. Damit wird gezeigt wie mehrere Einzelwerte zusammegefaßt von einer Methode zurück gegeben werden können.

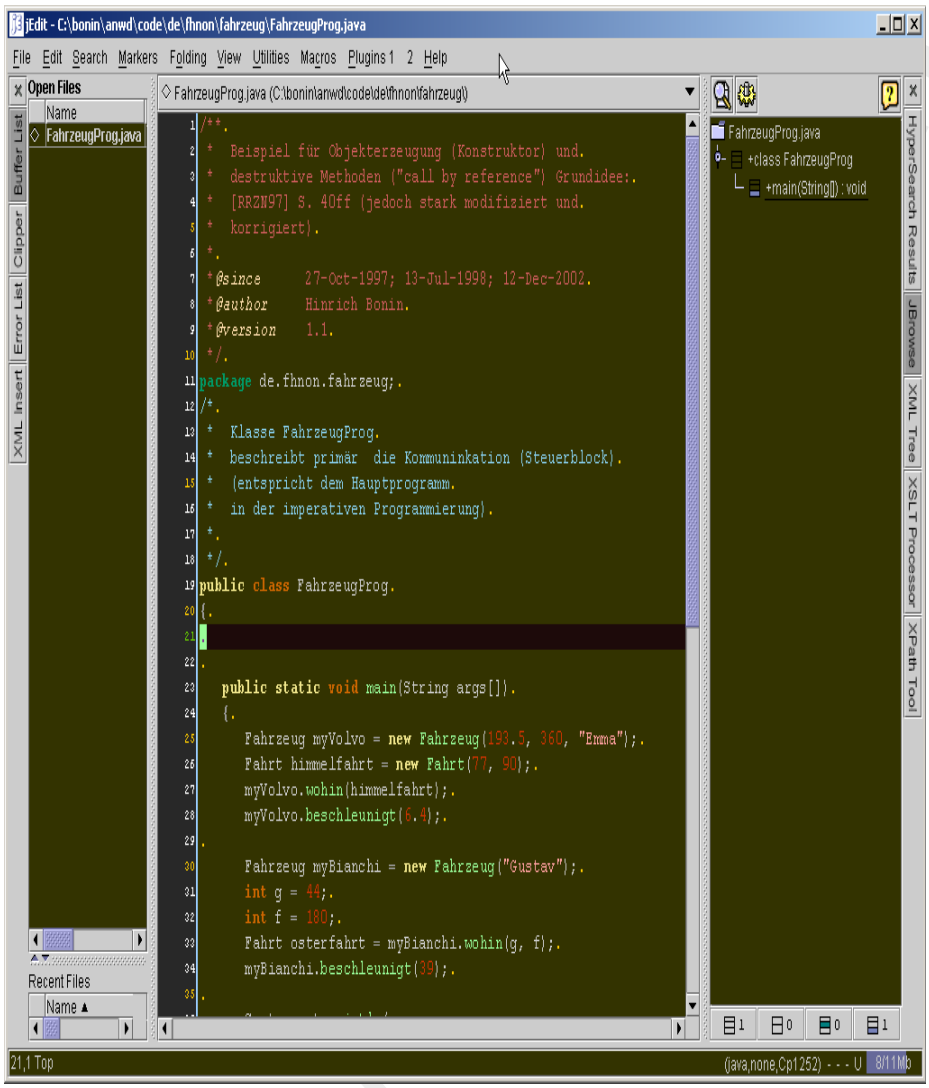
Diese Klassen befinden sich jeweils in einer eigenen Quellcodedatei mit der Extension (dem Suffix) `.java`. (Im „Editor“ `jEdit` ↪ Abbildung 6.6 S. 96 und im `GNU Emacs` ↪ Abbildung 6.7 S. 97). Die Quellcodedatei `FahrzeugProg.java` enthält die namensgleiche Klasse `FahrzeugProg` mit der `main`-Methode[2] enthält. Beim Compilieren der Quellcodedateien entstehen die drei Klassendateien:

- `Fahrt.class`
- `Fahrzeug.class`
- `FahrzeugProg.class`

Um eine Ordnung in die vielen Klassen zu bekommen, werden mehrere Klassen zu einem Paket (*package*) zusammengefaßt. Hier wurde als Paketname:

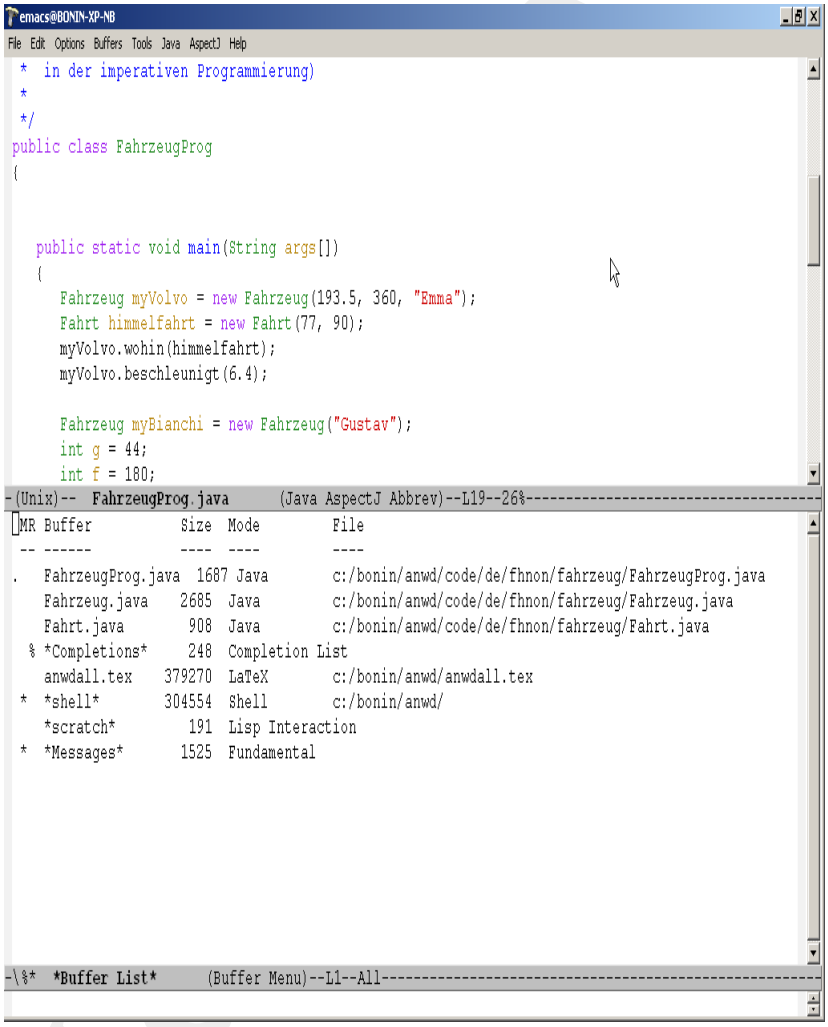
`de.fhnon.fahrzeug` **Paket**

[2] [Hinweis: Jede Klasse kann — zum Beispiel zum Testen — eine Methode `main()` enthalten. Entscheidend ist die Methode `main()` der Klasse, die vom Java-Interpreter aufgerufen wird.]



Legende:
Editor `jEdit` Version 4.1 pre 5 ↪ `http://www.jedit.org`

Abbildung 6.6: Beispieldateien im Editor `jEdit`



Legende:
Editor GNU Emacs Version 21.2.1
↪ http://www.gnu.org/software/software.html

Abbildung 6.7: Beispieldateien im Editor GNU Emacs

gewählt. Die drei Klassendateien sind relativ zum Pfad, der in `CLASSPATH` angegeben ist, zu speichern. Die Abbildung 6.8 S. 99 zeigt das Klassendiagramm für die Applikation `FahrzeugProg`.

Die Compilierung und der Aufruf der Klasse `FahrzeugProg` erfolgt mit dem vorangestellten Paketnamen, das heißt für diese Beispiel auf einer Windows-Plattform (XP) wie folgt (↪ Protokoll S.106):

```
C:\bonin\anwd\code>javac de/fhnon/fahrzeug/*.java
C:\bonin\anwd\code>java de.fhnon.fahrzeug.FahrzeugProg
```

Dieses Beispiel wird auf einer Unix-Plattform (IBM AIX) folgendermaßen compiliert und appliziert:

```
cl3:/home/bonin:>echo $CLASSPATH %$
/u/bonin/myjava:/usr/lpp/J1.1.6/lib/classes.zip:/usr/lpp/J1.1.6/lib:.
cl3:/home/bonin:>javac ./myjava/de/fhnon/fahrzeug/FahrzeugProg.java
cl3:/home/bonin:>java de.fhnon.fahrzeug.FahrzeugProg
```

[Hinweis: Ursprünglich wurden die Namensteile der Angabe für der Java^{TM} Runtime Environment mit einem Punkt (`.`) getrennt angebenen, während die Angabe für den Java-Compiler die übliche Pfadtrennzeichen aufweist, also mit Slash (`/`) in der UNIX-Welt bwz. Backlslash (`\`) in der Windows-Welt.]

Klasse `Fahrzeug`

```
/**
 *  Beispiel für Objekterzeugung (Konstruktor) und
 *  destruktive Methoden ("call by reference") Grundidee:
 *  [RRZN97] S. 40ff (jedoch stark modifiziert und
 *  korrigiert)
 *
 *@since      27-Oct-1997; 13-Jul-1998; 12-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.1
 */
package de.fhnon.fahrzeug;
/*
 *  Klasse Fahrzeug als "fachliches Objekt"
 *
 */
```

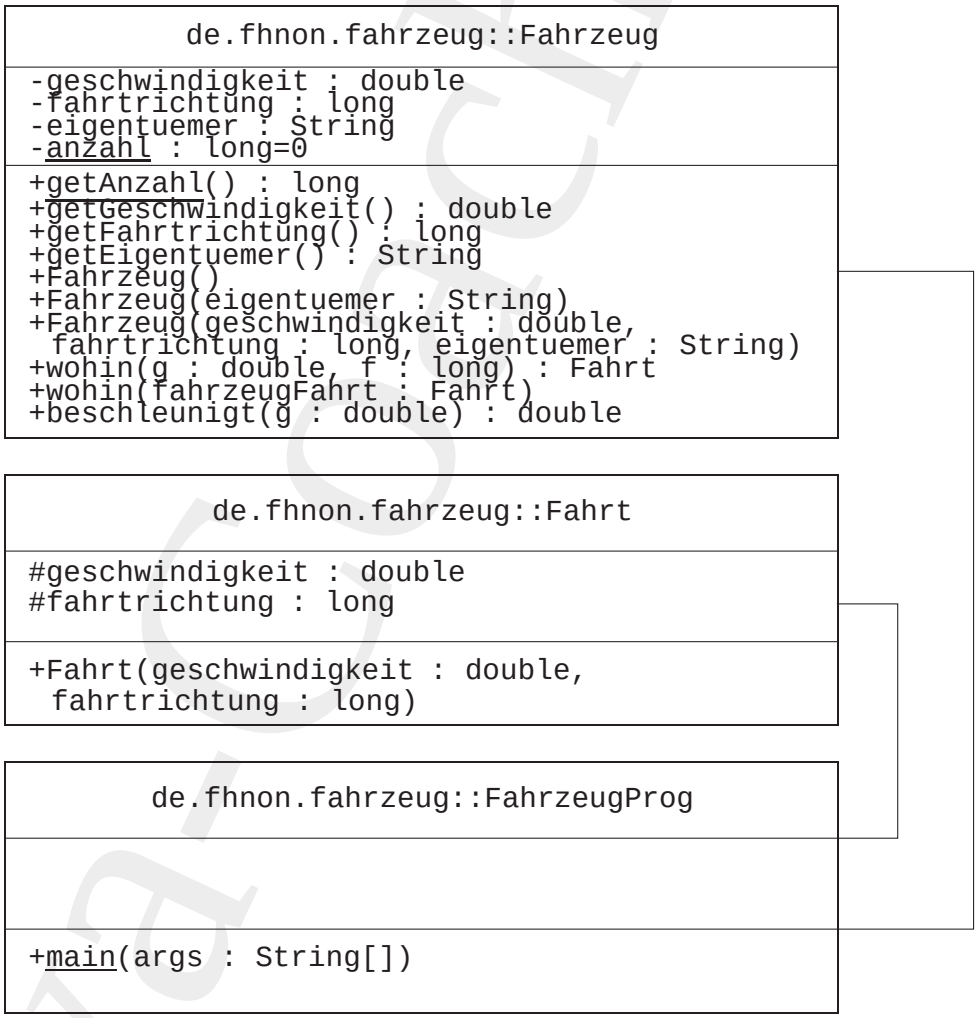


Abbildung 6.8: Klassendiagramm für `FahrzeugProg`

```
class Fahrzeug
{

   private double geschwindigkeit;
   private long fahrtrichtung;
   private String eigentuemer;

   private static long anzahl;
   /*
    *  Static Initialization Block zum
    *  setzten der Anfangszuweisungen
    *  (kein Rückgabewert!)
    */
   static
   {
      anzahl = 0;
   }


   public static long getAnzahl()
   {
      return anzahl;
   }


   /*
    *  Datenkapselung in Klasse Fahrzeug, daher
    *  Selektoren als Methoden definiert.
    */
   public double getGeschwindigkeit()
   {
      return geschwindigkeit;
   }


   public long getFahrtrichtung()
   {
      return fahrtrichtung;
   }
```

```
public String getEigentuemer()
{
   return eigentuemer;
}


/*
 *  Standardkonstruktor für "fachliche Objekte"
 */
public Fahrzeug()
{
   anzahl = anzahl + 1;
}


/*
 *  Konstruktor mit einem Parameter eigentuemer
 *  nutzt Standard-Konstruktor um
 *  Instanz zu gründen.
 */
public Fahrzeug(String eigentuemer)
{
   this();
   this.eigentuemer = eigentuemer;
}


/*
 *  Konstruktor nutzt Konstruktor-Hierarchie
 *  Fahrzeug(eigentuemer) --> Fahrzeug()
 */
public Fahrzeug(double geschwindigkeit,
      long fahrtrichtung, String eigentuemer)
{
   this(eigentuemer);
   this.geschwindigkeit = geschwindigkeit;
   this.fahrtrichtung = fahrtrichtung;
}


/*
 *  Gibt neue Instanz von Fahrt mit gewünschter
```

```
     *  Geschwindigkeit  und Fahrtrichtung zurück
     *  Bei einem einfachen Datentyp wird
     *  der Wert übergeben.
     *  Entspricht "call by value".
     */
    public Fahrt wohin(double g, long f)
    {
       Fahrt fahrzeugFahrt = new Fahrt(g, f);
       return fahrzeugFahrt;
    }


    /*
     *  Modifiziert eine übergebene Instanz mit
     *  den Werten des Objektes auf das die
     *  Methode angewendet wurde.
     *  Ein Objekt und ein Array werden
     *  als Referenz übergeben.
     *  Entspricht "call by reference"
     */
    public void wohin(Fahrt fahrzeugFahrt)
    {
       fahrzeugFahrt.geschwindigkeit = geschwindigkeit;
       fahrzeugFahrt.fahrtrichtung = fahrtrichtung;
    }


    /*
     *  Erhöht die Geschwindigkeit um einen festen Wert
     */
    public double beschleunigt(double g)
    {
       geschwindigkeit = geschwindigkeit + g;
       return geschwindigkeit;
    }
}
```

Klasse `FahrzeugProg`

```
/**
 *  Beispiel für Objekterzeugung (Konstruktor) und
 *  destruktive Methoden ("call by reference") Grundidee:
 *  [RRZN97] S. 40ff (jedoch stark modifiziert und
```

```
 *  korrigiert)
 *
 *@since      27-Oct-1997; 13-Jul-1998; 12-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.1
 */
package de.fhnon.fahrzeug;
/*
 *  Klasse FahrzeugProg
 *  beschreibt primär  die Kommuninkation (Steuerblock)
 *  (entspricht dem Hauptprogramm
 *  in der imperativen Programmierung)
 *
 */
public class FahrzeugProg
{


   public static void main(String args[])
   {
      Fahrzeug myVolvo = new Fahrzeug(193.5, 360, "Emma");
      Fahrt himmelfahrt = new Fahrt(77, 90);
      myVolvo.wohin(himmelfahrt);
      myVolvo.beschleunigt(6.4);

      Fahrzeug myBianchi = new Fahrzeug("Gustav");
      int g = 44;
      int f = 180;
      Fahrt osterfahrt = myBianchi.wohin(g, f);
      myBianchi.beschleunigt(39);

      System.out.println(
            "Fahrzeuganzahl: " +
            Fahrzeug.getAnzahl() + "\n" +
            "Richtung der Himmelfahrt: " +
            himmelfahrt.fahrtrichtung + "\n" +
            "Richtung der Osterfahrt: " +
            osterfahrt.fahrtrichtung + "\n" +
            "myVolvo: " +
            myVolvo.getGeschwindigkeit() + " | " +
            myVolvo.getFahrtrichtung() + " | " +
            myVolvo.getEigentuemer() + "\n" +
```

```
            "myBianchi: " +
            myBianchi.getGeschwindigkeit() + " | " +
            myBianchi.getFahrtrichtung() + " | " +
            myBianchi.getEigentuemer());

      /*
       *  Referenz auf ein Objekt freigaben
       */
      myVolvo = null;
      /*
       *  Garbage Collector aufrufen
       */
      System.gc();
   }
}
```

Klasse `Fahrt`

```
/**
 *  Beispiel für Objekterzeugung (Konstruktor) und
 *  destruktive Methoden ("call by reference") Grundidee:
 *  [RRZN97] S. 40ff (jedoch stark modifiziert und
 *  korrigiert)
 *
 *@since      27-Oct-1997; 13-Jul-1998; 12-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.1
 */
package de.fhnon.fahrzeug;
/*
 *  Die Klasse Fahrt als "Hilfsobjekt".
 *  Sie ist definiert, um für eine Methode den
 *  Rückgabewert einer solchen Instanz zu haben.
 */
class Fahrt
{
   protected double geschwindigkeit;
   protected long fahrtrichtung;


   /*
    *  Konstruktor eines Objektes Fahrt
    *  mit zwei Parametern, die den gleichen Namen
```

```
     *  wie die Datenkomponenten (Slots) haben.
     *  this-Referenzierung daher erforderlich
     */
    public Fahrt(double geschwindigkeit,
          long fahrtrichtung)
    {
       this.geschwindigkeit = geschwindigkeit;
       this.fahrtrichtung = fahrtrichtung;
    }
}
```

Im obigen Beispiel `FahrzeugProg` wird das Fahrzeug `MyVolvo` mit:

- der `geschwindigkeit` = 193.5,
- der `fahrtrichtung` = 360 und
- dem `eigentuemer` = `"Emma"`

angelegt. Außerdem wird die Fahrt `Himmelfahrt` mit:

- der `geschwindigkeit` = 77 und
- der `fahrtrichtung` = 90

angelegt. Durch die Anwendung der „destruktiven“ Methode:

```
wohin(himmelfahrt)
```

auf das Objekt `myVolvo` werden die Werte des Objektes `himmelfahrt` geändert, obwohl `himmelfahrt` nur als Argument übergeben wurde. Da `himmelfahrt` ein *ReferenceType* (↪ Tabelle 6.5 S. 150) ist, wird das Objekt als Referenz und nicht als Wert übergeben. Zur Erzeugung des Objektes `osterfahrt` wird
`myBianchi.wohin(g,f)`
ausgeführt. Bei dieser Methode sind die Parameter vom Typ `double` und `long`, das heißt einfache Datentypen (*PrimitiveType*). Die Argumente werden daher als Werte und nicht als Referenzen übergeben.

In der Klasse `Fahrzeug` sind zwei namensgleiche Methoden `wohin()` definiert. Die Entscheidung der jeweils anzuwendenden Methode `wohin()` erfolgt über den Vergleich der Anzahl und des Typs der Parameter mit

den jeweils angegebenen Argumenten.

[Hinweis: Einfache Datentypen (↪ Tabelle 6.6 S. 151) werden stets durch ihren Wert übergenen. Bei einem „zusammengesetzten“ Objekt und einem Array (*ReferenceType* ↪ Tabelle 6.5 S. 150) wird die Referenz auf das Objekt übergeben.]

Compilation und Ausführung von `FahrzeugProg`:

```
C:\bonin\anwd\code>echo %PATH%
C:\Programme\TeXLive\bin\win32;
C:\WINDOWS\system32;C:\WINDOWS;
C:\WINDOWS\System32\Wbem;
c:\programme\java2\j2sdk1.4.0_01;
c:\programme\java2\j2sdkee1.3;
c:\programme\aspectj1.0\bin;
c:\programme\java2\j2sdk1.4.0_01\bin;
c:\programme\java2\j2sdkee1.3\bin

C:\bonin\anwd\code>echo %CLASSPATH%
.;c:\programme\java2\j2sdk1.4.0_01\lib\tools.jar;
c:\programme\aspectj1.0\lib\aspectjrt.jar;
c:\programme\jxta_demo\lib\jxta.jar;
c:\programme\jxta_demo\lib\jxtacms.jar;
c:\programme\jxta_demo\lib\jxtaptls.jar;
c:\programme\jxta_demo\lib\jxtasecurity.jar;
c:\programme\jxta_demo\lib\jxtashell.jar;
c:\programme\jxta_demo\lib\log4j.jar;
c:\programme\jxta_demo\lib\minimalBC.jar;
c:\programme\jxta_demo\lib\org.mortbay.jetty.jar;
c:\programme\jxta_demo\lib\beepcore.jar;
c:\programme\jxta_demo\lib\cmsshell.jar;
c:\programme\jxta_demo\lib\cryptix32.jar;
c:\programme\jxta_demo\lib\cryptix-asn1.jar;
c:\programme\jxta_demo\lib\instantp2p.jar;
c:\programme\jxta_demo\lib\javax.servlet.jar

C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>dir Fahr*
```

```
Datei nicht gefunden

C:\bonin\anwd\code>javac de/fhnon/fahrzeug/*.java

C:\bonin\anwd\code>java de.fhnon.fahrzeug.FahrzeugProg
Fahrzeuganzahl: 2
Richtung der Himmelfahrt: 360
Richtung der Osterfahrt: 180
myVolvo: 199.9 | 360 | Emma
myBianchi: 39.0 | 0 | Gustav

C:\bonin\anwd\code>cd de/fhnon/fahrzeug

C:\bonin\anwd\code\de\fhnon\fahrzeug>dir
          310 Fahrt.class
          907 Fahrt.java
        1.229 Fahrzeug.class
        2.685 Fahrzeug.java
        1.543 FahrzeugProg.class
        1.687 FahrzeugProg.java

C:\bonin\anwd\code\de\fhnon\fahrzeug>
```

6.1.4 Kostprobe `Counter` — Eingabe von Konsole

Der kommandozeilengesteuerte Zähler wird bei Eingabe eines Pluszeichens inkrementiert, das heißt um den Wert 1 erhöht, und bei Eingabe eines Minuszeichens dekrementiert, das heißt um den Wert 1 verringert. (Idee für das Beispiel ↪ [Schader+03] S. 2–3)

Das Beispiel wurde im Java-Entwicklungswerkzeug *Eclipse*[3] (Version 2.1) erstellt.

Klasse `Counter`

```
/*
 * Created on 29.10.2003
 *
 */
package de.fhnon.nemo.counter;

/**
```

[3] Nähere Angaben zum Werkzeug *Eclipse* ↪ Abbildung 6.3 S. 87 und ↪ Abschnitt 8.1 S. 321.

```
 * @author bonin
 *
 */
public class Counter {
        private int value;

        /**
         * @return
         */
        public int getValue() {
                return value;
        }

        /**
         * @param i
         */
        public void setValue(int i) {
                value = i;
        }
        public void increment() {
                ++value;
        }
        public void decrement() {
                --value;
        }
}
```

Klasse `CounterApplication`

```
/*
 * Created on 29.10.2003
 *
 */
package de.fhnon.nemo.counter;
import java.io.*;
/**
 * @author bonin
 *
 */
public class CounterApplication {

        public static void main(String[] args) {
                Counter c = new Counter();
```

```
                int input = -1;
                do {
                        System.out.println(
                                "Counter State: "
                                        + c.getValue()
                                        + "\n"
                   + " Please the next action (+/-/e)? ");
                        do {
                                try {
                                        input = System.in.read();
                                } catch (IOException e) {
                                }
                        } while (input != '+'
                                 && input != '-'
                                 && input != 'e');
                        if (input == '+') {
                                c.increment();
                        } else if (input == '-') {
                                c.decrement();
                        }
                }
                while (input != 'e');
        }
}
```

6.1.5 Kostprobe `Essen` — Eingabe von Konsole

Die Tabelle 6.1 S. 110 zeigt einfache Regeln zur Thematik „Essen“. Die Klasse `Essen` bildet diese Entscheidungstabelle mit Hilfe der Klasse `Console` ab.

Klasse `Essen`

```
/**
 *  Beispiel: Abbildung der ET-Essen ---
 *  Dateneingabe per Konsole
 *
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.essen;
```

ET-Essen		R1	R2	R3	R4
B1	Gewicht $\leq$ 75 [kg]?	J	J	N	N
B2	Körperlänge $\leq$ 180 [cm]?	J	N	J	N
A1	Dringend abnehmen!			X	
A2	Mehr essen!		X		
A3	Weiter so!	X			X

Legende:
Sehr einfache, begrenzte Eintreffer-Entscheidungstabelle zur Ernährung.

Tabelle 6.1: Beispiel: ET-Ernährung

```
import java.io.*;

public class Essen
{

   public static void main(String[] args) throws IOException
   {
      final int kgGrenze = 75;
      final int cmGrenze = 180;

      int gewicht = Console.readInteger(
            "Gewicht in [kg] eingeben: ");
      int laenge = Console.readInteger(
            "Körperlänge in [cm] eingeben: ");
      if ((gewicht <= kgGrenze) & (laenge <= cmGrenze))
      {
         System.out.println("Weiter so!");
      } else if ((gewicht <= kgGrenze) &
            !(laenge <= cmGrenze))
      {
         System.out.println("Mehr essen!");
      } else if (!(gewicht <= kgGrenze) &
            (laenge <= cmGrenze))
      {
         System.out.println("Dringend abnehmen!");
      } else if (!(gewicht <= kgGrenze) &
            !(laenge <= cmGrenze))
```

```
      {
         System.out.println("Weiter so!");
      } else
      {
         System.out.println("Logikfehler!");
      }
   }
}
```

Klasse `Console`

```
/**
 *  Beispiel: Abbildung der ET-Essen ---
 *  Dateneingabe per Konsole
 *
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.essen;

import java.io.*;

public class Console
{

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


   public static String readString(String message)
         throws IOException
   {
      System.out.println(message);

      return input.readLine();
   }


   public static boolean readBoolean(String message)
         throws IOException
```

```
{
   System.out.println(message);

   return Boolean.valueOf(input.readLine()).booleanValue();
}


public static char readChar(String message)
       throws IOException
{
   System.out.println(message);

   return input.readLine().charAt(0);
}



public static double readDouble(String message)
       throws IOException
{
   System.out.println(message);

   return Double.parseDouble(input.readLine());
}


public static int readInteger(String message)
       throws IOException
{
   System.out.println(message);

   return Integer.parseInt(input.readLine());
}


public static long readLong(String message)
       throws IOException
{
   System.out.println(message);

   return Long.parseLong(input.readLine());
}
```

```
}
```

Compilation von `Essen`, `Console` und Ausführung:

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/essen/*.java

D:\bonin\anwd\code>java de.fhnon.essen.Essen
Gewicht in [kg] eingeben:
85
Körperlänge in [cm] eingeben:
179
Dringend abnehmen!

D:\bonin\anwd\code>
```

6.1.6 Kostprobe `Ei` & `Huhn` — Compilieren

Beziehen sich zwei Klassen aufeinander (*Assoziation*) dann wird beim Compilieren einer Klasse die referenzierte Klasse ebenfalls compiliert. Beim sogenannten „Ei-Huhn-Problem“ geht es um eine Klasse `Huhn` (↪ S. 115) die sich auf eine Klasse `Ei` (↪ S. 113) bezieht und umgekehrt. Wird nun die Klasse `Ei` compiliert, dann wird die Klasse `Huhn` benötigt. Ist sie nur im Quellcode vorhanden und wird sie ebenfalls compiliert. Die Protokolldatei (↪ S. 116) verdeutlicht dieses implizite Compilieren.

Klasse `Ei`

```
/**
 *  Beispiel: Kompilierung von Klassen --- das sogenannte
 *  "Ei-Huhn-Problem"
 *
 *@since      23-Dec-2002
```

```
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.huhn;

public class Ei
{

   private int gewicht;
   private Huhn mutter;



   public int getGewicht()
   {
      return gewicht;
   }


   public void setGewicht(int gewicht)
   {
      this.gewicht = gewicht;
   }


   public Huhn getMutter()
   {
      return mutter;
   }


   public void setMutter(Huhn mutter)
   {
      this.mutter = mutter;
   }


   public Ei() { }


   public Ei(int gewicht, Huhn mutter)
   {
```

```
      this();
      this.setGewicht(gewicht);
      this.setMutter(mutter);

   }

}
```

Klasse Huhn

```
/**
 *  Beispiel: Kompilierung von Klassen --- das sogenannte
 *  "Ei-Huhn-Problem"
 *
 *@since      23-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.huhn;

public class Huhn
{

   private String art;
   private Ei ei;


   public String getArt()
   {
      return art;
   }


   public void setArt(String art)
   {
      this.art = art;
   }


   public Ei getEi()
   {
```

```
      return ei;
   }


   public void setEi(Ei ei)
   {
      this.ei = ei;
   }


   public Huhn() { }


   public Huhn(String art, Ei ei)
   {
      this();
      this.setArt(art);
      this.setEi(ei);

   }


   public static void main(String[] args)
   {
      Huhn myHuhn =
            new Huhn(
            "hybrid", new Ei(60, new Huhn()));
      System.out.println(
            "Das Ei wiegt " +
            myHuhn.getEi().getGewicht() +
            "g.");

   }

}
```

Compilation von `Ei` und Ausführung von `Huhn`:

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
```

```
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>cd de/fhnon/huhn

C:\bonin\anwd\code\de\fhnon\huhn>dir
  743 Ei.java
  947 Huhn.java

C:\bonin\anwd\code\de\fhnon\huhn>cd ../../..

C:\bonin\anwd\code>javac de/fhnon/huhn/Ei.java

C:\bonin\anwd\code>cd de/fhnon/huhn

C:\bonin\anwd\code\de\fhnon\huhn>dir
  687 Ei.class
  743 Ei.java
1.282 Huhn.class
  947 Huhn.java

C:\bonin\anwd\code\de\fhnon\huhn>cd ../../..

C:\bonin\anwd\code>java de.fhnon.huhn.Huhn
Das Ei wiegt 60g.

C:\bonin\anwd\code>
```

6.1.7 Kostprobe `MyNetProg` — Internetzugriff

In diesem Beispiel wird auf ein dynamisches[4] Dokument vom Web-Server:

`http://as.fhnon.de` **URL**

zugegriffen. Verwendet wird dabei die Klasse `URL`[5] des Standardpaketes:

`java.net`. **java.net**

[4] „Dynamisches“ Dokument über CGI-Skript angestoßen (CGI ≡ Common Gateway Interface)

[5] URL ≡ Uniform Resource Locator

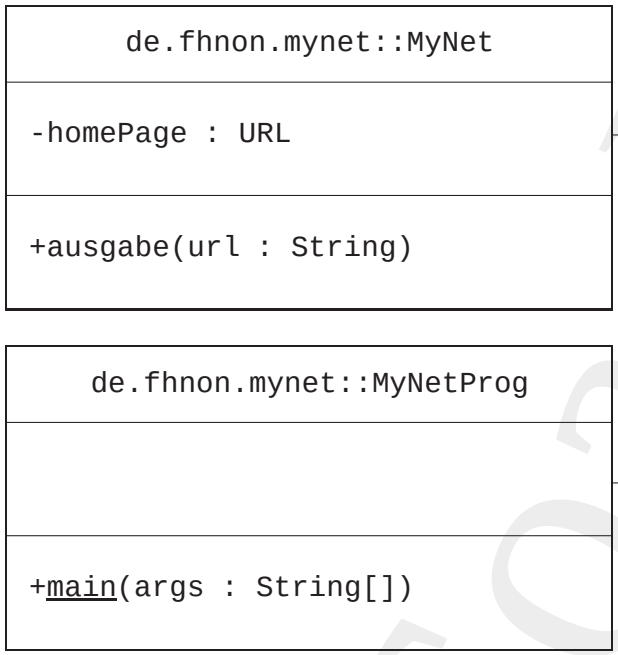


Abbildung 6.9: Klassendiagramm der Applikation `MyNetProg`

Mit dem Konstruktor wird das Objekt `homePage` an einen konkreten URL gebunden. Die Verbindung zum Web-Server wird im Objekt `homePageConnection` vom Typ `URLConnection` abgebildet. Mit der Methode `openConnection()` wird die Verbindung aktiviert und mit der Mehtode `get.InputStream` wird der HTTP[6]-Datenstrom gelesen. Die Abbildung 6.9 S. 118 zeigt das Klassendiagramm der Applikation `MyNetProg`. Die Abbildung 6.10 S. 119 zeigt die nachgefragte Datei `spass.ksh` direkt dargestellt im Browser.

Klasse `MyNet`

```
/**
 *  Beispiel für HTTP-Kommunikation; Grundidee: [RRZN97] S.
```

[6] HTTP ≡ HyperText Transfer Protocol

Legende:
CGI-File `spass.ksh` direkt angezeigt mit Browser *Netscape 7.0* ↪ Abbildung 6.12 S. 133

Abbildung 6.10: Darstellung der CGI-Datei `spass.ksh`

```
 *  111ff
 *
 *@since      28-Oct-1997
 *@author     Hinrich Bonin
 *@create     24-Dec-2002
 *@version    1.1
 */

package de.fhnon.mynet;

import java.net.*;
import java.io.*;

class MyNet
{
   private URL homePage;


   public void ausgabe(String url)
   {
      try
      {
         URL homePage = new URL(url);
         System.out.println(
               "URL: " +
               homePage + "\n" +
               "WWW-Server: " +
               homePage.getHost());

         /*
          *  Verbindung zum Dokument
          */
         URLConnection homePageConnection =
               homePage.openConnection();

         /*
          *  Den  gelieferten HTTP-Datenstrom in ein
          *  DataInputStream wandeln
          */
         DataInputStream in =
               new DataInputStream(
               homePageConnection.getInputStream());
```

```
            /*
             *  Ausgeben zeilenweise
             */
            for (int i = 0; true; i++)
            {
               String line = in.readLine();
               if (line == null)
               {
                  break;
               }
               System.out.println(i + ": " + line);
            }
         } catch (IOException e1)
         {
            // ... hier nicht abgefangen
         }
      }
}
```

Klasse `MyNetProg`

```
/**
 *  Beispiel für HTTP-Kommunikation; Grundidee: [RRZN97] S.
 *  111ff
 *
 *@since      27-Oct-1997
 *@author     Hinrich Bonin
 *@create     24-Dec-2002
 *@version    1.1
 */

package de.fhnon.mynet;

import java.net.*;
import java.io.*;

public class MyNetProg
{

   public static void main(String[] args)
   {
```

```
      MyNet netObject = new MyNet();

      /*
       *  Fest verdrahtete URL-Angabe
       */
      netObject.ausgabe(
            "http://as.fhnon.de/cgi-bin/spass.ksh"
            );
   }
}
```

Compilation und Ausführung von `MyNetProg`:

Hier wird der Java-Compiler mit der Option `deprecation` („Mißbilligung“) aufgerufen. Damit wird der Text der Warnungen ausgegeben. Die Methode `readLine()` der Klasse `DataInputStream` liest Zeichen aus einem *Stream* bis sie auf ein *Newline*-Zeichen, ein *Carriage Return* oder auf beide hintereinander trifft.

Protokoll `MyNetProg.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac -deprecation de/fhnon/mynet/*.java
de/fhnon/mynet/MyNet.java:51: warning:
  readLine() in java.io.DataInputStream has been deprecated
           String line = in.readLine();
                         ^
1 warning

C:\bonin\anwd\code>java de.fhnon.mynet.MyNetProg
java de.fhnon.mynet.MyNetProg
URL: http://as.fhnon.de/cgi-bin/spass.ksh
WWW-Server: as.fhnon.de
0: <?xml version="1.0" encoding="utf-8" ?>
```

```
1: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2:   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3: <html>
4: <head>
5: <link href="/myStyle.css" rel="stylesheet" type="text/css" />
6: <title>fac(n)</title>
7: </head>
8: <body>
9: <p>Das Datum meines Rechners IBM RS/6000
10: (cl3, IP=193.174.33.106) ist:</p> <pre>
11: Mon Jan  6 16:06:31 NFT 2003
12: </pre>
13: <p> Berechnung der Fakult&auml;t von n=100</p>
14: <p><img src="http://as.fhnon.de/facbild.gif"
15: alt="[Mathematische Notation fac(n)]"  /></p>
16: <pre>
17: Scheme Microcode Version 11.146
18: MIT Scheme running under AIX
19:

20: Scheme saved on Sunday November 21, 1993 at 9:15:23 PM
21:   Release 7.3.0 (beta)
22:   Microcode 11.146
23:   Runtime 14.166
24:
25: 1 ]=> ;;;; Spass-Aufgaben
26: ;;;; notiert im LISP-Dialekt: SCHEME
27: ;;;; Bonin: 10-Apr-95 Update: 06-Sep-95
28: ;;;;
29: ;;; Fakultaetsfunktion
30: (begin
31:
32: (define fac (lambda (n)
33:    (cond ((< n 1) 1)
34:          (#T (* n (fac (- n 1)))))))
35:
36: (fac 100))
    .
    .
    .
74: <p>Copyright Bonin 26-Apr-1995 all rights reserved</p>
75: <address>
76: <a href="mailto:bonin@fhnon.de"
77:    >bonin@fhnon.de</a>
78: <br /><br />
79: <a href="boninid.asc">Public Key (PGP)</a>
```

```
80: <br />
81: </address>
82: <hr />
83: </body>
84: </html>

C:\bonin\anwd\code>
```

Alternativlösung Klasse `MyGetWebPage` [Hinweis: Die Alternativlösung arbeitet direkt mit der Port-Nummer 80, die standardgemäß[7] für das *Hypertext Transfer Protocol* (HTTP) vorgesehen ist.]

```
/**
 *  Beispiel für den Zugriff auf eine Web-Page
 *  Idee: Java Technology Fundamentals Newsletter
 *  8-Mar-2004
 *
 *@author     Hinrich Bonin
 *@create     11-Mar-2004
 *@version    1.0
 */

package de.fhnon.mynet;

import java.io.*;
import java.net.*;

public class MyGetWebPage
{
   public static void main(String[] args)
          throws Exception
   {
      if (args.length != 2)
```

[7] Rechner im Internet kommunizieren mit unterschiedlichen Protokollen, die sich standardmäßig auf folgende Ports beziehen:

Port 21 FTP — File Transfer Program

Port 25 SMTP — Simple Mail Transport Protocol

Port 80 HTTP —Hypertext Transfer Protocol

Port 110 POP3 — Post Office Protocol 3

Port 443 HTTPS — HTTP Secure

```
        {
           System.err.println(
                 "java MyGetWebPage hostname document");
           return;
        }
        String host = args[0];
        String document = args[1];
        InetAddress addr =
              InetAddress.getByName(host);
        Socket socket = new Socket(addr, 80);
        InputStream is = socket.getInputStream();
        OutputStream os = socket.getOutputStream();
        BufferedReader br = new BufferedReader(
              new InputStreamReader(is));
        PrintWriter pw = new PrintWriter(
              new OutputStreamWriter(os));
        pw.print("GET /" +
              document +
              " HTTP/1.0 \n\n");
        pw.flush();
        String line;
        while ((line = br.readLine()) != null)
        {
           // read until EOF
           System.out.println(line);
        }
        pw.close();
        br.close();
     }
}
```

Protokoll `MyGetWebPage.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/mynet/MyGetWebPage.java
```

```
D:\bonin\anwd\code>java de.fhnon.mynet.MyGetWebPage
  as.fhnon.de cgi-bin/spass.ksh
HTTP/1.1 200 OK
Date: Thu, 11 Mar 2004 09:43:36 GMT
Server: Apache/1.3.9 (Unix) PHP/3.0.12 ApacheJServ/1.0
Connection: close
Content-Type: text/html

<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<link href="/myStyle.css" rel="stylesheet" type="text/css" />
<title>fac(n)</title>
</head>
<body>
<p>Das Datum meines Rechners IBM RS/6000
(cl3, IP=193.174.33.106) ist:</p> <pre>
Thu Mar 11 10:43:36 NFT 2004
</pre>
<p> Berechnung der Fakult&auml;t von n=100</p>
<p><img src="http://as.fhnon.de/facbild.gif"
alt="[Mathematische Notation fac(n)]"  /></p>
<pre>
Scheme Microcode Version 11.146
MIT Scheme running under AIX


Scheme saved on Sunday November 21, 1993 at 9:15:23 PM
  Release 7.3.0 (beta)
  Microcode 11.146
  Runtime 14.166

1 ]=> ;;;; Spass-Aufgaben
;;;; notiert im LISP-Dialekt: SCHEME
;;;; Bonin: 10-Apr-95 Update: 06-Sep-95
.
.
.
</body>
</html>

D:\bonin\anwd\code>
```

6.1.8 Kostprobe `ImpulseGenerator` — Thread

Java^{TM} ist eine *Multithreaded Environment*. Während beispielsweise die `main()`-Methode „läuft“ werden andere Aufgaben wie *Garbage Collection* oder *Event Handling* im Hintergrund durchgeführt. Diese Arbeiten sind sogenannte *System-managed Threads*. Eine einfaches Beipiel ist die Aufgabe „Tue etwas jede Sekunde!“. [Hinweis: Üblicherweise arbeiten Systemroutinen auf der Basis von Millisekunden.] Die Klasse `ImpulseGenerator` gibt jede Sekunde die Nachricht „`Tick : n`“ aus und zwar bis die Enter-Taste gedrückt wird.

Klasse `ImpulseGenerator`

```
/**
 *  Example: ImpulseGenerator
 *
 *@author     Hinrich Bonin
 *@version    1.0 17-Mar2004
 */
import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;

public class ImpulseGenerator
{
   public static int i = 0;


   public static void main(String args[])
          throws IOException
   {

      TimerTask task =
         new TimerTask()
         {
            public void run()
            {
               i = i + 1;
               System.out.println("Tick: " + i);
            }
         };
      Timer timer = new Timer();
```

```
        timer.schedule(task, 0, 1000);
        System.out.println("Press ENTER to stop");
        System.in.read(new byte[10]);
        timer.cancel();
    }
}
```

Protokoll `ImpulseGenerator.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac ImpulseGenerator.java

D:\bonin\anwd\code>dir ImpulseGenerator*

  699 ImpulseGenerator$1.class
  889 ImpulseGenerator.class
  743 ImpulseGenerator.java

D:\bonin\anwd\code>java ImpulseGenerator
Press ENTER to stop
Tick: 1
Tick: 2
Tick: 3
Tick: 4


D:\bonin\anwd\code>
```

6.1.9 Kostprobe `ActionApplet` — GUI

Implementierungsvererbung:
Wer von einer Klasse erbt,
der bekommt etwas geschenkt
— er spart ein paar Programmzeilen!

Verhaltensvererbung:
Wer von einem Interface erbert,
der muss etwas tun,

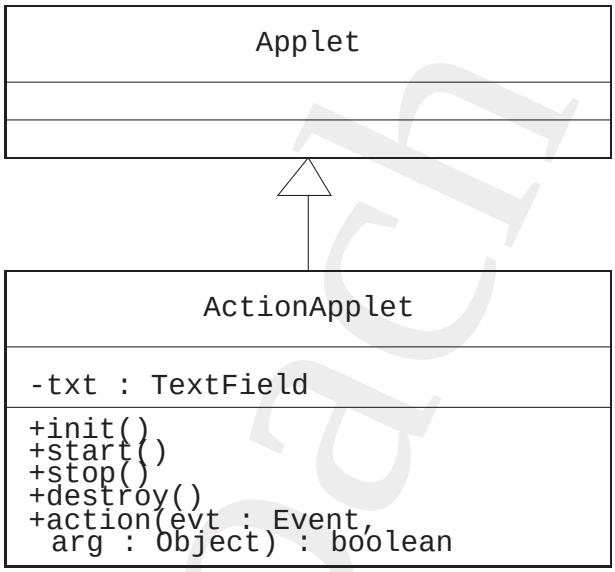


Abbildung 6.11: Klassendiagramm für `ActionApplet`

denn er verpflichtet sich,
das in der Schnittstelle definierte Verhalten
zu implementieren.
(↪ [Broy/Siedersleben02] S. 7)

Die Abbildung 6.11 S. 129 zeigt das Klassendiagramm für `ActionApplet`.

Klasse `ActionApplet`

```
/**
 *  ActionApplet.class zeichnet geschachtelte Panels und
 *  akzeptiert in der Mitte einen Text, der in der
 *  Statuszeile des Browsers und auf der Console
 *  (System.out) ausgegebnen wird. Grundidee: [HSS96]
 *
 *@author     Bonin 22-Jan-1997
 *@version    1.0
 *@since      13-Jul-1998
 */
```

```
import java.awt.*;
import java.applet.*;

public class ActionApplet extends Applet
{
   /*
    *  Textfeld zur Erfassung eines Textes, der
    *  dann in der Statuszeile des Browsers angezeigt wird.
    */
   private TextField txt;


   public void init()
   {
      Font pFont = new Font(
            "Helvetica", Font.BOLD, 24);
      setLayout(new BorderLayout());
      setBackground(Color.white);
      setForeground(Color.green);

      add("North", new Button("Norden"));
      add("South", new Button("Süden"));

      /*
       *  Erzeugt ein Panel p0 mit Struktur im Zentrum
       */
      Panel p0 = new Panel();
      p0.setBackground(Color.red);
      p0.setForeground(Color.white);
      p0.setLayout(new BorderLayout());
      add("Center", p0);
      p0.add("North", new Button("Oben"));
      p0.add("South", new Button("Unten"));

      /*
       *  Erzeugt ein Panel p1 mit Struktur im Zentrum von p0
       */
      Panel p1 = new Panel();
      p1.setBackground(Color.blue);
      p1.setForeground(Color.yellow);
      p1.setLayout(new BorderLayout());
      add("Center", p1);
```

```
        p1.add("North", new Button("Hamburg"));
        p1.add("South", new Button("Hannover"));

        /*
         *  Setzt das Textfeld in die Mitte des inneren Panels
         */
        txt = new TextField(10);
        txt.setFont(pFont);
        p1.add("Center", txt);

        p1.add("East", new Button("Lüneburg"));
        p1.add("West", new Button("Salzhausen"));

        p0.add("West", new Button("Links"));
        p0.add("East", new Button("Rechts"));

        add("West", new Button("Westen"));
        add("East", new Button("Osten"));
    }


    public void start()
    {
    }


    public void stop()
    {
    }


    public void destroy()
    {
    }


    public boolean action(Event evt, Object arg)
    {
        System.out.println(
              ((Button) evt.target).getLabel()
               + ": " + txt.getText());
        showStatus(
```

```
            ((Button) evt.target).getLabel()
             + ": " + txt.getText());
        return true;
    }
}
```

Laden und Ausführen des Applets `ActionApplet`:
Dieses Applet wird mit dem Browser *Netscape 7.0*[8] auf einer Windows-XP-Plattform angezeigt (↪ Abbildung 6.12 S. 133). Es ist in der XHTML-Datei `PruefeApplet.html` (↪ Abschnitt 6.2 S. 132) eingebunden. Die Java-Console des Browsers dokumentiert die Interaktionen des Benutzers (↪ Abbildung 6.13 S. 134).

6.2 Applet-Einbindung in ein Dokument

6.2.1 Applet ⇔ Applikaton

Java^{TM} unterscheidet zwei Ausführungstypen:[9]

Applikation

- Applikation
  Als Applikation bezeichnet man ein „eigenständiges“ Programm, das als „Startmethode“ `main()` enthält und **direkt**, also nicht in einem Browser oder im `appletviewer`, ausgeführt wird..

Applet

- Applet
  Als Applet bezeichnet man ein Programm, das über eine HTML--Seite aufgerufen wird und über diese auch die Aufrufargumente erhält. Das Laden und die Ausführung des Applets steuert der Browser oder der *appletviewer* aus dem `J2SE SDK`. Ein Applet, ursprünglich als kleines Programm gedacht, kann durchaus sehr umfangreich sein. Ein Applet ist eine Unterklasse der Klasse `java.applet.Applet`.

[8] Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE; rv:1.0.1) Gecko/20020823 Netscape/7.0 (BDP)
[9] auch als Programm(formen) bezeichnet

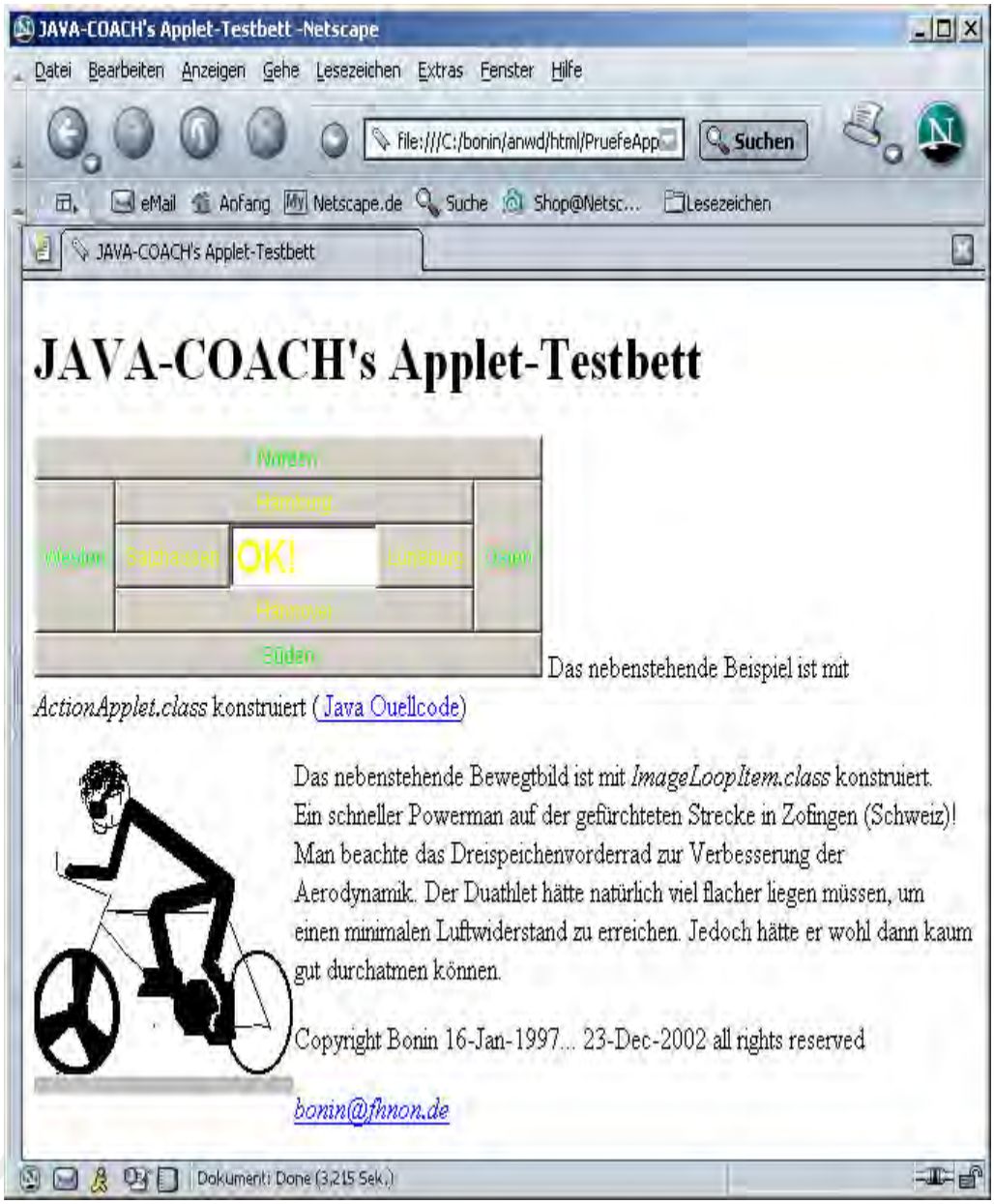
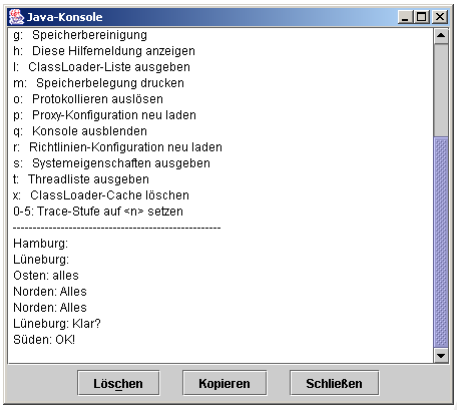


Abbildung 6.12: Beispiel: `ActionApplet`



Legende:
Java Console des Browsers *Netscape 7.0* beim Nutzen von `ActionApplet` ↪ Abbildung 6.12 S. 133

Abbildung 6.13: Beispiel: Java Console von *Netscape 7.0*

6.2.2 HTML-Marken: <object> und <applet>

Ein XHTML-Konstrukt ist definiert:

- durch eine (Anfangs-)Marke, notiert als „<bezeichner>“ und gegebenenfalls
- durch eine Endemarke, notiert als „</bezeichner>“.

Einige Konstrukte haben keine Endemarke oder diese Marke kann entfallen. Zusätzlich zum Bezeichner können Marken Attribute (Argumente) haben, denen über ein Gleichheitszeichen ein Wert zugewiesen werden kann. Der Wert ist in doppelte Hochkommata einzuschließen.[10] Bei der Angabe eines Wertes wird Groß/Klein-Schreibung unterschieden. Die Syntax für ein Konstrukt in XHTML verdeutlicht Tabelle 6.2 S. 135. Sie ist dort rekursiv notiert, da Konstrukte geschachtelt werden können. Ein Konstrukt kann eine Sequenz von weiteren Konstrukten einschließen.

Ein Applet wird in XHTML mit Hilfe des `<object>`-Konstruktes eingebunden. In vorhergehenden HTML-Versionen dient dazu das

[10] Viele *Browser* benötigen jedoch die doppelten Hochkommata nicht (mehr).

Syntax eines XHTML-Konstruktes

$$\overbrace{\underbrace{< bezeichner\ \{attribut[= value]\} >}_{Anfangsmarke}\ [t]\ |\ \{[t]\ k\ [t]\}\ \underbrace{[< /bezeichner >]}_{Endmarke}}^{Konstrukt}$$

zwingend erforderlich — manchmal optional

Legende:
Notation gemäß Backus-Naur-Form (BNF)

`[...]`	≡ das Eingeklammerte kann entfallen (optional)
{...}	≡ das Eingeklammerte kann einmal, mehrmals oder gar nicht vorkommen
a `|` *b*	≡ Alternative, entweder *a* oder `b`
bezeichner	≡ aus Buchstabe(n), manchmal gefolgt von Integerzahl
attribut	≡ Attribut aus ASCII-Zeichen
value	≡ Wert aus ASCII-Zeichen
t	≡ Text aus ASCII-Zeichen
k	≡ Konstrukt

Beispiel: Geschachtelte Konstrukte „`<title>...</title>` in `<head>...</head>`“

```
<head><title>Softwarekonstruktion</title></head>
```

Beispiel: Sequenz der Konstrukte: *Link*, Neue Zeile, Strich („`<a><br><hr>`“)

```
<a href="/w3/media.html">Multimedia</a><br /><hr />
```

Näheres ↪ [Bonin96]

Tabelle 6.2: Syntax eines XHTML-Konstruktes

<applet>-Konstrukt. Das <object>-Konstrukt ermöglicht außer Applets, die auch in anderen Sprachen als Java geschrieben sein können, quasi beliebige Multimedia-Objekte wie zum Beispiel Videos und Bilder syntaktisch einheitlich einzubauen. Der object-Begriff beschreibt hier alle Dinge, die man in ein HTML-Dokument plazieren möchte.

<applet>

Ältere HTML-Versionen (< 4.0**):** <applet>...</applet>

```
<body>
...
<p>
<applet codebase="myPath"
  code="myApplet.class"
  width="300" height="500"
  alt="Mein Logo als tolles Applet myApplet">
  Java myApplet.class: Mein Logo
</applet>
</p>
...
<body>
```

<object>

XHTML: <object>...</object>

```
<body>
...
<p>
<object codetype="application/java"
  codebase="myPath"
  classid="java:myApplet.class"
  width="300" height="500"
  alt="Mein Logo als tolles Applet myApplet">
  Java myApplet.class: Mein Logo
</object>
</p>
...
</body>
```

Syntaktisch gleichartig ist zum Beispiel der Einbau eines Bildes:

```
<body>
<p>Hier ist mein tolles Hundefoto:
<object data="http://www.irgendwo.de/Foo/Edi.png"
  type="image/png">
  Mein tolles Hundefoto.
</object>
</p>
</body>
```

Die Tabelle 6.3 S. 138 beschreibt Attribute des `<object>`-Konstruktes.[11] Das Attribut `align` sollte jedoch entsprechend dem CSS-Konzept (↪ Abschnitt 9.2 S. 376) verwendet werden, das heißt nicht direkt im `<object>`-Konstrukt sondern im `<style>`-Konstrukt.
Hinweis: Nicht jeder marktübliche Browser unterstützt alle Attribute (korrekt).

6.2.3 Beispiel `PruefeApplet.html`

Das Dokument `PruefeApplet.html` umfaßt zwei Applets. Das Applet `ActionApplet` ist über das `<object>`-Konstrukt eingebunden. Für das Applet `ImageLoopItem` wird das „überholte“[12] `<applet>`-Konstrukt genutzt. Das `<style>`-Konstrukt spezifiziert nur das Layout, das heißt hier Farben, Fonts und die Textausrichtung. Seine Wirkungweise wird später eingehend erläutert (↪ Abschnitt 9.2 S. 376).

Das Beispieldokument `PruefeApplet.html` weist die übliche Grundstruktur zum Einbinden eines Applets auf:

```
<!DOCTYPE ...>
<html>
<head>
<title>...</title>
</head>
<body>
...
```

[11] Umfassende, vollständige Beschreibung des `<object>`-Konstruktes siehe HTML4.0-Spezifikation, zum Beispiel:
`http://www.w3.org/TR/REC-html40`

[12] Das `<applet>`-Konstrukt wird im HTML4.0-Standard mißbilligt (≡ *a deprecated element*).

Attribute des `<object>`-Konstruktes	
`classid`=uri	Ort der Objekt-Implementation als URI-Angabe.
`codebase`=uri	Basispfad für die Auflösung der relativen URI-Angaben in `classid`, `data` und `archive`. Bei keiner Angabe wird als Ersatzwert die URI-Basis des aktuellen Dokumentes verwendet.
`codetype`=content-t.	Gibt den erwarteten Datentyp an, wenn ein Objekt, das durch `classid` spezifiziert wurde, geladen wird. Es ist ein optionales Attribut, das ein Laden eines nicht unterstützten Datentyps vermeiden soll.
`data`=uri	Gibt den Ort der Objektdaten an, zum Beispiel für Bilddaten.
`type`=content-t.	Spezifiziert den Datentyp für `data`. Es ist ein optionales Attribut, das ein Laden eines nicht unterstützten Datentyps vermeiden soll.
`archive`=uri list	Eine URI-Liste für die Angabe von Archiven, die relevante Quellen für das Objekt enthalten (Trennzeichen in der Liste ist das Leerzeichen.)
`standby`=text	Text, der angezeigt wird während das Objekt geladen wird.
`width`=length	Angabe für den *User Agent* seinen *Default*-Wert mit der angegebenen Länge (in Pixel) zu überschreiben.
`height`=length	analog zu `width`
`align`=position	Gibt die Objekt-Position bezogen auf den (Kon)Text an. • `left` Linke Rand ist Objekt-Position • `right` Rechte Rand ist Objekt-Position • `bottom` Unterkante fluchtet mit aktueller Basisline • `middle` Mitte fluchtet mit aktueller Basisline • `top` Oberkante fluchtet mit aktueller Basisline

Legende:
uri ≡ *Universal Resource Identifier* (≈ allgemeine Dokumentenadresse)

Tabelle 6.3: Applet-Einbindung: Einige Attribute des `<object>`-Konstruktes

```
<object ...>
  ...
</object>
...
</body>
</html>
```

Die zusätzlichen Angaben wie zum Beispiel die `<meta>`-Konstrukte beschreiben das HTML-Dokument als Ganzes und betreffen hier nicht direkt das `<object>`-Konstrukt.

Datei `PruefeApplet.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Testbett fuer Applets                              -->
<!-- Bonin  13-Jan-1997                                 -->
<!-- Update ... 23-Dec-2002                             -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
<base href="http://as.fhnon.de/" />
<title>JAVA-COACH's Applet-Testbett</title>
<meta http-equiv="Last-Modified"
      content="14-Dec-2002 13:00:00 GMT" />
<meta http-equiv="Expires"
      content="31-Dec-2010 00:00:00 GMT" />
<meta name="KEYWORDS"
      content="Applet-Beispiele, Arbeiten von H. Bonin" />
<meta name="DESCRIPTION"
      content="Bonin, JAVA-COACH" />
<link rev=owns
      title="Hinrich E.G. Bonin"
      href="mailto:bonin@fhnon.de" />
<link href="/myStyle.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h1>JAVA-COACH's Applet-Testbett</h1>
<p class="links">
 <object
  codetype="application/java"
  codebase="ActionApplet"
```

```
  classid="java:ActionApplet.class"
  width="350" height="125"
  standby="Hier kommt gleich was zum Clicken!">
  Java Applet ActionApplet.class
</object>
Das nebenstehende Beispiel ist mit
<em>ActionApplet.class</em> konstruiert
(<a href="http://as.fhnon.de/ActionApplet/ActionApplet.java">
Java Ouellcode</a>)
<br class="freiLassen" /></p>
<p class="links">
<applet code="ImageLoopItem.class"
  width="179" height="175" align="left"
  alt="Der schnelle Powerman!">
  <param name="NIMGS" value="5" />
  <param name="IMG"   value="IrinaRad" />
  <param name="PAUSE" value="0" />
  Java Applet ImageLoopItem:
  Schneller Powerman auf der Radstrecke!
</applet>
Das nebenstehende Bewegtbild ist mit
<em>ImageLoopItem.class</em> konstruiert.
<br />
Ein schneller Powerman auf der gef&uuml;rchteten
Strecke in Zofingen (Schweiz)! Man beachte das
Dreispeichenvorderrad zur Verbesserung der Aerodynamik.
Der Duathlet h&auml;tte nat&uuml;rlich  viel flacher
liegen m&uuml;ssen, um einen minimalen Luftwiderstand
zu erreichen. Jedoch h&auml;tte er wohl dann kaum
gut durchatmen k&ouml;nnen.
<br class="freiLassen" /></p>
<p>
Copyright Bonin 16-Jan-1997... 23-Dec-2002 all rights reserved
</p>
<address>
<a href="mailto:bonin@fhnon.de">bonin@fhnon.de</a>
</address>
</body>
<!-- Ende der Datei PruefeApplet.html -->
</html>
```

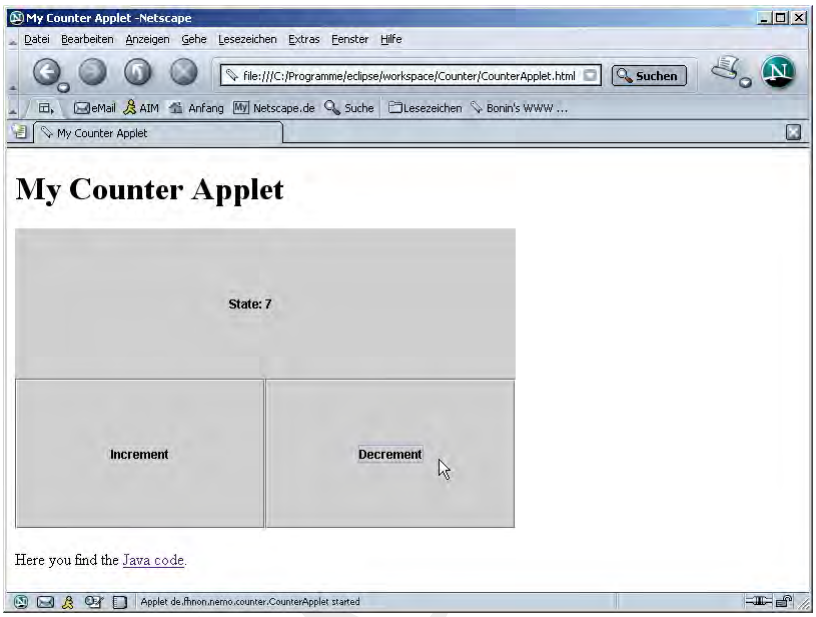


Abbildung 6.14: Beispiel: `CounterApplet`

6.2.4 Beispiel `CounterApplet.html`

Dieses Beispiel entwickelt aus dem als erste Java-Kostprobe dargestellten kommandozeilengesteuerten Zähler (↪ Abschnitt 6.1.4 Seite 107) einen Zähler mit einer graphischen Benutzungsoberfläche. Diese Klasse `CounterGUI` (↪ S. 143) wird dann in der Klasse `CounterApplet` (↪ S. 144) genutzt. Über ein `<object>`-Konstrukt ist das Applet in die XHTML-Datei `CounterApplet.html` (↪ S. 141) eingebunden. Die Abblidung 6.14 Seite 141 zeigt das Ergebnis im Browser *Netscape 7.1*[13].

Datei `CounterApplet.html`

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

[13] Browser *Netscape 7.1* Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE; rv:1.4) Gecko/20030619 Netscape/7.1 (ax)

```
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Bonin Version 1.0  -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="Content-Type"
  content="text/html; charset=utf-8" />
<title>My Counter Applet</title>
</head>
<body>
<h1>My Counter Applet</h1>
<p>
<object codetype="application/java"
  classid="java:de.fhnon.nemo.counter.CounterApplet"
  code="de.fhnon.nemo.counter"
  width="500" height="300">
[Counter Applet --- enable Java to see this applet]
</object>
</p>
<p>Here you find the
<a href="file://localhost/C:/Programme/eclipse/
  workspace/Counter/de/fhnon/nemo/counter/CounterApplet.java">
  Java code</a>.
</p>
</body>
</html>
```

Klasse `Counter`

```
/*
 * Created on 29.10.2003
 *
 */
package de.fhnon.nemo.counter;

/**
 * @author bonin
 *
 */
public class Counter {
        private int value;

        /**
         * @return
```

```
         */
        public int getValue() {
                return value;
        }

        /**
         * @param i
         */
        public void setValue(int i) {
                value = i;
        }
        public void increment() {
                ++value;
        }
        public void decrement() {
                --value;
        }
}
```

Klasse `CounterGUI` Diese Klasse erzeugt zwei Buttons zum Inkrementieren beziehungsweise Dekrementieren mit den entsprechenden Beschriftungen. An den beiden Buttons hängen jeweils ein `ActionListener`-Objekt, dessen Methode `actionPerformed` aufgerufen wird, wenn der Benutzer den Button drückt.

```
/*
 * Created on 29.10.2003
 *
 */
package de.fhnon.nemo.counter;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;
/**
 * @author bonin
 * Idea: Schrader / Schmidt-Thieme 2003 p.4
 */
public class CounterGUI extends JPanel {
        private JButton plus;
        private JButton minus;
        private JLabel state;
        private Counter c;
```

```
        CounterGUI() {
                c = new Counter();
                setLayout(new GridLayout(2, 2));
                add(new JLabel("State: ", JLabel.RIGHT));
                add(state =
                  new JLabel(String.valueOf(c.getValue())));
                add(plus = new JButton("Increment"));
                add(minus = new JButton("Decrement"));
                plus.addActionListener(new ActionListener() {
                        public void
                          actionPerformed(ActionEvent e) {
                                c.increment();
                                state.setText(
                                  String.valueOf(c.getValue()));
                        }
                });
                minus.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                c.decrement();
                                state.setText(
                                  String.valueOf(c.getValue()));
                        }
                });
        }

        public static void main(String[] args) {
                JFrame f = new JFrame("Counter GUI");
                f.getContentPane().add(new CounterGUI());
                f.pack();
                f.setDefaultCloseOperation(
                  JFrame.DISPOSE_ON_CLOSE);
                f.setVisible(true);

        }
}
```

Klasse `CounterApplet`

```
/*
 * Created on 29.10.2003
 *
 */
package de.fhnon.nemo.counter;
```

```
import javax.swing.*;
/**
 * @author bonin
 * Idea: Schrader / Schmidt-Thieme 2003 p.6
 */
public class CounterApplet extends JApplet {
        public void init() {
                getContentPane().add(new CounterGUI());
        }
}
```

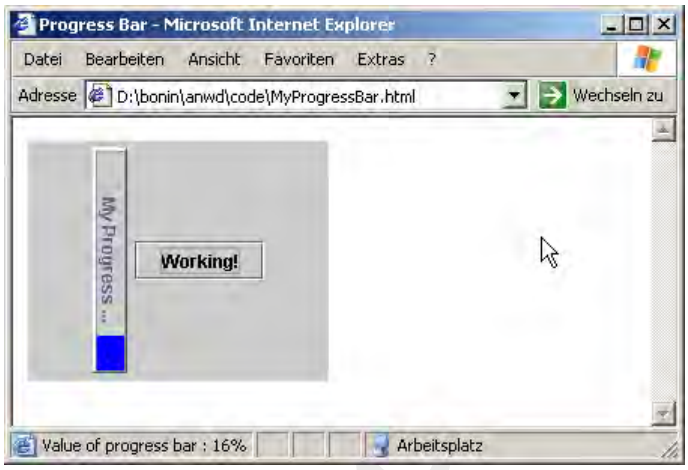
6.2.5 Beispiel `MyProgressBar.html`

Häufig wird ein sich änderndes Fortschrittssymbol, zum Beispiel ein sich füllender Balken (*progress bar*), eingesetzt, um dem Benutzer anzuzeigen, dass das Programm erfolgreich arbeitet. Im Java^{TM}-Swing-API (Application Programming Interface) ist dafür die besondere Klasse `JProgressBar` vorgesehen. Dieses einfache Beispiel verdeutlicht Möglichkeiten dieser Klasse. Dazu simmuliert das Drücken des Button `Working!` einen Programmfortschritt.

Klasse `MyProgressBar`

```
/**
 *  Example: Progress Bar
 *
 *@author     Hinrich Bonin
 *@version    1.0 16-Mar2004
 */
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;

public class MyProgressBar extends JApplet
{
   JProgressBar jpb;
   JButton jb;


   public void init()
```

```
    {
       Container contentPane = getContentPane();
       contentPane.setLayout(new FlowLayout());
       jpb = new JProgressBar();
       // Note the default orientation is horizontal
       jpb.setOrientation(JProgressBar.VERTICAL);
       jpb.setValue(2);
       jpb.setForeground(Color.blue);
       jpb.setStringPainted(true);
       jpb.setBorder(
             BorderFactory.createRaisedBevelBorder());
       jpb.setString("My Progress ...");
       contentPane.add(jpb);

       jb = new JButton("Working!");
       contentPane.add(jb);
       jb.addActionListener(
          new ActionListener()
          {
             public void actionPerformed(
                   ActionEvent e)
             {
                jpb.setValue(
                      jpb.getValue() + 2);
             }
          });

       jpb.addChangeListener(
          new ChangeListener()
          {
             public void stateChanged(
                   ChangeEvent e)
             {
                showStatus(
                     "Value of progress bar : " +
                     jpb.getValue() +
                     "%");
             }
          });
    }
}
```



Legende:
Java Applet `MyProgressBar` dargestellt mit *Microsoft Internet Explorer Version: 6.0*

Abbildung 6.15: Beispiel: `MyProgressBar`

Protokolldatei `MyProgressBar.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac MyProgressBar.java

D:\bonin\anwd\code>dir MyProgressBar*
   628 MyProgressBar$1.class
   888 MyProgressBar$2.class
 1.522 MyProgressBar.class
   557 MyProgressBar.html
 1.476 MyProgressBar.java

D:\bonin\anwd\code>
```

Datei `MyProgressBar.html`

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- My progress bar example                    -->
<!-- Bonin  16-Mar-2004                         -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
  <title>Progress Bar</title>
</head>
<body>
  <p>
    <applet code="MyProgressBar.class"
      width="200" height="160" align="left">
      [Java Applet MyProgressBar]
    </applet>
  </p>
</body>
</html>
```

6.3 Syntax & Semantik & Pragmatik

Die Tabelle 6.4 S. 149 nennt die in Java^{TM} reservierten Wörter. Diese können nicht als Namen für eine eigene Klasse, Schnittstelle, Variable oder Methode verwendet werden. Darüber hinaus sollten die folgenden Methodennamen aus der `Object`-Klasse nicht benutzt werden, es sei denn man möchte die `Object`-Methode überschreiben.

Reservierte Methodennamen: `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString` und `wait`.

In Java^{TM} werden „zusammengesetzte“ Objekte (*ReferenzType*) von einfachen Datentypen (*PrimitiveType*) unterschieden. Die Tabelle 6.5 S. 150 zeigt anhand einer rekursiven Beschreibung die unterschiedlichen Typen.

Wort	Stichworthafte Erläuterung	Wort	Stichworthafte Erläuterung
`abstract`	Deklaration von Klasse / Meth.	`boolean`	einf. Datentyp: `true` / `false`
`break`	Kontrollkonstrukt; terminiert	`byte`	einfacher Datentyp (8 Bit Zahl)
`byvalue`	— nicht genutzt —	`case`	Kontrollkonstrukt; mit `switch`
`cast`	— nicht genutzt —	`catch`	Kontrollkonstrukt; mit `try`
`char`	einfacher Datentyp	`class`	Deklariert eine Klasse
`const`	— nicht genutzt —	`continue`	Kontrollkonstrukt
`default`	Kontrollkonstrukt; mit `switch`	`do`	Kontrollkonstrukt; mit `while`
`double`	einf. Datentyp (64 Bit Fließkom.)	`else`	Kontrollkonstrukt; mit `if`
`extends`	Superklassenangabe	`false`	`boolean`-Wert
`final`	Keine Subklasse; unübersch. M.	`finally`	Kontrollkon.; mit `try/catch`
`float`	einf. Datentyp (32 Bit Fließkom.)	`for`	Kontrollkonstrukt; Iteration
`future`	— nicht genutzt —	`generic`	— nicht genutzt —
`goto`	— nicht genutzt —	`if`	Kontrollkonstrukt; Alternative
`implements`	Implementiert Schnittstelle	`import`	Namensabkürzungen
`inner`	— nicht genutzt —	`instanceof`	Prüft Instanz einer Klasse
`int`	einfacher Datentyp (32 Bit Zahl)	`interface`	Deklariert Schnittstelle
`long`	einfacher Datentyp (64 Bit Zahl)	`native`	„Andere“(C-)Implemtation
`new`	Erzeugt neues Objekt / Array	`null`	„kein Objekt“-Referenze
`operator`	— nicht genutzt —	`outer`	— nicht genutzt —
`package`	Paket; erste Anweisung	`private`	Zugriffsrecht
`protected`	Zugriffsrecht	`public`	Zugriffsrecht
`rest`	— nicht genutzt —	`return`	Kontrollkonstruk; Rückgabe
`short`	einfacher Datenyp (16 Bit Zahl)	`static`	Klassen-Variable / -Methode
`super`	Zugriff auf Super-Klasse	`switch`	Kontrollk.; Fallunterscheidung
`synchronized`	Sperrmechanismus	`this`	Verweist auf „dieses Objekt“
`throw`	Erzeugt Ausnahmeobjekt	`throws`	Deklariert Ausnahmezustände
`transient`	Kein persistenter Objektteil	`true`	`boolean`-Wert
`try`	Kontrollk.; mit `catch/finally`	`var`	— nicht genutzt —
`void`	Kein Rückgabewert	`volatile`	Asynchrone Wertänderung
`while`	Kontrollkonstrukt		

Tabelle 6.4: Reservierte Java^{TM}-SchlüsserWörter

6.3.1 Attribute für Klasse, Schnittstelle, Variable und Methode

Bei der Deklaration einer Klasse, Schnittstelle, Variable oder Methode können Attribute, sogenannte Modifikatoren, angegeben werden. Neben den Modifikatoren für die Zugriffsrechte[14] (Sichtbarkeit) sind es folgende Attribute:

- `static`
  - *Variable*: Die Variable ist eine Klassenvariable (hat „Speicherplatz“ nur in der Klasse) und wird durch den Klassennamen angesprochen.
  - *Methode*: Die Methode ist eine Klassenmethode und ist implizit `final`. Sie wird durch den Klassennamen aufgerufen.
- `abstract`

[14] ↪ Tabelle 6.7 S. 155

Type ≡
    PrimitiveType | *ReferenceType*

PrimitiveType ≡
    NumericType |
    `boolean`

NumericType ≡
    IntegralType | *FloatingPointType*

IntegralType ≡
    `byte` | `short` | `int` | `long` | `char`

FloatingPointType ≡
    `float` | `double`

ReferenceType ≡
    ClassOrInterfaceType | *ArrayType*

ClassOrInterfaceType ≡
    Name

ArrayType≡
    PrimitiveType[] | *Name*[] | *ArrayType*[]

Legende:

`terminal`	≡ definiert als
nonterminal	≡ ist noch weiter zu definieren
a | b	≡ a oder b

Weitere Definitionen ↪ [Arnold/Gosling96]

Tabelle 6.5: Datentypbeschreibung anhand von Produktionsregeln

Typ	Enthält	Standard	Größe in Bit	Werte Minimal	Maximal
`boolean`	`true` oder `false`	`false`	1	—	—
`char`	Unicode Zeichen	`\u0000`	16	`\u0000`	`\uFFFF`
`byte`	Integer mit Vorzeichen	0	8	-128	127
`short`	Integer mit Vorzeichen	0	16	-32768	32767
`int`	Integer mit Vorzeichen	0	32	-2147483648	2147483647
`long`	Integer mit Vorzeichen	0	64	-9223372036854775808	9223372036854775807
`float`	Fließkomma IEEE 754	0.0	32	±3.40282347E+38	±1.40239846E-45
`double`	Fließkomma IEEE 754	0.0	32	±1.79769313486231570E+308	±4.94065645841246544E-324

Quelle: [Flanagan96] Seite 183
Hinweis: Die Größe in Bit ist nicht implementationsabhängig!

Tabelle 6.6: Javas einfache Datentypen

  - *Klasse*: Die Klasse kann keine Instanz haben. Sie kann nicht-implementierte Methode beinhalten.
  - *Schnittstelle*: Eine Schnittstelle ist stets `abstract`. Die Angabe kann daher entfallen.
  - *Methode*: Die einschließende Klasse muß ebenfalls `abstract` sein. Es wird kein Methodenkörper angegeben. Dieser wird von der Subklasse bereitgestellt. Der Methodenangabe (Signatur[15]) folgt ein Semikolon.

- `final`
  - *Klasse*: Von der Klasse kann keine Subklasse gebildet werden.
  - *Methode*: Die Methode ist nicht überschreibbar. Der Compiler kann daher effizienteren Code erzeugen.
  - *Variable*: Der Wert der Variablen ist nicht änderbar. Schon der Compiler kann Ausdrücke auswerten und muß dies nicht dem Interpreter überlassen.

- `native`
  - *Methode*: Die Methode ist in einer maschinennahen, plattformabhängigen Art implementiert, zum Beispiel in C. Es wird kein Methodenbody angegeben und die Nennung wird mit einem Semikolon abgeschlossen.

- `synchronized`
  - *Methode*: Die Methode nimmt eine oder mehrere Veränderungen der Klasse oder einer Instanz vor, wobei sichergestellt ist, das zwei Threads eine Änderung nicht gleichzeitig vornehmen können. Dazu wird die Instanz für den Mehrfachzugriff gesperrt. Bei einer `static`-Methode wird die Klasse gesperrt.

- `transient`

[15] Signatur besteht aus Namen, Anzahl und Typ der Parameter

  - *Variable*: Damit wird gekennzeichnet, daß die Variable kein Teil des persistenten Zustandes des Objektes ist.

- `volatile`
  - *Variable*: Die Variable ändert sich asynchron (zum Beispiel kann sie ein Hardwareregister eines Peripheriegerätes sein). Ihr Wert sollte daher vom Compiler nicht in Registern gespeichert werden.

6.3.2 Erreichbarkeit bei Klasse, Schnittstelle, Variable und Methode

Zur Einschränkung oder Erweiterung des Zugriffs gegenüber keiner Angabe (*default*) dienen die Modifikatoren `public`, `private` und `protected`. Die Tabelle 6.7 S. 155 beschreibt ihre Wirkungen. Praxisrelevante Zugriffssituationen zeigt die Tabelle 6.8 S. 156.

Lfd. Nr.	**Modifikator**	**Erläuterung der Erreichbarkeit**
1	keine Angabe (default)	Wenn kein Zugriffsrecht (Modifikator) angegeben wird, ist eine Klasse, Schnittstelle, Variable oder Methode nur innerhalb des gleichen Paketes zugreifbar.
2	`public`	Eine Klasse oder Schnittstelle ist überall zugreifbar. Eine Methode oder Variable ist überall in der Klasse zugreifbar.
3	`private`	Eine Variable oder Methode ist nur in ihrer eigenen Klasse zugreifbar. • Eine Klasse kann nicht als `private` gekennzeichnet werden.
4	`protected`	Eine Variable oder Methode ist im gesamten Paket ihrer Klasse zugreifbar und in allen Subklassen der Klasse. Eine Subklasse in einem anderen Paket als ihre Superklasse kann auf die `protected`-Einträge zugreifen, die ihre Instanzen geerbt haben, aber sie kann nicht die Einträge in den (direkten) Instanzen der Superklasse erreichen. • Eine Klasse kann nicht als `protected` gekennzeichnet werden.
5	`private protected`	Eine Variable oder Methode ist nur innerhalb ihrer eigenen Klasse und den zugehörigen Subklassen zugreifbar. Eine Subklasse kann auf alle „`private protected`“-Einträge zugreifen, die ihre Instanzen geerbt haben, aber sie kann nicht die Einträge in den (direkten) Instanzen der Superklasse erreichen. • Eine Klasse kann nicht als `private protected` gekennzeichnet werden.

Hinweis: Die Standardzugreifbarkeit (keine Angabe) ist **strikter** als die `protected`-Angabe! (Quelle [Flanagan96] Seite 188)

Tabelle 6.7: Java-Zugriffsrechte für Klasse, Schnittstelle, Variable und Methode

Lfd. Nr.	**Situation**	**Gekenn-zeichnet mit:**
1	Erreichbar für Subklassen eines **anderen** Paketes!	`public`

2	Erreichbar für Subklassen des **gleichen** Paketes!	— (default)
		`public`
		`protected`

3	Erreichbar für **Nicht**-Subklassen eines **anderen** Paketes!	`public`

4	Erreichbar für **Nicht**-Subklassen des **gleichen** Paketes!	— (default)
		`public`
		`protected`

5	Geerbt von Subklassen in einem **anderen** Paket!	`public`
		`protected`
		`private protected`

6	Geerbt von Subklassen im **gleichen** Paket!	— (default)
		`public`
		`protected`
		`private protected`

Tabelle 6.8: Zugriffssituationen

6.3.3 Operator — Priorität und Assoziativität

Die Tabelle 6.9 S. 158 fasst die Operatoren mit ihrer Priorität und Assoziativität zusammmen (Quelle: ↪ [Schader+03] S. 588)

Operator	Funktion	Priorität	Assoziativität
`.`	Zugriff auf Variable oder Methode	14	l
`[]`	Indexoperator	14	l
`()`	Funktionsaufruf	14	l
`+,-`	Vorzeichen	13	r
`++,--`	Inkrement, Dekrement	13	r
$\sim$	bitweise Negation	13	r
!	logische Negation	13	r
`()`	Cast	13	r
`*,/,%`	multiplikative Operatoren	12	l
`+, -`	additive Operatoren	11	l
`<<,>>,>>>`	Shift-Operatoren	10	l
`<,<=,>,<=, instanceof`	relationale Operatoren	9	l
`==,!=`	Gleichheits-Operatoren	8	l
`&`	UND (bitweise bzw. logisch)	7	l
`^`	Exklusiv-ODER (bitweise bzw. logisch)	6	l
`|`	Inklusiv-ODER (bitweise bzw. logisch)	5	l
`&&`	UND (bedingt logisch)	4	l
`||`	Inklusiv-ODER (bedingt logisch)	3	l
`?:`	Konditional-Operator	2	r
`=,*=,/=,%=,+=,-= <<=,>>=,>>>=,&=,^=,|=`	Zuweisungsoperatoren	1	r

Legende:

l $\equiv$ Links-assoziativ

r $\equiv$ Rechts-assoziativ

Ein weiter oben stehender Operator hat höhere Priorität.

Tabelle 6.9: Operator — Priorität und Assoziativität

Kapitel 7

Konstruktionen (Analyse und Synthese)

Primitive Bausteine werden zu komplexen Konstruktionen zusammengefügt. Dabei dreht sich alles um das Wechselspiel zwischen Aufteilen in mehrere kleinere Objekte und Zusammenfassen in größere Objekte. Das Konstruieren ist ein verwobener Prozeß von Analyse- und Synthese-Aktionen.
Im Mittelpunkt stehen die konstruktiven („handwerklichen“) Aspekte des Programmierens und weniger die ästhetischen, wie sie etwa im Leitmotto „The Art of Programming“ zum Ausdruck kommen.

Trainingsplan

Das Kapitel „Konstruktionen“ erläutert:

- die nebenläufige Ausführung von Teilprozessen (*Multithreading*),
  ↪ Seite 161 . . .
- die Behandlung von Ereignissen (Delegationsmodell),
  $hookrightarrow$ Seite 168 . . .
- die Realisierung von persistenten Objekten,
  ↪ Seite 182 . . .
- das Schachteln von Klassen ineinander (*Inner Classes*),
  ↪ Seite 191 . . .
- die Möglichkeit auf die innere Struktur einer Klasse zuzugreifen (*Reflection*),
  ↪ Seite 215 . . .
- das Arbeiten mit einem objekt-orientierten Datenbanksystem am Beispiel `FastObjects t7` der Firma Poet Software
  (↪ `http://www.fastobjects.de` Zugriff 18.12.2001),
  ↪ Seite 228 . . .
- die Zusicherung eines bestimmten Wertes für eine Variable
  ↪ Seite 246 . . .
- das Zusammenarbeiten verteilter Objekte (*Stub, Skeleton, RMI*),
  ↪ Seite 253 . . .
- die Verarbeitung von XML-Daten (`JDOM`) und
  ↪ Seite 281 . . .
- die Modelle Komponenten zu spezifizieren und zu verteilen (*JavaBeans*^{TM} & EJB).
  ↪ Seite 300 . . .

7.1 Nebenläufigkeit (*Multithreading*)

Ein *Thread* („Faden“) ist ein ablauffähiges Codestück, daß nebenläufig zu anderen Codestücken ausgeführt wird. Dabei bedeutet Nebenläufigkeit ein unabhängiges, quasi zeitgleiches (paralleles) Ablaufen der einzelnen *Threads*. Ein solches *Multithreading* basiert auf den folgenden beiden Objekten:

Thread

- Objekt der `class Thread`
  Ein Objekt dieser Klasse dient zur Steuerung, also beispielsweise zum Starten und Beenden des *Thread*.

Runnable

- Objekt einer Klasse vom `interface Runnable`
  Ein Objekt einer Klasse, die das Interface `Runnable` implementiert, bildet das nebenläufig abarbeitbare Objekt. Ein solches Objekt `myThread` wird folgendermaßen erzeugt und gestartet:

  ```
  Thread myThread = new Thread(myRunnable).start;
  ```

  Die Klasse der Instanz `myRunnable` muß die Methode `run()` implementieren, beispielsweise wie folgt:

  ```
  public class MyBesteIdee implements Runnable {
    public void run() {
      // tue was
    }
  }
  ...
  MyBesteIdee myRunnable = new MyBesteIdee();
  ...
  ```

Die Methode `stop()` könnte innerhalb und außerhalb des `Runnable`-Objektes (hier `myRunnable`) appliziert werden. Startbar ist der Thread natürlich nur außerhalb, das heißt, die Methode `start()` kann nur außerhalb des `Runnable`-Objektes appliziert werden. Innerhalb des `Runnable`-Objektes kann das zugehörige `Thread`-Objekt mit Hilfe der Klassenmethode `currentTread()` festgestellt werden.

Das folgende Beispielapplet zeigt ein Textfeld in das drei Zeilen geschrieben werden. Zunächst werden diese drei Zeilen hintereinander erstellt und dann der Reihe nach, also sequentiell, ausgegeben (↪ Seite 162). Danach erfolgt die Ausgabe mit Hilfe von drei *Threads*, für jede Zeile ein eigener *Thread* (↪ Seite 166). In beiden Fällen wird dazu die Klasse `EinfacherAusgeber` benutzt (↪ Seite 164). Dieses einfache Beispiel verdeutlich, daß eine sequentiell konzipierte Lösung durchaus noch in eine nebenläufige verwandelt werden kann. Die Abbildung 7.1 S. 163 zeigt das Klassendiagramm für diese Multithreading-Beispiel.

Teil des HTML-Dokumentes für das Applet `SequenzArbeit.class`

```
<object
   codetype="application/java"
   classid="java:SequenzArbeit.class"
   name="SeqArbeit"
   width="500"
   height="600"
   alt="Kleiner Scherz &uuml;ber SAP.">
   Java SequenzArbeit.class
</object>
```

Klasse `SequenzArbeit`

```
/**
 *  Sequentielles Abarbeiten in einem Applet! Idee bzw. Code
 *  ähnlich: Doug Lea; Concurrent Programming in Java, ISBN
 *  0-201-63455-4
 *
 *@author     Hinrich E. G. Bonin
 *@version    1.0
 *@since      16-Jul-1998
 */

import java.applet.*;
import java.awt.*;
```

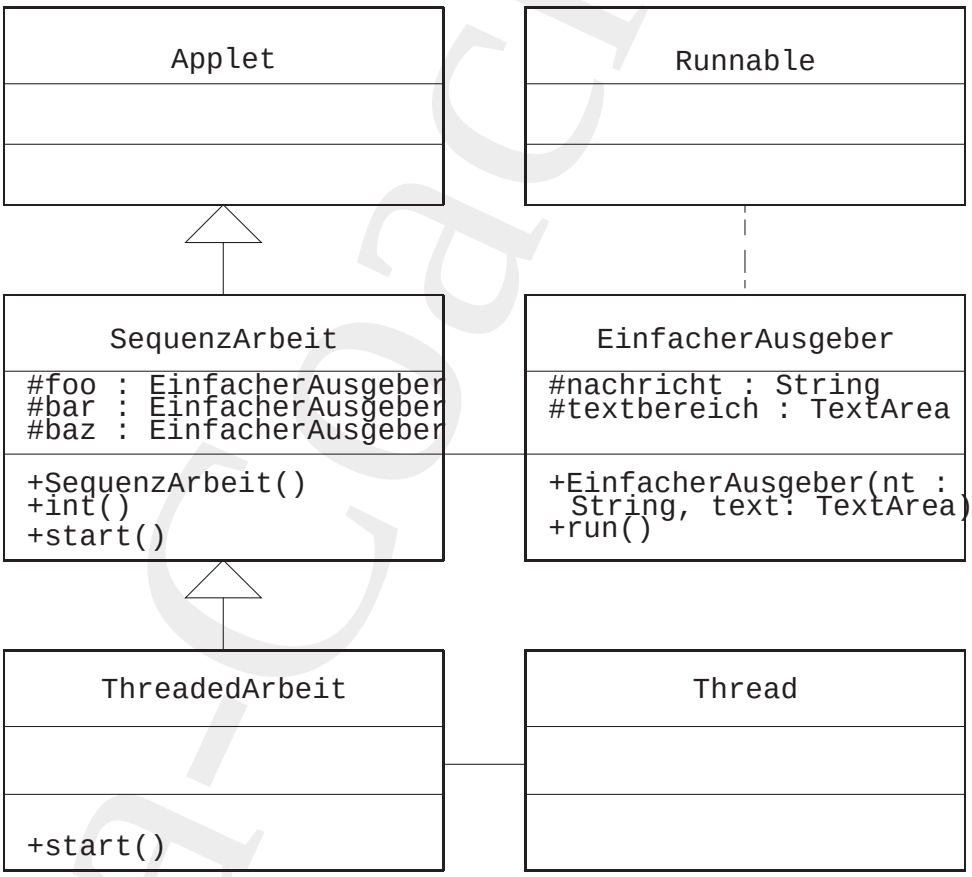


Abbildung 7.1: Klassendiagramm für das Multithreading-Beispiel „Textausgabe“

```
public class SequenzArbeit extends Applet
{
   protected TextArea text;
   protected EinfacherAusgeber foo;
   protected EinfacherAusgeber bar;
   protected EinfacherAusgeber baz;


   public SequenzArbeit()
   {
      /*
       *  Textbereich von 5 Zeile mit 20 Spalten
       */
      text = new TextArea(5, 20);

      foo = new EinfacherAusgeber(
            "SAP ist ...\n", text);
      bar = new EinfacherAusgeber(
            "SAP go go ...\n", text);
      baz = new EinfacherAusgeber(
            "Tschüß ...\n", text);
   }


   public void init()
   {
      add(text);
   }


   public void start()
   {
      foo.run();
      bar.run();
      baz.run();
   }
}
```

Klasse `EinfacherAusgeber`

```
/**
 *  Einfache Ausgabe in einem Fenster! Idee bzw. Code
 *  ähnlich: Doug Lea; Concurrent Programming in Java, ISBN
 *  0-201-63455-4
 *
 *@author     Hinrich E. G. Bonin
 *@created    14. Dezember 2002
 *@version    1.0
 *@since      13-Jan-1998
 */

import java.awt.*;

public class EinfacherAusgeber implements Runnable
{
   /*
    *  Variable für die auszugebende Nachricht
    */
   protected String nachricht;
   /*
    *  Variable für den Textbereich in den ausgegeben wird
    */
   protected TextArea textbereich;


   public EinfacherAusgeber(
         String nachricht, TextArea textbereich)
   {
      this.nachricht = nachricht;
      this.textbereich = textbereich;
   }


   public void run()
   {
      textbereich.appendText(nachricht);
   }
}
```

Teil des HTML-Dokumentes für das Applet `ThreadedArbeit`

```
<object
   codetype="application/java"
   classid="java:ThreadedArbeit.class"
   name="ThreadedArbeit"
   width="500"
   height="600"
   alt="Kleiner Scherz &uuml;ber SAP.">
   Java ThreadedArbeit.class
</object>
```

Klasse `ThreadedArbeit`

```
/**
 *  Nebenläufiges Abarbeiten in einem Applet! Idee bzw. Code
 *  ähnlich: Doug Lea; Concurrent Programming in Java ISBN
 *  0-201-63455-4
 *
 *@author     Hinrich E. G. Bonin
 *@created    26. November 2002
 *@version    1.0
 *@since      13-Jan-1998
 */

import java.applet.*;
import java.awt.*;

public class ThreadedArbeit extends
      SequenzArbeit
{
   public void start()
   {
      new Thread(foo).start();
      new Thread(bar).start();
      new Thread(baz).start();
   }
}
```

7.1.1 Unterbrechung (`sleep`)

Bei mehreren *Threads* spielt die Aufteilung der Rechenzeit ein große Rolle. Durch das direkte Setzten von Prioritäten mit der Methode `setPriority()` besteht eine Möglichkeit der Beeinflussung. Trotzdem kann es passieren, daß ein *Thread* die gesamte Kapazität in Anspruch nimmt und die anderen nicht zum Zuge kommen. Um eine solche Situation zu vermeiden, sollte jeder *Thread* mal unterbrochen werden, damit die anderen eine Chance bekommen. Dazu dient die Klassenmethode `sleep(long zeitMilliSekunden)` der Klasse `Thread`. Beim Aufruf dieser Methode muß die Fehlersituation `InterruptedException` abgefangen werden. Unbedingt eingebaut werden sollte die `sleep`-Methode bei Iterationen und besonders bei Endlosschleifen. Dazu bietet sich beispielsweise folgende Konstruktion an:

```
public class MyBesteIdee implements Runnable {
  public void run() {
    while (true) {
      // tue was
      try {
        Thread.sleep(500);
      }
      catch (InterruptedException e) {
        // Fehler behandeln
      }
    }
  }
}
```

7.1.2 Synchronisation (`wait()`, `notify()`, `synchronized`, `join`)

Das `Thread`-Konzept ist in Java^{TM} konsequent für alle Objekte verwirklicht. So kennt jedes Objekt, jede Subklasse von `Object`, folgende Steuerungsmethoden:

- `wait()` und `wait(long timeout)`
  Die Methode zum Abwarten, bis der gestartete *Thread* ein `notify()`

für dieses Objekt sendet. Dabei gibt ein Argument die maximale Wartezeit in Millisekunden an.

- `notify()` und `notifyAll()`
  Die Methode dient zum Beenden des Wartezustandes. Wenn ein oder mehrere *Threads* mit `wait()` warten, wird danach der Wartezustand beendet. Wartet kein *Thread* ist diese Methode ohne Bedeutung.

Diese Steuerung der *Threads* kann zu Blockaden („*Deadlocks*“) führen, wenn beispielsweise der *Thread* X wartet, das der *Thread* Y ein `notify()` sendet, und der *Thread* Y wartet, daß der *Thread* X ein `notify()` sendet.

Wenn ein `Thread`-Objekt die Ergebnisse, die ein anderer *Thread* X produziert, benötigt, dann muß gewartet werden, bis X fertig ist. Dazu dient die Methode `join()`. Diese Methode aus der Klasse `Thread` sorgt für das Abwarten, bis der gestartete *Thread* X fertig ist. Wenn mehrere laufende *Threads* dasselbe Objekt ändern, kommt es zu Problemen, wenn nur halb ausgeführte Zwischenzustände eines *Threads* vom anderen schon geändert werden (*concurrent update problem*). Um dies zu verhindern, sind alle gefährdeten Aktionen in einem Block zusammenzufassen. Dazu dient das `synchronized`-Konstrukt.

7.2 Ereignisbehandlung (Delegationsmodell)

Ein GUI-System (*Graphical User Interface*)[1] muß Interaktionen mit dem Benutzer steuern. Dazu nutzt es eine Ereigniskontrollschleife (*event loop*) in der festgestellt wird, ob eine betreffende Benutzeraktion eingetreten ist. Im `J2SE SDK` wird diese Ereignisbehandlung von Sun Microsystems, Inc. USA, als Delegationsmodell bezeichnet. Einem GUI-Objekt kann jetzt ein *event handler* direkt zugeordnet werden. Dieser überwacht das Eintreten eines Ereignisses. Er „horcht“ permanent und appliziert eine fest vorgegebene Methode, wenn das Ereignis eintritt. Er wird daher als **Listener** bezeichnet. Erfolgt beispielsweise ein Mausclick auf einen Button, dann führt sein zugeordneter `ActionListener` die Methode `actionPerformed()` aus. Der Listener selbst ist ein Interface und spezialisiert die Klasse `EventListener`. Die Abbildung 7.2 S. 169 skizziert als Klassendiagramm dieses Delegationsmodell.

Listener

[1] Andere Beispiele sind Microsoft’s Windows und Unix’s Motif.

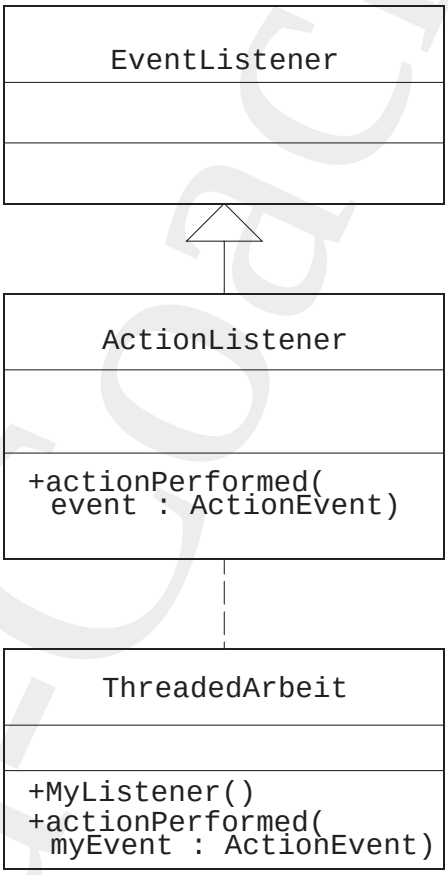


Abbildung 7.2: Java^{TM} AWT: Konstruktion des Delegationsmodells

```
public interface ActionListener extends EventListener {
  public abstract void actionPerformed(ActionEvent event}
}
```

Das Delegationsmodell für einen aktionskontrollierten Auslöseknopf (Button) fußt dann beispielsweise auf folgender Konstruktion. Der eigene Listener `MyListener` implementiert das Interface `ActionListener` wie folgt:

```
public class MyListener implements ActionListener {
  public MyListener(...){
  // Konstruktormethode
  // Die drei Punkte stehen fuer das Objekt welches beim
  // Ereigniseintritt modifiziert werden soll. So wird
  // es fuer den Listener erreichbar.
  }
  public void actionPerformed(ActionEvent myEvent) {
    // irgend eine Aktion wie modifiziere ...
    // und/oder zum Beispiel
    System.out.println{Ereignis eingetreten};
  }
}
```

Einer Instanz von `MyButton` wird dann eine Instanz von `MyListener` mit der Methode `addActionLister()` wie folgt zugeordnet:

```
public class MyButton extends Button {
  Button anmelden;
  public MyButton()
    // irgendeinen Button definieren, zum Beipiel
    anmelden = new Button("Anmelden");
  }
}
public class MyApplication extends Frame {
  public MyApplication() {
    MyListener mL = new MyListener(...);
    MyButton   mB = new MyButton();
    mB.addActionListener(mL);
    ...
  }
  public static void main(String argv[]) {
    new MyApplication().show();
  }
}
```

Die Zuordnung einer Instanz von `MyListener` zu einer Instanz von `MyButton` kann auch im Konstruktor `MyButton()` geschehen und zwar wie folgt:

```
public MyButton extends Button implements ActionListener {
  Button anmelden;
  public MyButton() {
    anmelden = new Button("Anmelden");
    this.addActionListener(this);
  }
  public void actionPerformed(ActionEvent myEvent) {
    System.out.println{Ereignis eingetreten};
  }
}
```

7.2.1 `ActionListener` — Beispiel `SetFarbe`

Mit einem Beispiel wird im folgenden das Delegationsmodell verdeutlicht. Es wird angenommen, daß ein Applet-Benutzer die Farbe eines Ausgabetextes direkt verändern möchte. Dabei werden zwei Lösungen angeboten: Im ersten Fall geschieht die Farbwahl durch Eintragung des gewünschten Farbnamens in ein Textfeld (↪ Seite 174). Im zweiten Fall kann durch einen Doppelklick auf eine Liste mit Farbnamen die gewünschte Farbe ausgewählt werden (↪ Seite 175). Für beide Fälle wird nur eine Klasse `SetFarbe`, die das Interface `ActionListener` implementiert, benötigt (↪ Seite 171), denn die zu kontrollierende Benutzeraktion besteht jeweils aus dem Setzen einer Farbe. In der Methode `actionPerformed()` wird der Farbname als ein `String` über die Methode `getActionCommand()` gewonnen. In beiden Fällen liefert diese Methode einen `String`, unabhängig davon, ob die Benutzeraktion durch einen Doppelklick oder über die Entertaste nach Eingabe des Textes erfolgte. Die Abbildung 7.3 S. 172 zeigt das Klassendiagramm für den Fall der textlichen Eingabe der gewünschten Farbe.

Klasse `SetFarbe`

```
/**
*  Kleines Beispiel fuer die ActionListener Idee aus Glenn
```

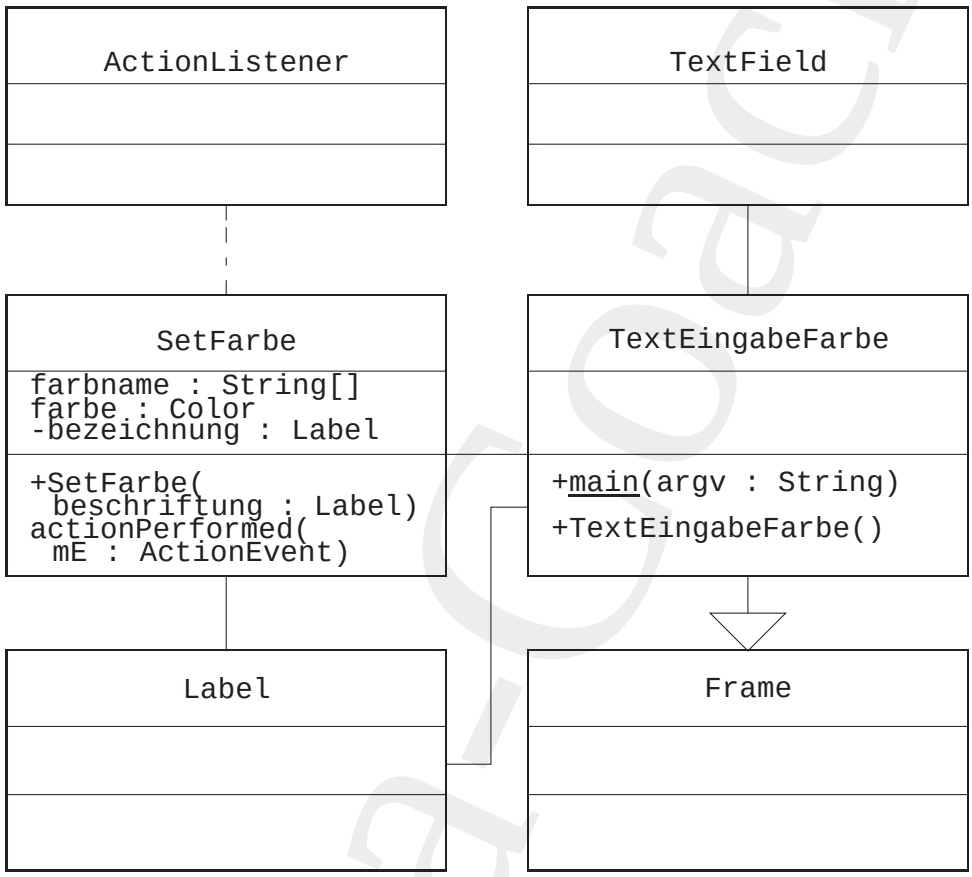


Abbildung 7.3: Klassendiagramm für `TextEingabeFarbe`

```
 *  Vanderburg, et al.; MAXIMUM JAVA 1.1, pp.230--234
 *  Quellcode modifiziert. Teil I: Listener SetFarbe
 *
 *@since      27-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.farbe;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.*;

public class SetFarbe implements ActionListener
{
   final String farbname[]
          = {"rot", "gruen", "blau"};
   final Color farbe[]
          = {Color.red, Color.green, Color.blue};
   private Label bezeichnung;


   public SetFarbe(Label beschriftung)
   {
      bezeichnung = beschriftung;
   }


   public void actionPerformed(ActionEvent myE)
   {
      String name = myE.getActionCommand();
      for (int n = 0; n < farbname.length; n++)
      {
         if (farbname[n].equals(name))
         {
            bezeichnung.setForeground(farbe[n]);
            return;
         }
      }
      System.out.println(
            "Unbekannter Farbwunsch: " + name);
```

```
   }
}
```

Klasse `TextEingabeFarbe`

```
/**
 *  Kleines Beispiel fuer ActionListener Teil IIa: Farbwahl
 *  per Texteingabe
 *
 *@since      27-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.farbe;

import java.awt.*;

public class TextEingabeFarbe extends Frame
{
   public static void main(String argv[])
   {
      new TextEingabeFarbe().show();
   }


   public TextEingabeFarbe()
   {
      Label myL = new Label("Hello World");
      TextField text = new TextField(20);

      this.add("North", text);
      this.add("Center", myL);
      this.pack();

      SetFarbe sf = new SetFarbe(myL);
      /*
       *  Aktion ist Enter-Taste druecken
       */
```

Abbildung 7.4: Ergebnis: `java de.fhnon.farbe.TextEingabeFarbe`

```
      text.addActionListener(sf);
   }
}
```

Klasse `ListWahlFarbe`

```
/**
 *  Kleines Beispiel fuer ActionListener Teil IIb: Farbliste
 *
 *@since      127-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.farbe;

import java.awt.*;

public class ListWahlFarbe extends Frame
{
   public static void main(String argv[])
   {
      new ListWahlFarbe().show();
   }
```

Abbildung 7.5: Ergebnis: `java de.fhnon.farbe.ListWahlFarbe`

```
   public ListWahlFarbe()
   {
      Label myL = new Label("Hello World");
      List myFarbList = new List(3);
      myFarbList.add("rot");
      myFarbList.add("gruen");
      myFarbList.add("blau");

      this.add("West", myFarbList);
      this.add("Center", myL);
      this.pack();

      SetFarbe sf = new SetFarbe(myL);
      /*
       *  Aktion besteht im Doppelklick
       *  auf List-Eintragung
       */
      myFarbList.addActionListener(sf);
   }
}
```

7.2.2 Event→Listener→Method

Für verschiedene Ereignisse gibt es unterschiedliche *Listener* mit fest vorgegebenen Methoden (↪ Tabelle 7.1 S. 178). Das Beispiel:

`ZeigeTastenWert` nutzt den `KeyListener` (↪ Seite 177).

Verständlicherweise kann nicht jedes Objekt jedem *Listener* zugeordnet werden; beispielsweise setzt der `WindowListener` ein Objekt der Klasse `Frame` voraus. Tabelle 7.2 S. 179 zeigt die Zuordnung. Die Auslösung des Ereignisses ist ebenfalls abhängig vom jeweiligen Objekt. Zum Beispiel ist es ein Mausklick beim `Button` und ein Mausdoppelclick beim `List`-Objekt. Tabelle 7.3 S. 179 zeigt die jeweiligen Aktionen. Darüber hinaus ist bedeutsam, welcher Wert zur Identifizierung des jeweiligen Objekts von `getActionCommand()` zurückgegeben wird. Tabelle 7.4 S. 179 nennt die Werte.

7.2.3 `KeyListener` — Beispiel `ZeigeTastenWert`

Die Abbildung 7.6 S. 180 zeigt das Klassendiagramm.

Klasse `ZeigeTastenWert`

```
/**
 *  Kleines Beispiel fuer die Tasten-Rueckgabewerte der
 *  Tastatur Idee aus Glenn Vanderburg, et al.; MAXIMUM JAVA
 *  1.1, pp.239 Quellcode modifiziert.
 *
 *@author     Hinrich Bonin
 *@since      27-Apr-1998
 *@version    1.1
 */
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import java.awt.*;

public class ZeigeTastenWert extends Frame
{
   public static void main(String args[])
   {
      new ZeigeTastenWert().show();
   }


   public ZeigeTastenWert()
```

event	*listener*	*method*
`Action-Event`	`ActionListener`	`actionPerformed()`
`Adjustment-Event`	`AdjustmentListener`	`adjustmentValueChanged()`

`Component-Event`	`ComponentListener`	`componentResized()`
		`componentMoved()`
		`componentShown()`
		`componentHidden()`

`Focus-Event`	`FocusListener`	`focusGained()` `focusLost()`
`Item-Event`	`ItemListener`	`itemStateChanged()`

`Key-Event`	`KeyListener`	`keyTyped()`
		`keyPressed()`
		`keyReleased()`

`Mouse-Event`	`MouseListener`
	
	`MouseMotionListener`
		`mouseClicked()`
		`mouseEntered()`
		`mouseExited()`
		`mousePressed()`
		`mouseReleased()`
		`mouseDragged()`
		`mouseMoved()`

`Window-Event`	`WindowListener`	`windowClosed()`
		`windowClosing()`
		`windowDeiconified()`
		`windowIconified()`
		`windowOpened()`

Legende:
Aus Effizienzgründe gibt es zwei *Listener* für ein `MouseEvent`.

Tabelle 7.1: Listener-Typen: event→listener→method

GUI *object*	*listener*
`Button`	`ActionListener`
`Choice`	`ItemListener`
`Checkbox`	`ItemListener`

`Component`	`ComponentListener`
	`FocusListener`
	`KeyListener`
	`MouseListener`
	`MouseMotionListener`

`Dialog`	`WindowListener`
`Frame`	`WindowListener`
`List`	`ActionListener` `ItemListener`
`MenuItem`	`ActionListener`
`Scrollbar`	`AdjustmentListener`
`TextField`	`ActionListener`

Legende:
listener ↪ Tabelle 7.1 S. 178

Tabelle 7.2: Object→listener

GUI *object*	Benutzeraktion
`Button`	Click auf den `Button`
`List`	Doppelclick auf das gewählte `Item`
`MenuItem`	Loslassen auf dem `MenuItem`
`TextField`	Auslösen der Entertaste

Tabelle 7.3: Benutzeraktion auf das GUI *object*

GUI *object*	Rückgabewert von `getActionCommand()`
`Button`	Text von `Button`
`List`	Text vom gewählten `Item`
`MenuItem`	Text vom gewählten `MenuItem`
`TextField`	Gesamte Text des Feldes

Tabelle 7.4: Rückgabewert von `getActionCommand()`

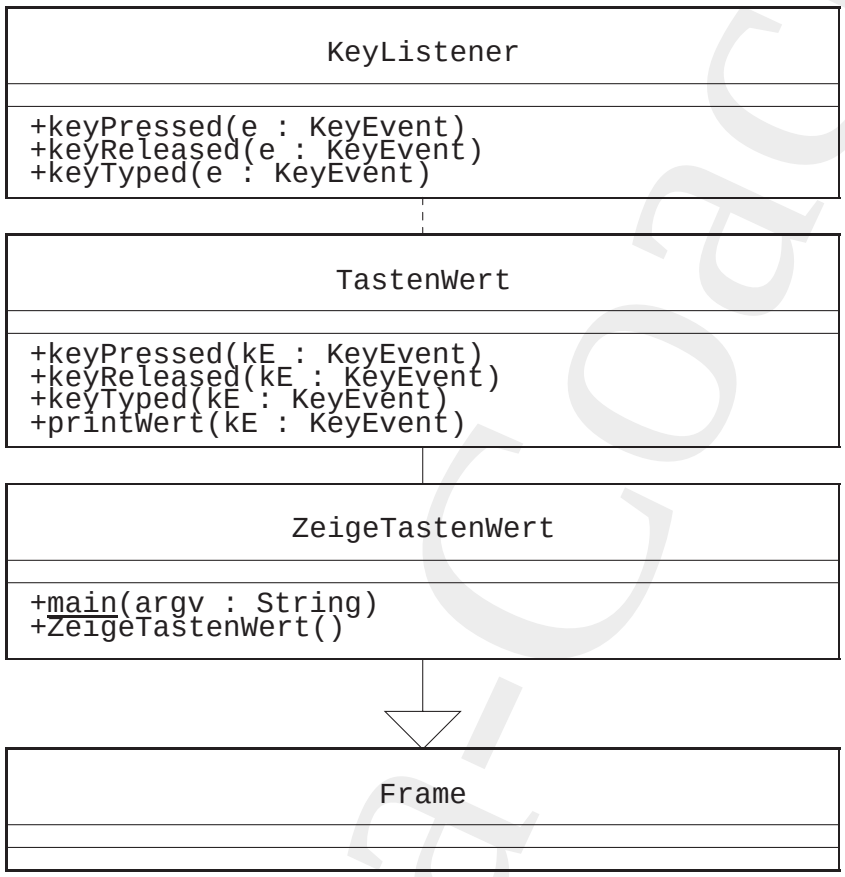
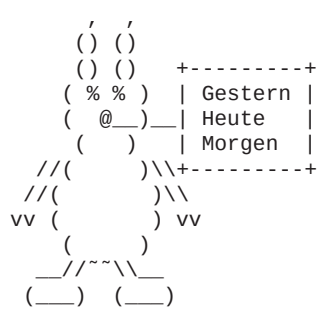


Abbildung 7.6: Klassendiagramm für `ZeigeTastenWert`

```
   {
      Label l = new Label("Hello World");
      this.add(l);
      this.pack();

      TastenWert tW = new TastenWert();
      l.addKeyListener(tW);
   }
}
class TastenWert implements KeyListener
{


   public void keyPressed(KeyEvent kE)
   {
      if (!Character.isLetter(kE.getKeyChar()))
      {
         Toolkit.getDefaultToolkit().beep();
         kE.consume();
      }
      printWert(kE);
   }


   public void keyReleased(KeyEvent kE)
   {
      printWert(kE);
   }


   public void keyTyped(KeyEvent kE)
   {
      printWert(kE);
   }


   public void printWert(KeyEvent kE)
   {
      System.out.println(kE.toString());
   }
}
```

7.3 Persistente Objekte



Der Begriff **Persistenz** bedeutet üblicherweise (besonders in der Biologie und der Medizin) das Beibehaltung eines Zustandes über einen längeren Zeitraum. Im Zusammenhang mit der Objekt-Orientierung beschreibt Persistenz die Existenz eines Objektes **unabhängig vom Ort und der Lebensdauer seines erzeugenden Programmes**. Ein persistentes Objekt kann in eine Datei geschrieben werden und später benutzt oder an einen anderen Rechner übertragen werden.

Um Objekt-Persistenz zu erreichen, sind folgende Schritte erforderlich:

Serialization

1. Konvertierung der Repräsentation des Objektes im Arbeitsspeicher (≡ memory layout) in einen sequentiellen Bytestrom, der geeignet ist für eine Speicherung in einer Datei oder für eine Netzübertragung.

Casting

2. (Re-)Konstruktion eines Objektes aus dem sequentiellen Bytestrom in der ursprünglichen Form mit dem „identischen Verhalten“.

Da die Persistenz über einen Bytestrom erreicht wird, bleibt die Objektidentität selbst nicht erhalten. Persistent ist nur das Objektverhalten, da alle Methoden und Variablen mit ihren Werten aus dem Bytestrom und dem Lesen der jeweiligen Klassen wieder rekonstruiert werden.[2]

Bei der Serialization werden nur die Variablen mit ihren Werten und die Klassendeklaration codiert und in den Bytestrom geschrieben. Der *Java Virtual Maschine Byte Code*, der die Methoden des Objektes abbildet, wird dabei nicht gespeichert. Wenn ein Objekt rekonstruiert wird,

[2] Damit ist Objekt-Persistenz zu unterscheiden von einem einfachen Speichern einer Zeichenkette (string) wie es beispielsweise durch die Methoden `save()` und `load` der Klasse `java.util.Properties` erfolgt. Dort wird nur der Inhalt der Zeichenkette gespeichert und die zugehörenden Methoden der Klasse `String` werden nicht berücksichtigt.

dann wird die Klassendeklaration gelesen und der normale Klassenladungsmechanismus, das heißt, Suchen entlang dem `CLASSPATH`, wird ausgeführt. Auf diese Art wird der *Java Byte Code* der Methoden verfügbar. Wird die Klasse nicht gefunden, kommt es zu einer Ausnahme, genauer formuliert:

```
readObject() throws ClassNotFoundException
```

!Agent

Diese Persistenz ist daher nicht hinreichend für Objekte, die sich wie Agenten völlig frei im Netz bewegen sollen.

Ein besonderes Problem der *Serialization* liegt in der Behandlung der Referenzen auf andere Objekte. Von allen Objekten die vom persistenten Objekt referenziert werden, muß eine „Objektkopie“ mit in den Bytestrom gespeichert werden. Referenziert ein referenziertes Objekt wieder ein anderes Objekt, dann muß auch dessen Kopie in den Bytestrom kommen und so weiter. Der Bytestrom wird daher häufig sehr viele Bytes umfassen, obwohl das zu serialisierende Objekt selbst recht klein ist. Die Referenzen können ein Objekt mehrfach betreffen oder auch zirkulär sein. Um dieses Problem zu lösen wird nur jeweils einmal der „Inhalt eines Objektes“ gespeichert und die Referenzen dann extra. (Näheres dazu beispielsweise ↪ [Vanderburg97] pp. 554–559)

Im folgenden werden einige Konstrukte, die ein Objekt persistent machen, anhand eines Beispiels dargestellt. Als Beispielobjekt dient ein Button, der beim Drücken eine Nachricht auf die Java-Console schreibt. Dieser Button wird in der Datei `PersButton.java` beschrieben (↪ Seite 186). Seine Klasse `PersButton` ist eine Unterklasse von `Button` und implementiert das Interface `ActionListener` damit über die Methode:

```
actionPerformed()
```

das Drücken als Ereignis die Systemausgabe veranlaßt. Weil eine mit dem Konstruktor `PersButton()` erzeugte Instanz serialisiert werden soll, implementiert die Klasse `PersButton` auch das Interface `Serializable`. Ohne eine weitere Angabe als `implements Serializable` wird das *default Java runtime serialization format* benutzt.

Mit der Klasse `PersButtonProg` in der Datei `PersButtonProg.java` wird in deren Methode `main()` der Beispielbutton `foo` erzeugt (↪ Seite 187). Das Schreiben in die Datei `PButton.ser` er-

folgt in einem `try-catch`-Konstrukt um Fehler beim Plattenzugriff abzufangen. Die eigentliche Serialization und das Schreiben von `foo` erfolgt durch:

```
out.writeObject(foo)
```

Dabei ist `out` eine Instanz der Klasse `ObjectOutputStream`. Bei der Erzeugung von `out` wird dem Konstruktor eine Instanz der Klasse `FileOutputStream` übergeben. Diese übergebene Instanz selbst wird erzeugt mit ihrem Konstruktor, dem der Dateiname als Zeichenkette übergeben wird. Durch die Klasse `java.util.zip.GZIPOutputStream` wird dafür gesorgt, daß die Daten komprimiert werden, bevor sie in die Datei geschrieben werden. Die Serialization fußt hier auf der folgenden Konstruktion:

→**file**

```
FileOutputStream  fout
    = new FileOutputStream("PButton.ser");
GZIPOutputStream gzout
    = new GZIPOutputStream(fout);
ObjectOutputStream out
    = new ObjectOutputStream(gzout);
out.writeObject(foo);
out.close();
```

In der Datei `UseButton.java` steht die Klasse `UseButton` mit ihrer Methode `doUserInterface()` (↪ Seite 188). In dieser Methode wird der Bytestrom gelesen und unser Buttonobjekt wieder rekonstruiert. Dazu dient die Methode `readObjekct()`. Diese erzeugt eine Instanz der Klasse `Object` und nicht automatisch eine Instanz der Klasse `PersButton`. Es ist daher ein *Casting* erforderlich, das heißt, es bedarf einer Konvertierung von `Object`→`PersButton`. Die Rekonstruktion fußt hier auf der folgenden Konstruktion:

←**file**

```
FileInputStream  fin
    = new FileInputStream("PButton.ser");
GZIPInputStream gzin
    = new GZIPInputStream(fin);
```

```
ObjectInputStream in
    = new ObjectInputStream(gzin);
PersButton bar
    = (PersButton) in.readObject();
in.close();
```

Der ursprüngliche Name des Beispielsbuttons `foo` geht verloren. Der rekonstruierte Button aus dem Bytestrom der Datei `PButton.ser` wird jetzt mit `bar` referenziert.

Um mehr Kontrolle über die *Serialization* zu gewinnen, gibt es das Interface `Externalizable`, eine Spezialisierung des Interfaces `Serializable`. Dabei kann man dann beispielsweise selbst entscheiden, welche Superklassen mit in den Bytestrom kommen.

Beim Speichern unseres Beispielbuttons `foo` wird eine `serialVersionUID` der Klasse `PersButton` mit in den Bytestrom geschrieben. Der Wert von `serialVersionUID` wird aus einem Hashcode über die Variablen, Methoden und Interfaces der Klasse berechnet. Beim Rekonstruieren des Beispielbuttons in `UseButton` wird aus der (nun über den `CLASSPATH`) geladenen Klasse `PersButton` wieder mittels Hashcode der Wert berechnet. Gibt es eine Abweichung, dann stimmt die Version der Klasse zum Zeitpunkt der Rekonstruktion nicht mit der Version der Klasse zum Zeitpunkt des Schreibens in den Bytestrom überein. Um „alte Objekte“ trotz einer Veränderung ihrer Klasse noch verfügbar zu machen, gibt es eine Unterstützung des Versionsmanagements, das heißt, es kann in die Versionskontrolle eingegriffen werden (Stichwort: Investitionsschutz für alte, nützliche Objekte).

Nicht jedes Objekt ist *serializable*, zum Beispiel wäre eine Instanz `baz` der Klasse `java.lang.Thread` so nicht behandelbar. Um die Serialization zu verhindern, ist bei der Deklaration das Kennwort `transient` anzugeben.

transient

```
transient Thread baz;
```

Um nach der Rekonstruktion wieder über das mit `transient` gekennzeichnete Objekt zu verfügen, kann in die zu serialisierende Klasse eine Methode `public void readObject()` aufgenommen werden. Wird die serialisierte Klasse rekonstruiert, dann sucht Java^{TM} nach dieser Methode und wendet sie für das Rekonstruieren der Klasse an (↪

Abbildung 7.7 S. 188).

7.3.1 Serialization — Beispiel PersButton

Klasse PersButton

```
/**
 *  Kleines Beispiel fuer die ,,Serializing a button'',
 *  Grundidee ,,Button'' aus: Glenn Vanderburg, et al.;
 *  MAXIMUM JAVA 1.1, pp.543 jedoch eigene
 *  Quellcodestruktur. Teil I: Button-Beschreibng
 *
 *@author     Hinrich Bonin
 *@since      30-Apr-1998
 *@version    1.0
 */
import java.io.*;
import java.awt.event.*;
import java.awt.*;

public class PersButton extends Button
       implements ActionListener, Serializable
{

   Button myButton;


   public PersButton()
   {
      myButton =  new Button("Anmelden");
      System.out.println("Button erzeugt");
      this.myButton.addActionListener(this);
   }


   public void actionPerformed(ActionEvent e)
   {
      System.out.println("Button gedrueckt");
   }
}
```

Klasse `PersButtonProg`

```
/**
 *  Kleines Beispiel fuer die ,,Serializing a
 *  button'' Teil II: Schreiben eines Button in
 *  PButton.dat
 *
 *@author     Hinrich Bonin
 *@since      28-Apr-1998
 *@version    1.1
 */
import java.io.*;
import java.util.zip.*;

public class PersButtonProg
{

   public static void main(String args[])
   {
      PersButton foo = new PersButton();
      try
      {
         FileOutputStream fout
               = new FileOutputStream("PButton.ser");
         GZIPOutputStream gzout
               = new GZIPOutputStream(fout);
         ObjectOutputStream out
               = new ObjectOutputStream(gzout);
         out.writeObject(foo);
         out.close();
         System.exit(0);
      } catch (Exception e)
      {
         e.printStackTrace(System.out);
      }
   }
}
```

7.3.2 Rekonstruktion — Beispiel `UseButton`

Klasse `UseButton`

```
/**
 *  Kleines Beispiel fuer die ,,Use a persistent button
 *  object''
 *
 *@author     Hinrich Bonin
 *@since      29-Apr-1998
 *@version    1.0
 */
import java.io.*;
import java.awt.event.*;
import java.awt.*;
import java.util.zip.*;

public class UseButton extends Frame
{

   public static void main(String args[])
   {
      Frame myFrame = new Frame("Willi will ...?");
      Panel myP = new Panel();
      UseButton myUseButton = new UseButton();
      myUseButton.doUserInterface(myFrame, myP);
      myFrame.pack();
      myFrame.show();
   }


   public void doUserInterface(Frame frame, Panel panel)
   {
      try
      {
         FileInputStream fin
                = new FileInputStream("PButton.ser");
         GZIPInputStream gzin
                = new GZIPInputStream(fin);
         ObjectInputStream in
                = new ObjectInputStream(gzin);

         PersButton bar
```

```
                = (PersButton) in.readObject();

            in.close();
            frame.setLayout(new BorderLayout());
            panel.add(bar.myButton);
            frame.add("Center", panel);
         } catch (Exception e)
         {
            e.printStackTrace(System.out);
         }
      }
}
```

Protokolldatei `PersButton.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac PersButton.java PersButtonProg.java

C:\bonin\anwd\code>javac UseButton.java

C:\bonin\anwd\code>java PersButtonProg
Button erzeugt

C:\bonin\anwd\code>java UseButton
java UseButton
Button gedrueckt
Button gedrueckt
```

7.3.3 JAR (*Java Archiv*)

Da ein serialisiertes Objekt zu seiner Rekonstruktion seine Klasse benötigt, bietet es sich an beide in einem gemeinsamen Archiv zu verwalten. Dazu gibt es im J2SE das Werkzeug JAR, das *Java Archiv*. JAR . Üblicherweise werden in einer solchen Archivdatei mit dem Suffix `.jar` folgende Dateitypen zusammengefaßt:

Legende:
Beispiel *Serialization & Rekonstruktion*
Quellecode `PersButton` ↪ S. 186; `PersButtonProg` ↪ S. 187
und `UseButton` ↪ S. 188.

Abbildung 7.7: Beispiel `UseButton`

- `<filename>.ser`
  Dateien der serialisierten Objekte
- `<filename>.class`
  Dateien der dazugehörenden Klassen
- Sound- und Image-Dateien

Die `jar`-Kommandos werden mit dem Aufruf ohne Kommandos angezeigt:

```
cl3:/u/bonin/myjava:>jar
Usage: jar {ctx}[vfm0M] [jar-file] [manifest-file] files ...
Options:
  -c  create new archive
  -t  list table of contents for archive
  -x  extract named (or all) files from archive
  -v  generate verbose output on standard error
  -f  specify archive file name
  -m  include manifest information from specified manifest file
  -0  store only; use no ZIP compression
  -M  Do not create a manifest file for the entries

If any file is a directory then it is processed recursively.
Example: to archive two class files into an archive called classes.jar:
  jar cvf classes.jar Foo.class Bar.class
Note: use the '0' option to create a jar file that
can be put in your CLASSPATH
cl3:/u/bonin/myjava:>
```

In unserem Beispiel bietet sich folgendes `jar`-Kommando an:

```
>jar -cf Button.jar PButton.ser PersButton.class UseButton.class
```

Dabei bedeuten die Parameter `cf`, daß ein neues Archiv mit dem Dateinamen des ersten Argumentes erzeugt wird.

Das *Java Archiv* ist beispielsweise nützlich für Applets, da so mehrere Dateien mit einem HTTP-Request übertragen werden können. Das *Java Archiv* bildet die Grundlage für *JavaBeans*^{TM} (↪ Abschnitt 7.12.1 S. 301) und *Java EnterpriseBeans*^{TM} (↪ Abschnitt 7.12.2 S. 307).

7.4 Geschachtelte Klassen (*Inner Classes*)

In Java^{TM} können Klassen innerhalb von Klassen definiert werden. Um diesen Mechanismus der sogenannten *Inner Classes* zu erläutern, werden einige Konstruktionsalternativen anhand eines sehr einfachen Beispiel gezeigt. Dieses Beispiel gibt den Witztext[3] `Piep, piep ... lieb!` als String auf der Systemconsole aus.

Witztext als lokale Variable Zunächst wird eine einfache Klassendefinition `WitzA` (↪ Seite 191) mit einer lokalen Variable betrachtet. In `WitzA` ist innerhalb der Klassenmethode `main()` die lokale Variable `text` initialisiert und anschließend wird sie ausgegeben.

Beispiel `WitzA` Die Abbildung 7.8 S. 192 zeigt das Klassendiagramm.

Klasse `WitzA`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als lokale Variable
 *
 *@author     Bonin
 *@created    26. November 2002
 *@version    1.1
```

[3] Schlagerkurztext von Guildo Horn, Mai 1998

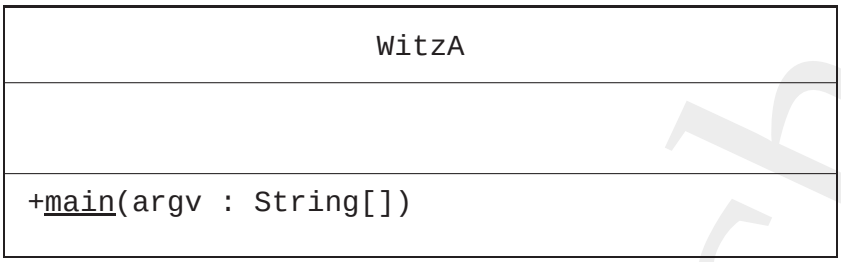


Abbildung 7.8: Klassendiagramm für `WitzA`

```
 */
public class WitzA
{
   public static void main(String argv[])
   {
      final String text = "Piep, piep ... lieb!";
      System.out.println(text);
   }
}
```

Witztext als Instanzvariable In `WitzB` (↪ Seite 192) wird statt einer lokalen Variablen eine Instanzvariable `text` definiert. Um diese Instanzvariable ausgeben zu können, ist vorher eine Instanz dieser Klasse zu erzeugen. Dazu wird der Konstruktur der Klasse, also `WitzB()`, angewendet.

Beispiel `WitzB` Die Abbildung 7.9 S. 193 zeigt das Klassendiagramm.

Klasse `WitzB`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als Instanzvariable
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
```

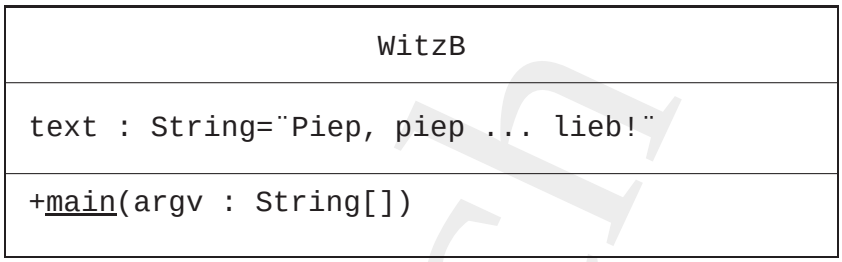


Abbildung 7.9: Klassendiagramm für `WitzB`

```
 *@version    1.1
 */
public class WitzB
{
   final String text = "Piep, piep ... lieb!";


   public static void main(String[] argv)
   {
      System.out.println((new WitzB()).text);
   }
}
```

Witztext als Instanzvariablen einer anderen Klasse In `WitzC` (↪ Seite 193) wird die Instanzvariable `text` in einer eigenen Klasse `WitzText` definiert. Erzeugt wird sie daher mit dem Konstruktor `WitzText()`. Beide Klassen `WitzC` und `WitzText` stehen in derselben Datei[4] `WitzC`. Sie sind im gemeinsamen Paket.

Beispiel `WitzC` Die Abbildung 7.10 S. 194 zeigt das Klassendiagramm.

Klasse `WitzC`

[4] Sollte die Klasse `WitzText` allgemein zugreifbar sein, also mit `public` definiert werden, dann muß sie in einer eigenen Datei mit dem Namen `WitzText.java` stehen.

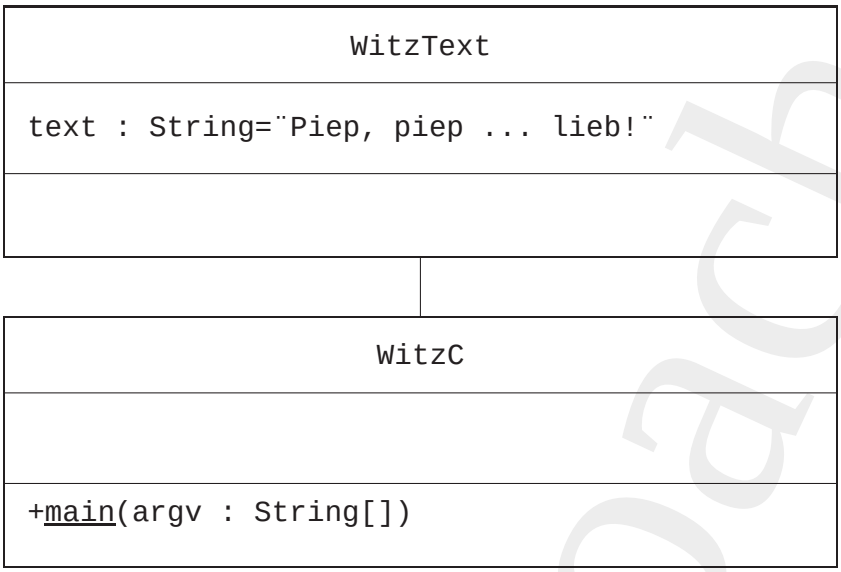


Abbildung 7.10: Klassendiagramm für `WitzC`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als Instanzvariable einer
 *  anderen Klasse
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */
public class WitzC
{
   public static void main(String[] argv)
   {
      System.out.println((new WitzText()).text);
   }
}

class WitzText
{
```

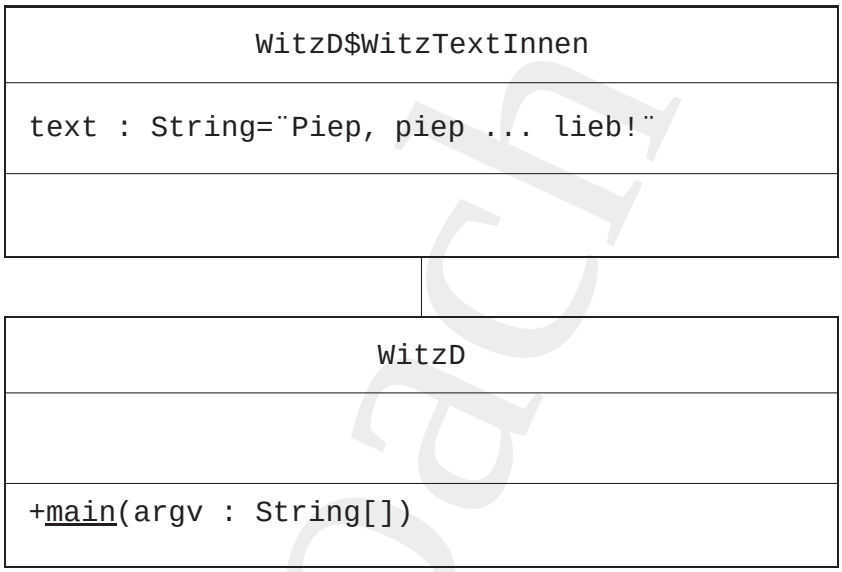


Abbildung 7.11: Klassendiagramm für `WitzD`

```
   final String text = "Piep, piep ... lieb!";
}
```

Witztext als Member Class Wird nun die Klasse `WitzText` nicht außerhalb der Klasse `WitzC` definiert, sondern innerhalb der Definition der Klasse `WitzC`, dann ändert sich auch die Art und Weise des Zugriffs auf die Instanzvariable `text`. Das folgende Beispiel `WitzD` (↪ Seite 195) zeigt diesen Mechanismus der *Inner classes*. Zur Hervorhebung dieser Schachtelung wird die „innere Klasse “ in `WitzTextInnen` umgetauft.

Beispiel `WitzD` Die Abbildung 7.11 S. 195 zeigt das Klassendiagramm.

Klasse `WitzD`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als Member Class
 *
 *@author     Bonin
 *@created    26. November 2002
 *@version    1.1
 */
public class WitzD
{
   public static void main(String[] argv)
   {
      WitzD.WitzTextInnen myText
             = (new WitzD()).new WitzTextInnen();
      System.out.println(myText.text);
   }


   class WitzTextInnen
   {
      final String text = "Piep, piep ... lieb!";
   }
}
```

Witztext als Instanzvariable einer Superklasse Für die Objekt-Orientierung ist die Vererbung ein charakteristisches Merkmal. Daher kann die Instanzvariable `text` der Klasse `WitzText` auch darüber zugänglich gemacht werden (↪ Seite 196). Der Konstruktor der Subklasse `WitzE()` erzeugt eine Instanz, die auch die Variable `text` enthält.

Beispiel `WitzE` Die Abbildung 7.12 S. 197 zeigt das Klassendiagramm.

Klasse `WitzE`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als Instanzvariable einer
 *  Superklasse
 *
```

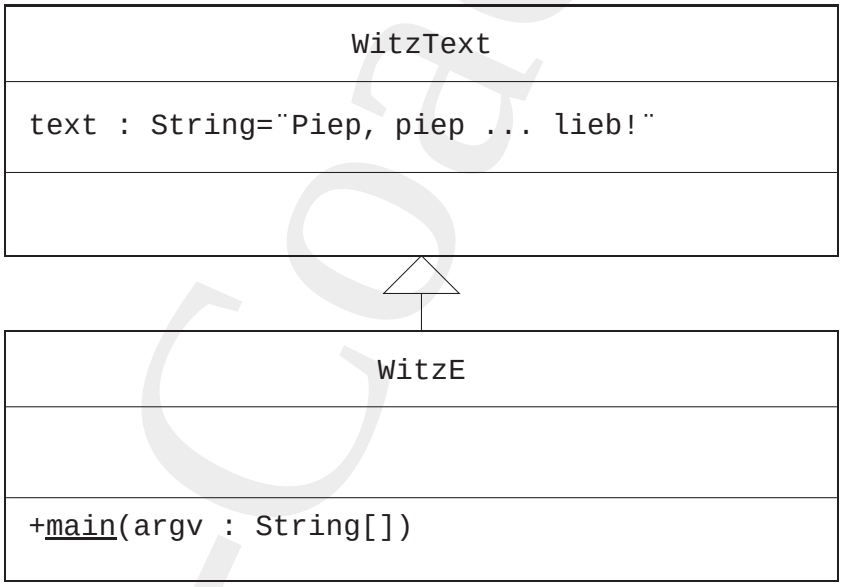


Abbildung 7.12: Klassendiagramm für `WitzE`

```
 *@author     Bonin
 *@created    26. November 2002
 *@version    1.1
 */
public class WitzE extends WitzText
{
   public static void main(String[] argv)
   {

      System.out.println((new WitzE()).text);
   }
}

class WitzText
{


   final String text = "Piep, piep ... lieb!";
}
```

Witztext als lokale Klasse Eine lokale Klasse wird in einem Block oder einer Methode deklariert. Sie ist daher ähnlich wie eine *Member Class* zu betrachten. Innerhalb der Methode (oder Block), welche die lokale Klasse deklariert, kann direkt mit ihren Konstruktor eine Instanz erzeugt werden (⟶ Seite 198). Außerhalb des Blockes oder der Methode ist keine Instanz erzeugbar.

Beispiel `WitzF` Die Abbildung 7.13 S. 199 zeigt das Klassendiagramm.

Klasse `WitzF`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als lokale Klasse
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
```

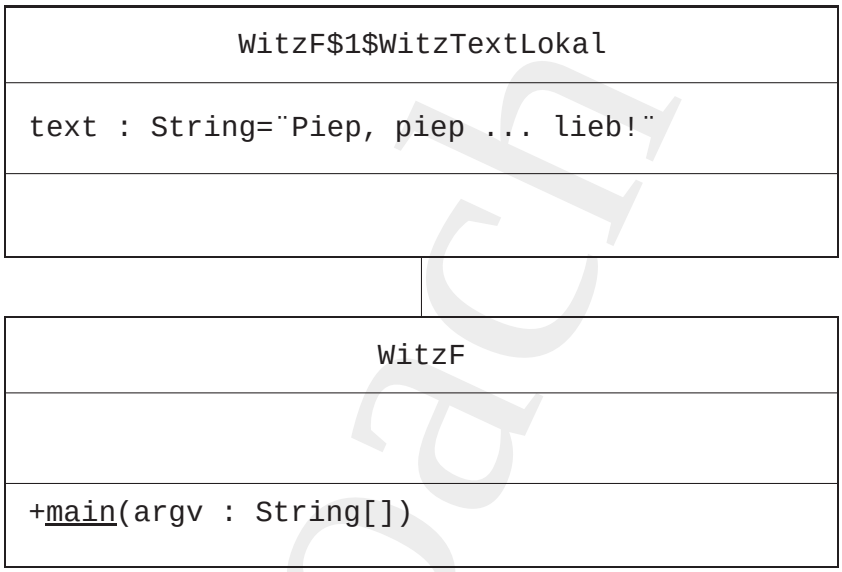


Abbildung 7.13: Klassendiagramm für `WitzF`

```
 */
public class WitzF
{
   public static void main(String[] argv)
   {
      class WitzTextLokal
      {
         final String text
               = "Piep, piep ... lieb!";
      }

      System.out.println((new WitzTextLokal()).text);
   }
}
```

Witztext als anonyme Klasse Wenn der Name einer lokalen Klasse unbedeutend ist und eher die Durchschaubarkeit verschlechtert als erhöht, dann kann auch eine anonyme Klasse konstruiert werden (↪

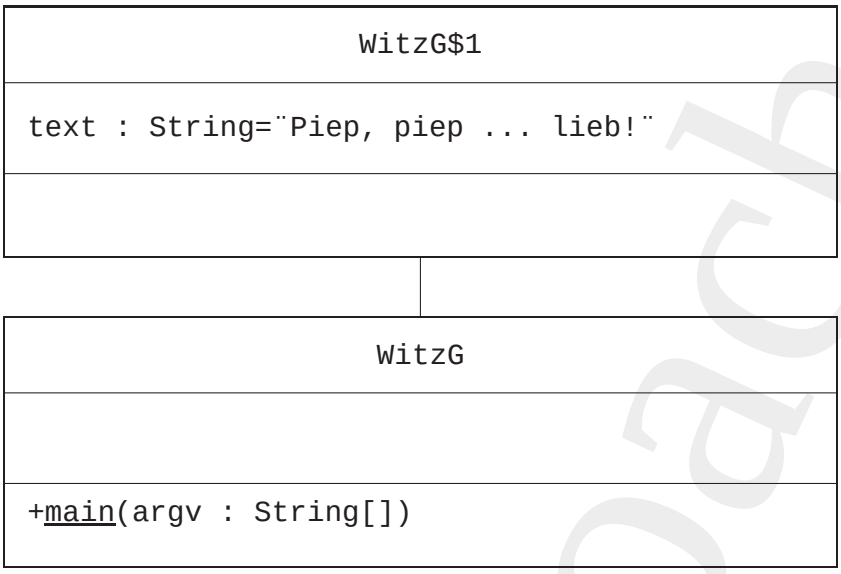


Abbildung 7.14: Klassendiagramm für `WitzG`

Seite 200). Mit der Ausführung von:

```
WitzG foo = new WitzG(){ .. }
```

wird eine anonyme Klasse erzeugt und instanziert, die die Klasse `WitzG` spezialisiert, also hier die Methoden `bar()` der Klasse `WitzG` überschreibt. Die Instanz `foo` ist als eine Instanz dieser anonymen Klasse zu betrachten. Daher muß die Signatur der Methode `bar()` in beiden Fällen gleich sein. Um dies zu verdeutlichen, ist zusätzlich die alternative Konstruktion `WitzGa` hier angeführt (→ Seite 201).

Beispiel `WitzG` Die Abbildung 7.14 S. 200 zeigt das Klassendiagramm.

Klasse `WitzG`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
```

```
 *  hier: Witztext als anonyme Klasse
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

public class WitzG
{
   public void bar() { }


   public static void main(String[] argv)
   {
      WitzG foo =
         new WitzG()
         {
            public void bar()
            {
               final String text =
                     "Piep, piep ... lieb!";
               System.out.println(text);
            }
         };
      foo.bar();
   }
}
```

Beispiel `WitzGa` Die Abbildung 7.15 S. 202 zeigt das Klassendiagramm.

Klasse `WitzGa`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als anonyme Klasse - Alternative
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
```

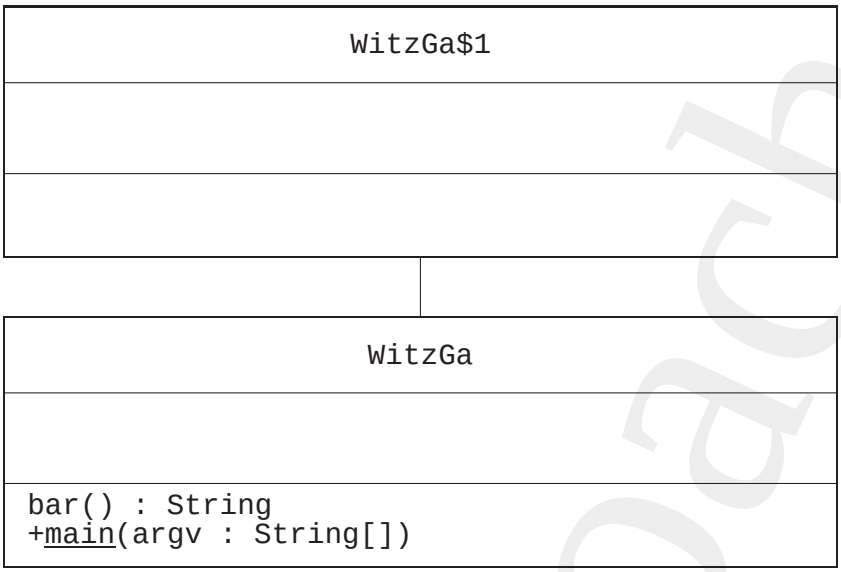


Abbildung 7.15: Klassendiagramm für `WitzGa`

```
 */
public class WitzGa
{
   public String bar()
   {
      final String text = "Piep, piep ... lieb!";
      return text;
   }


   public static void main(String[] argv)
   {
      WitzGa foo =
         new WitzGa()
         {
            public String bar()
            {
               System.out.println(
                     super.bar());
```

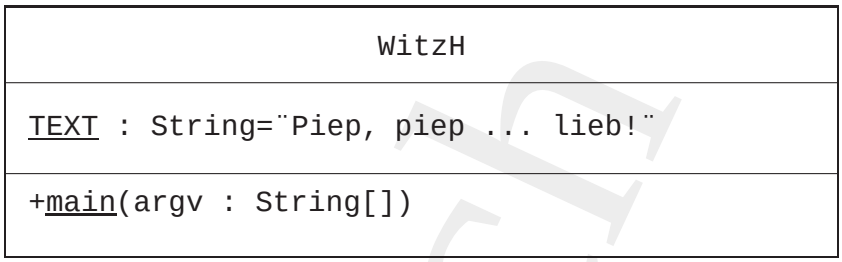


Abbildung 7.16: Klassendiagramm für `WitzH`

```
            return "";
         }
      };

   foo.bar();
  }
}
```

Witztext als Klassenvariable derselben Klasse Eine einfache Konstruktion definiert den Witztext als eine Klassenvariable `text` der eigenen Klasse (↪ Seite 203).

Beispiel `WitzH` Die Abbildung 7.16 S. 203 zeigt das Klassendiagramm.

Klasse `WitzH`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als Klassenvariable derselben
 *  Klasse
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */
public class WitzH
```

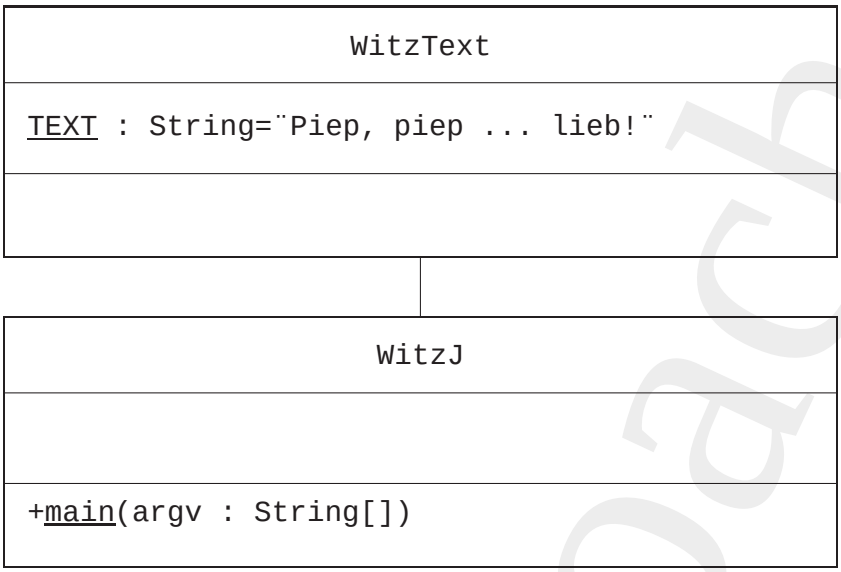


Abbildung 7.17: Klassendiagramm für `WitzJ`

```
{
   final static String TEXT
          = "Piep, piep ... lieb!";


   public static void main(String[] argv)
   {
      System.out.println(TEXT);
   }
}
```

Witztext als eigene Klasse mit Klassenvariable Etwas aufwendiger ist eine Klassenvariable `text` in einer eigenen Klasse `WitzText` (↪ Seite 204).

Beispiel `WitzJ` Die Abbildung 7.17 S. 204 zeigt das Klassendiagramm.

Klasse `WitzJ`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als eigene Klasse mit
 *  Klassenvariable
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */
public class WitzJ
{
   public static void main(String[] argv)
   {
      System.out.println(WitzText.TEXT);
   }
}
class WitzText
{


   final static String TEXT
         = "Piep, piep ... lieb!";
}
```

Witztext als Klassenvariable der Superklasse Als Klassenvariable `TEXT` der Superklasse kann aufgrund der Vererbung die eigene Klasse `WitzK` referenziert werden (↪ Seite 205).

Beispiel `WitzK`

Klasse `WitzK`

```
/**
 *  Kleines Beispiel fuer ,,Konstruktionsalternativen''
 *  hier: Witztext als Klassenvariable der
 *  Superklasse
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
```

```
 *@version    1.1
 */

public class WitzK extends WitzText
{
   public static void main(String[] argv)
   {

      System.out.println(WitzK.TEXT);
   }
}

class WitzText
{


   final static String TEXT = "Piep, piep ... lieb!";
}
```

7.4.1 Beispiel `Aussen`

Nachdem die verschiedene Konstruktionen über die Ausgabe eines Witztextes den Mechanismus der *Inner Classes* im Gesamtzusammenhang verdeutlicht haben, wird nun anhand eines neuen Beispiels die Schachtelungstiefe erhöht.[5] Zu beachten ist dabei, daß im `new`-Konstrukt die Klasse relativ zur Instanz, die diese enthält, angegeben wird, das heißt zum Beispiel:

```
Aussen.Innen.GanzInnen g = i.new GanzInnen();
```

und nicht:

```
Aussen.Innen.GanzInnen g = new Aussen.Innen.GanzInnen();
```

Klasse `Aussen`

[5] Die Idee zum folgenden Beispiel für „*inner classes*“ stammt von [Flanagan97] S. 109–110.

```
/**
 *  Kleines Beispiel fuer einen ,,inner classes''
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */
public class Aussen
{
   public String name = "Aussen";


   public class Innen
   {
      public String name = "Innen";


      public class GanzInnen
      {
         public String name = "GanzInnen";


         public void printSituation()
         {
            System.out.println(name);
            System.out.println(this.name);
            System.out.println(GanzInnen.this.name);
            System.out.println(Innen.this.name);
            System.out.println(Aussen.this.name);
         }
      }
   }


   public static void main(String[] argv)
   {
      Aussen a = new Aussen();
      Aussen.Innen i = a.new Innen();
      Aussen.Innen.GanzInnen g = i.new GanzInnen();
      g.printSituation();
   }
}
```

Die inneren Klassen werden als eigene Dateien erzeugt. Der Dateiname besteht aus den Namen der „äußeren Klassen“ jeweils getrennt durch ein Dollarzeichen, dem Klassennamen und dem Suffix `.class`.

Protokolldatei `Aussen.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac Aussen.java

D:\bonin\anwd\code>dir Aussen*
  763 Aussen$Innen$GanzInnen.class
  518 Aussen$Innen.class
  610 Aussen.class
  854 Aussen.java

D:\bonin\anwd\code>java Aussen
GanzInnen
GanzInnen
GanzInnen
Innen
Aussen

D:\bonin\anwd\code>
```

7.4.2 Beispiel `BlinkLicht`

Das eine *Inner Class* nützlich sein kann, soll das folgende Applet `BlinkLicht` verdeutlichen. Um es zu verstehen, sei an dieser Stelle kurz (nochmals) erwähnt, daß die Methode `paint()` auf zwei Weisen appliziert wird:

1. expliziter Aufruf

durch Angabe von `paint()`, `repaint()` oder `setVisible(true)` und

2. automatischer („impliziter“) Aufruf
immer dann, wenn sich die Sichtbarkeit des Fensters am Bildschirm ändert, zum Beispiel durch Verschieben, Verkleinern, oder indem Verdecktes wieder Hervorkommen soll.

Die Methode `drawImage()` hat hier vier Argumenten. Im Quellcode steht folgendes Statement:

```
g.drawImage(bild,0,0,this);
```

Die Argumente haben folgende Bedeutung:

- Das erste Argument ist eine Referenz auf das Bildobjekt.
- Das zweite und dritte Argument bilden die x,y-Koordinaten ab, an deren Stelle das Bild dargestellt werden soll. Dabei wird durch diese Angabe die linke, obere Ecke des Bildes bestimmt.
- Das vierte Argument ist eine Referenz auf ein Objekt von `ImageObserver`.

`ImageObserver` ist ein Objekt „auf welchem das Bild gezeigt“ wird. Hier ist es durch `this` angegeben, also durch das Applet selbst. Ein `ImageObserver`-Objekt kann jedes Objekt sein, daß das Interface `ImageObserver` implementiert. Dieses Interface wird von der Klasse `Component` implementiert. Da `Applet` eine Unterklasse von `Panel` ist und `Panel` eine Unterklasse von `Container` und `Container` eine Unterklasse von `Component`, ist in einem Applet das Interface `ImageObserver` implementiert.

Datei `BlinkLicht.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Testbett fuer Applet BlinkLicht.class           -->
<!-- Bonin  12-May-1998                              -->
<!-- Update ... 09-Oct-2003                          -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
```

```
<head>
<title>Blinklicht</title>
<style type="text/css">
  p.links {
    text-align: left;
  }
  em {
    font-size: 28pt;
    font-style: italic;
    color: #FFFFFF;
    background-color: #000000;
  }
  body {
    color: white;
    background-color: #000000
  }
</style>
</head>
<body>
<p class="links">
<applet
  name="Blinker"
  code="BlinkLicht.class"
  width="100"
  height="260"
  standby="Hier kommt gleich ein Blinklicht!"
  alt="Java Applet BlinkLicht.class">
  Java Applet BlinkLicht.class
</applet>
</p>
<p><em>Gelbes Blinklicht</em> mittels Thread</p>
<p>Copyright Bonin 1998 all rights reserved</p>
<address>
<a href="mailto:bonin@fhnon.de">bonin@fhnon.de</a>
</address>
</body>
<!-- End of File BlinkLicht.html -->
</html>
```

Klasse `BlinkLicht`

```
/**
 *  Kleines Beispiel fuer einen ,,Thread'', Idee
 *  aus: Hubert Partl; Java-Einfuehrung, Version
 *  April 1998, Mutsterlösungen, S. 33 http://www.boku.ac.at/javaeinf/
 *  Quellcode leicht modifiziert
```

```
 *
 *@author     Hinrich Bonin
 *@version    1.0
 */
import java.applet.*;
import java.awt.*;

public class BlinkLicht extends Applet
       implements Runnable
{

   Graphics grafik;
   Image bild;

   MyCanvas theCanvas;

   Thread myT = null;


   public void init()
   {
      setLayout(new FlowLayout());
      theCanvas = new MyCanvas();
      theCanvas.setSize(100, 260);
      add(theCanvas);
      setVisible(true);

      Dimension d = theCanvas.getSize();
      bild =
            theCanvas.createImage(d.width, d.height);
      grafik = bild.getGraphics();
   }


   public void start()
   {
      if (myT == null)
      {
         myT = new Thread(this);
         myT.start();
      }
      System.out.println("start() appliziert");
```

```
    }


    public void stop()
    {
       if (myT != null)
       {
          myT.stop();
          myT = null;
       }
       System.out.println("stop() appliziert");
    }


    public void run()
    {
       boolean onOff = false;
       while (true)
       {
          grafik.setColor(Color.black);
          grafik.fillRect(10, 10, 80, 240);
          if (onOff)
          {
             grafik.setColor(Color.yellow);
             grafik.fillOval(20, 100, 60, 60);
          }
          onOff = !onOff;
          theCanvas.repaint();
          try
          {
             Thread.sleep(1000);
          } catch (InterruptedException e)
          {
          }
       }
    }


    private class MyCanvas extends Canvas
    {

       public Dimension getMinimumSize()
```

```
        {
            return new Dimension(100, 260);
        }


        public Dimension getPreferredSize()
        {
            return getMinimumSize();
        }


        public void paint(Graphics g)
        {
            update(g);
        }


        public void update(Graphics g)
        {
            if (bild != null)
            {
                g.drawImage(bild, 0, 0, this);
            }
        }
    }

}
```

Protokolldatei `BlinkLicht.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac BlinkLicht.java -deprecation
BlinkLicht.java:55: warning: stop()
  in java.lang.Thread has been deprecated
            myT.stop();
                ^
1 warning
```

Legende:
Java Applet `BlinkLicht` dargestellt mit *Microsoft Internet Explorer Version: 6.0*

Abbildung 7.18: *Inner class* — Beispiel `BlinkLicht`

```
D:\bonin\anwd\code>dir BlinkLicht*

  171 BlinkLicht$1.class
  978 BlinkLicht$MyCanvas.class
2.066 BlinkLicht.class
1.086 BlinkLicht.html
2.074 BlinkLicht.java

D:\bonin\anwd\code>
```

↪ Abbildung 7.18 Seite 214

7.5 Interna einer Klasse (*Reflection*)

Das Paket `java.lang.reflect` ermöglicht zusammen mit der Klasse `java.lang.Class` auf die Interna[6] einer Klasse oder eines Interfaces zuzugreifen. Einige Möglichkeiten dieser sogenannten *Reflection* zeigt das folgende Beispiel `ZeigeKlasse` (↪ Seite 216).

Ausgangspunkt ist die Möglichkeit eine Klasse dynamisch zu laden, indem man der Methode `forName()` von `Class` als Argument den voll qualifizierten Klassennamen (Paketname plus Klassennamen) übergibt. Die Methode `forName(String className)` lädt die Klasse in den Java Interpreter, falls sie nicht schon geladen ist. Rückgabewert ist ein Objekt vom Datentyp `Class`. Mit dem Beispielprogramm `ZeigeKlasse` soll beispielsweise die existierende Klasse `OttoPlan` reflektiert werden und zwar mit folgendem Aufruf:

```
C:\myjava>java ZeigeKlasse OttoPLan
```

Die Klasse `OttoPlan` erhält man innerhalb von `ZeigeKlasse` als eine Klasse (≡ Instanz von `Class`) mit folgendem Statement:

```
Class c = Class.forName(argv[0]);
```

Das Paket `java.lang.reflect` hat die Klassen `Field`, `Constructor` und `Method` für die Abbildung von Feldern (= Variablen), Konstruktoren und Methoden (hier von `OttoPlan`). Objekte von ihnen werden zurückgegeben von den Methoden `getDeclared...()` der Klasse `Class`.

```
Field[] myFields = c.getDeclaredFields();
Constructor[] myConstructors
                  = c.getDeclaredConstructors();
Method[] myMethods = c.getDeclaredMethods();
```

Auch lassen sich auch die implementierten Interfaces mit einer Methode `getInterfaces()` verfügbar machen.

[6] Interna ≈ nur die inneren, eigenen Verhältnisse angehenden Angelegenheiten; vorbehaltenes eigenes Gebiet

```
Class[] myInterfaces = c.getInterfaces();
```

Die Klasse `java.lang.reflect.Modifier` definiert einige Konstanten und Klassenmethoden, um die Integerzahlen, die von der Methode `getModifiers()` zurückgegeben werden, zu interpretieren. Mit `Modifier.toString(c.getModifiers())` erhält man daher die Modifikatoren in Form der reservierten Wörter.

Das Interface `java.lang.reflect.Member` wird von den Klassen `Field`, `Method` und `Constructor` implementiert. Daher kann man die folgende Methode:

```
public static void printMethodOrConstructor(
    Member member)
```

einmal mit einem Objekt der Klasse `Method` und das andere Mal mit einem Objekt der Klasse `Constructor` aufrufen. Die Typerkennung erfolgt dann mit dem Operator `instanceof` und einem *Casting*, beispielsweise in der folgenden Form:

```
Method m = (Method) member;
```

In der Ausgabe von `ZeigeKlasse` sind die Methoden mit ihrem Namen und dem Typ ihrer Parameter angegeben. Die Parameternamen selbst fehlen, denn diese werden nicht in der `class`-Datei gespeichert und sind daher auch nicht über das *Reflection Application Programming Interface* (API) verfügbar.

Klasse `ZeigeKlasse`

```
/**
 *  Kleines Beispiel fuer die ,,Reflection
 *  API''-Moeglichkeiten Idee von David Flanagan;
 *  Java Examples in a Nutshell, 1997, p. 257
 *  Quellcode modifiziert.
 *
 *@author     Hinrich Bonin
 *@version    1.1
 */
import java.lang.reflect.*;

public class ZeigeKlasse
{
```

```
String[] myWitz
       = new String[]{"Piep", "piep", "...", "lieb!"};


public static void main(String argv[])
       throws ClassNotFoundException
{

   Class c = Class.forName(argv[0]);
   printClass(c);
}


/**
 *  Gibt von der Klasse die Modifkatoren, den
 *  Namen, die Superklasse und das Interface
 *  aus.
 *
 *@param  c  Description of the Parameter
 */
public static void printClass(Class c)
{
   if (c.isInterface())
   {
      /*
       *  Modifikatoren enthalten das Wort "interface"
       */
      System.out.print(
            Modifier.toString(c.getModifiers())
             + c.getName());
   }
   /*
    *  es gibt kein c.isClass() daher else
    */
       else if (c.getModifiers() != 0)
   {
      System.out.print(
          Modifier.toString(c.getModifiers())
           + " class " + c.getName() + " extends "
           + c.getSuperclass().getName());
   } else
   {
```

```
      /*
       *  Modifier.toString(0) gibt "" zurueck
       */
      System.out.print("class " + c.getName());
   }

   /*
    *  Interfaces oder Super-Interfaces
    *  einer Klasse oder eines Interface
    */
   Class[] myInterfaces = c.getInterfaces();
   if ((myInterfaces != null) &&
         (myInterfaces.length > 0))
   {
      if (c.isInterface())
      {
         System.out.println(" extends ");
      } else
      {
         System.out.print(" implements ");
      }
      for (int i = 0; i < myInterfaces.length; i++)
      {
         if (i > 0)
         {
            System.out.print(", ");
         }
         System.out.print(
               myInterfaces[i].getName());
      }
   }
   /*
    *  Beginnklammer fuer Klassenbody
    */
   System.out.println(" {");

   /*
    *  Ausgabe der Felder
    */
   System.out.println("  // Feld(er)");
   Field[] myFields = c.getDeclaredFields();
   for (int i = 0; i < myFields.length; i++)
```

```
        {
           printField(myFields[i]);
        }

        /*
         *  Ausgabe der Konstruktoren
         */
        System.out.println("  // Konstruktor(en)");
        Constructor[] myConstructors =
              c.getDeclaredConstructors();
        for (int i = 0; i < myConstructors.length; i++)
        {
           printMethodOrConstructor(myConstructors[i]);
        }

        /*
         *  Ausgabe der Methoden
         */
        System.out.println("  // Methode(n)");
        Method[] myMethods = c.getDeclaredMethods();
        for (int i = 0; i < myMethods.length; i++)
        {
           printMethodOrConstructor(myMethods[i]);
        }

        /*
         *  Endeklammer fuer Klassenbody
         */
        System.out.println("}");
    }


    /**
     *  Drucken Methoden und Konstruktoren
     *
     *@param  member  Description of the Parameter
     */
    public static void printMethodOrConstructor
          (Member member)
    {
        Class returnType = null;
        Class myParameters[];
```

```
        Class myExceptions[];
        if (member instanceof Method)
        {
           Method m = (Method) member;
           returnType = m.getReturnType();
           myParameters = m.getParameterTypes();
           myExceptions = m.getExceptionTypes();
        } else
        {
           Constructor c = (Constructor) member;
           myParameters = c.getParameterTypes();
           myExceptions = c.getExceptionTypes();
        }
        System.out.print("  " +
              modifiersSpaces(member.getModifiers()) +
              ((returnType != null) ? typeName(returnType)
               + " " : "") +
              member.getName() + "(");
        for (int i = 0; i < myParameters.length; i++)
        {
           if (i > 0)
           {
              System.out.print(", ");
           }
           System.out.print(
                 typeName(myParameters[i]));
        }
        System.out.print(")");
        if (myExceptions.length > 0)
        {
           System.out.print(" throws ");
        }
        for (int i = 0; i < myExceptions.length; i++)
        {
           if (i > 0)
           {
              System.out.print(", ");
           }
           System.out.print(typeName(myExceptions[i]));
        }
        System.out.println(";");
     }
```

```
/**
 *  Aufbereitung der Modifiers mit Zwischenraeumen
 *
 *@param  m  Description of the Parameter
 *@return    Description of the Return Value
 */
public static String modifiersSpaces(int m)
{
   if (m == 0)
   {
      return "";
   } else
   {
      return Modifier.toString(m) + " ";
   }
}


/**
 *  Feld-Ausgabe mit Modifiers und Type
 *
 *@param  f  Description of the Parameter
 */
public static void printField(Field f)
{
   System.out.println("  " +
         modifiersSpaces(f.getModifiers()) +
         typeName(f.getType()) + " " +
         f.getName() + ";");
}


/**
 *  Aufbereitung des Namens mit Array-Klammern
 *
 *@param  t  Description of the Parameter
 *@return    Description of the Return Value
 */
public static String typeName(Class t)
{
```

```
      String myBrackets = "";
      while (t.isArray())
      {
         myBrackets += "[]";
         t = t.getComponentType();
      }
      return t.getName() + myBrackets;
   }
}
```

Protokolldatei `ZeigeKlasse.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac ZeigeKlasse.java

D:\bonin\anwd\code>java ZeigeKlasse ZeigeKlasse
public class ZeigeKlasse extends java.lang.Object {
  // Feld(er)
  java.lang.String[] myWitz;
  // Konstruktor(en)
  public ZeigeKlasse();
  // Methode(n)
  public static void main(java.lang.String[])
    throws java.lang.ClassNotFoundException;
  public static void printClass(java.lang.Class);
  public static void
    printMethodOrConstructor(java.lang.reflect.Member);
  public static java.lang.String modifiersSpaces(int);
  public static void printField(java.lang.reflect.Field);
  public static java.lang.String typeName(java.lang.Class);
}

D:\bonin\anwd\code>
```

7.6 Referenzen & *Cloning*

Wenn man eine „Kopie“ eines Objektes mittels einer Zuweisung anlegt, dann verweist die Referenz der Kopie auf dasselbe Orginalobjekt. Zum Beispiel sollen zwei fast gleiche Kunden erzeugt werden. Dann bietet es sich an, eine Kopie vom zunächst erzeugten Kunden für den zweiten Kunden als Ausgangsbasis zu nutzen.

```
Kunde original
    = new Kunde("Emma AG", "Hamburg", "4-Nov-98");
Kunde copie = original;
// Peng! Emma AG vernichtet
copie.setName("Otto AG");
```

Wenn man eine Kopie als ein neues Objekt benötigt, dann muß das Objekt geklont werden. Dazu dient die Methode `clone()`.

```
Kunde original
    = new Kunde("Emma AG", "Hamburg", "4-Nov-98");
Kunde copie
   = (Kunde) orginal.clone();
// OK! original bleibt unveraendert
copie.setName("Otto AG");
```

Die Dinge mit der Methode `clone()` sind aber nicht ganz so einfach. Man stelle sich einmal vor, wie die Methode `clone()` der Klasse `Object` arbeiten kann. Sie hat keine Information über die Struktrur der Klasse `Kunde`. Daher kann sie nur eine *Bit-für-Bit*-Kopie fertigen. Bei nicht primitiven Objekten stellen diese Bits zum Teil Referenzen auf andere Objekt dar. Da nun diese Bits genauso in der Kopie enthalten sind, verweist die Kopie auf dieselben Objekte wie das Orginal. Es gilt daher drei Fälle beim *Cloning* zu unterscheiden:

Bit→Bit

Cloning

1. Problemloses *Cloning*
   Die Default-Methode `clone()` reicht aus, weil das Orginal nur aus primivtiven Objekten besteht oder die referenzierten Objekte werden später nicht modifiziert.

2. Mühsames *Cloning*
   Für das Objekt kann eine geeignete Methode `clone()` definiert

werden. Für jede Instanzvariable wird die Default-Methode `clone()` mit Zusammenhang mit einer `Cast`-Operation explizit angewendete.

3. Hoffnungsloses *Cloning*
   Häufiger Fall — man muß auf das *Cloning* verzichten.

Für den Zugriff auf die Methode `clone()` hat die Klasse das Interface `Cloneable` zu implementieren. Darüberhinaus ist `clone()` der Klasse `Object` zu redefinierten und zwar mit dem Zugriffrecht `public`. Dabei verhält sich das Interface `Cloneable` anders als ein übliches Interface. Man kann es sich mehr als einen Erinnerungsposten für den Programmier vorstellen. Er verweist darauf, daß *Cloning* nicht ohne Kenntnis des Kopierprozesses angewendete werden kann.

Das folgende Beispiel `Person` enthält darüberhinaus ein paar Konstruktionsaspekte, die zum Nachdenken anregen sollen. Einerseits zeigt es eine rekursive Beschreibung einer Person durch den Ehegatten, der selbst wieder eine Person ist. Andererseits nutzt es die Klasse `Vector`[7].

Klasse `Person`

```
/**
 *  Beispiel einer Rekursion innerhalb der Klasse:
 *  Der Ehegatte und die Kinder sind wieder eine
 *  Person
 *
 *@author     Hinrich Bonin
 *@version    1.1
 */
import java.util.*;

class Person implements Cloneable
{
   private String name = "";
   private Person ehegatte;
   private Vector kinder = new Vector();
```

[7] Hinweis: In dem ursprünglichen `JDK 1.1.n` gab es keine Methode `add(int index, Object element)` in der Klasse `Vector`.

```
public Person(String name)
{
   this.name = name;
   System.out.println(name + " lebt!");
}


public String getName()
{
   return name;
}


public Person getEhegatte()
{
   return ehegatte;
}


public Person getKind(int i)
{
   return (Person) kinder.get(i - 1);
}


/*
 *  Achtung!
 *  Set-Metode mit Return-Wert.
 */
public Person setName(String name)
{
   this.name = name;
   return this;
}

public void setEhegatte(Person ehegatte)
{
   this.ehegatte = ehegatte;
}
```

```
public void setKind(Person kind, int i)
{
   kinder.add(i - 1, (Object) kind);
}


public Object clone()
{
   try
   {
      return super.clone();
   } catch (CloneNotSupportedException e)
   {
      System.out.println("Objekt nicht geklont");
      return null;
   }
}


public static void main(String[] argv)
{
   Person oE = new Person("Otto Eiderente");
   /*
    *  Applikation der set-Methode mit Retrun-Wert
    */
   Person eE =
         (new Person(
         "Musterperson")).setName("Emma Eiderente");

   oE.setEhegatte(eE);
   eE.setEhegatte(oE);

   Person madel =
         new Person("Emmchen Eiderente");
   eE.setKind(madel, 1);
   oE.setKind(eE.getKind(1), 1);

   Person junge =
         new Person("Ottochen Eiderente");
   junge.setEhegatte(madel);
   eE.setKind(junge, 2);
   oE.setKind(eE.getKind(2), 2);
```

```
      System.out.println(
            "Person eE: Ehegatte von Ehegatte ist " +
            eE.getEhegatte().getEhegatte().getName());
      System.out.println(
            "Ehegatte vom zweiten Kind\n" +
            "    vom Ehegatten ist " +
            eE.getEhegatte().getKind(2).getEhegatte().getName());

      /*
       *  Simple Loesung fuer das Problem der
       *  mehrfachen Applikation der
       *  Methode getEhegatte() auf das
       *  jeweilige Resultatobjekt.
       */
      /*
       *  Das Clonen (= Bitstromkopierung)
       *  sichert hier nur
       *  die urspruengliche Referenz fuer eE
       */
      Person eEclone = (Person) eE.clone();
      int i;
      for (i = 1; (i < 4); i++)
      {
         eEclone = eEclone.getEhegatte();
      }
      System.out.println("Von " + eE.getName() +
            " ausgehend immer wieder Ehegatte\n" +
            "    von Ehegatte ist " +
            eEclone.getName());
   }
}
```

Protokolldatei `Person.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)
```

```
D:\bonin\anwd\code>javac Person.java

D:\bonin\anwd\code>java Person
Otto Eiderente lebt!
Musterperson lebt!
Emmchen Eiderente lebt!
Ottochen Eiderente lebt!
Person eE: Ehegatte von Ehegatte ist Emma Eiderente
Ehegatte vom zweiten Kind
    vom Ehegatten ist Emmchen Eiderente
Von Emma Eiderente ausgehend immer wieder Ehegatte
    von Ehegatte ist Otto Eiderente

D:\bonin\anwd\code>
```

7.7 Integration eines ODBMS — Beispiel `FastObjects`

POET

ODMG

Als charakteristisches Beispielprodukt für ein objekt-orientiertes Daten-BankManagementSystem dient im Folgenden das Produkt `FastObjects` der Firma POET Software GmbH, Hamburg[8]. Das Modell der ODMG[9] (*Object Data Management Group*) für persistente Objekte in einer Datenbank spezifiziert das Erzeugen eines solchen Objektes im Rahmen einer Transaktion.

7.7.1 Transaktions-Modell

Man kreiert ein solches „Transaktionsobjekt“[10] zwischen den Methode `begin()` und `commit()`.

```
import com.poet.odmg.*;
... // Deklaration von Klassen
```

[8] Beziehungsweise: POET Software Corporation, San Mateo, USA

[9] ODMG ≡ vormals: Object Database Management Group,
↪ `http://www.odmg.org/frrbottom.htm` (Zugriff: 27-May-1998)

[10] Bei dieser Form von Transaktion handelt es sich um den Typ *Short Running Transaction*. Bei der Abbildung von komplexen Geschäftsprozesses gibt es häufig noch den Typ *Long Running Transaction* bei dem beispielsweise mehrere länger laufender Batch-Prozesse in die Transaktion einzubinden sind.

```
Transaction txn = new Transaction();
txn.begin();
... // Datenbankobjekte werden immer
... // innerhalb der Transaktion erzeugt.
txn.commit();
```

Das Erzeugen eines Objektes für die Datenbank und die Zuweisung von Werten geschieht auf die übliche Art und Weise, jedoch innerhalb der Transaktion. Ein Beispiel sei hier die Instanz `myP` einer Klasse `Person`.

```
class Person {
  private String name;
  private Person ehegatte;
   ...
  public static void main(String[] argv) {
    Person myP = new Person();
    myP.setName("Emma Musterfrau");
    Person myE = new Person();
    myE.setName("Otto Mustermann");
    myP.setEhegatte(myE);
  }
}
```

7.7.2 Speichern von Objekten mittels Namen

Noch ist das Objekt `myP` nicht in einer Datenbank gespeichert. Dazu muß zunächst ein Objekt `database` erzeugt werden. Mit Hilfe der Methode `bind()` geschieht dann die Zuordnung zwischen dem Objekt `myP` und der Datenbank `database`. Um das Objekt `myP` später wieder aus der Datenbank `database` zu selektieren, wird als zweites Argument von `bind()` ein String als kennzeichnender Name übergeben:

bind()

```
database.bind(myP,"EMusterF");
```

Die Methode `bind()` speichert auch noch nicht endgültig das Objekt `myP` in die Datenbank. Dies geschieht erst zum Zeitpunkt des Aufrufs der Methode `commit()`. Wird beispielsweise die Transaktion mit der Methode `abort()` abgebrochen, dann ist `myP` nicht gespeichert.

Hier sei nochmals kurz zusammengefaßt der Quellcode dargestellt um das Objekt `myP` zu erzeugen und persistent in der Datenbank `database` zu speichern.

```
import com.poet.odmg.*;
... // Deklaration von Klassen
Transaction txn = new Transaction();
txn.begin();
  Person myP = new Person();
  myP.setName("Emma Musterfrau");
  database.bind(myP,"EMusterF");
txn.commit();
```

Aus der Datenbank wird das Objekt mit der Methode `lookup()` wiedergewonnen und anschließend mittels einer *Casting*-Operation rekonstruiert.

lookup()

```
Person p = (Person) database.lookup("EMusterF");
System.out.println(p.name);
```

Man kann auch ein Objekt ohne Namen in der Datenbank ablegen. Dazu wird die Methode `bind()` mit dem zweiten Argument `null` aufgerufen.

```
database.bind(myP,null);
```

Solche Datenbankobjekte ohne Namen werden häufig mit Hilfe von `Extent` selektiert (↪ Abschnitt 7.7.5 S. 232).

7.7.3 Referenzierung & Persistenz

In Java^{TM} werden Beziehungen zwischen Objekten durch Referenzen abgebildet. Hat beispielsweise eine Person einen Ehegatten, dann wird auf das Objekt Ehegatte über eine Referenz (zum Beispiel über einen Instanzvariablennamen) zugegriffen. Damit sich das aus der Datenbank rekonstruierte Objekt genauso verhält wie das ursprüngliche, müssen alle referenzierten Objekte ebenfalls gespeichert werden. Diese Notwendigkeit wird als *persistence-by-reachability* charakterisiert. In unserem Beispiel „Ehegatte“ muß ein Objekt Ehegatte mit gespeichert werden, und zwar zum Zeitpunkt, wenn die Person selbst gespeichert wird.

```
Transaction txn = new Transaction();
txn.begin();
  Person myP = new Person();
  myP.setName("Emma Musterfrau");
  Person myE = new Person();
  myE.setName("Otto Musterfrau");
  myP.setEhegatte(myE);
  database.bind(myP,"EMusterF");
txn.commit();
```

Nachdem das Objekt `myP` aus der Datenbank wieder selektiert und rekonstruiert ist, ist auch das Objekt „Ehegatte“ wieder verfügbar.

```
Person p = (Person) database.lookup("EMusterF");
// Gibt "Otto Musterfrau" aus.
System.out.println(p.getEhegatte().getName());
```

Im Regelfall ist beim Speichern eines Objektes ein umfangreiches Netzwerk von referenzierten Objekten betroffen, damit das Orginalverhalten des Objektes nach dem Selektieren und Rekonstruieren wieder herstellbar ist.

7.7.4 Collections

Eine *Collection* ermöglicht einem Objekt, eine Sammlung mit mehreren Objekten[11] zu enthalten. Beim Beispiel „Ehegatte“ könnten so die Kinder des Ehepaares abgebildet werden. Der folgende Quellcode gibt daher die Kinder von `Emma Musterfrau` aus.

```
Person p = (Person) database.lookup("EMusterF");
Iterator e = p.getKinder().iterator();
while (e.hasNext()) {
  Person k = (Person) e.next();
  // Gibt den Namen des Kindes aus.
  System.out.println(k.getName());
}
```

[11] oder eine Sammlung von Referenzen auf mehrere Objekte

ODMG-Collections		
Typ	**Sortierung**	**Duplikate**
`Bag`	vom System bestimmt	Ja
`List`	selbstgewählt	Ja
`Set`	vom System bestimmt	Nein
`Array`	selbstgewählt	Ja

Tabelle 7.5: FastObjects Java Binding Collection Interfaces

Entsprechend der ODMG-Spezifikation unterstützt *FastObjects Java Binding* die *Collections* `Bag`, `List` `Set` und `Array`. Die Tabelle 7.5 S. 232 zeigt die Unterschiede in Bezug auf die Sortierung der Objekte und auf das mehrfache Vorkommen des gleichen Objektes.

7.7.5 Extent

Extent

Objekte können gespeichert werden ohne spezifizierten Namen oder indirekt weil sie referenziert werden über ein Objekt mit Namen. Zusätzlich ist es häufig notwendig auf alle Objekte mit „bestimmten Eigenschaften“ in der Datenbank zugreifen zu können und zwar nicht nur über den Weg der einzelnen Objektnamen. Um einen solchen Zugriff auf eine größere Menge von Datenbankobjekten zu ermöglichen, gibt es die Klasse `Extent`[12]. Diese wird benutzt, um alle Objekte einer Klasse in der Datenbank zu selektieren. Immer wenn ein Objekt in die Datenbank gespeichert wird, dann wird eine Referenz für den späteren Zugriff über `Extent` zusätzlich in der Datenbank gespeichert. Der Konstrukor `Extent()` hat daher zwei Parameter: 1. die gewählte Datenbank und 2. den Klassennamen, der zu selektierenden Objekte. In dem obigen Beispiel wäre folgende Konstruktion erforderlich:

```
Extent allePersonen = new Extent(database, "Person");
```

[12] Derzeit definiert weder Java noch das *ODMG Java Binding* ein Konstrukt `Extent`. Es handelt sich dabei (noch?) um eine spezifische FastObjects-Leistung.

Das `Extent`-Objekt wird dann benutzt um alle einzelnen persistenten Objekte zu rekonstruieren.

```
while (allePersonen.hasMoreElements()) {
  Person p = (Person) allePersonen.nextElement();
  // Gibt den Namen der Person aus.
  System.out.println(p.getName());
}
```

7.7.6 Transientes Objekt & Constraints

In Java^{TM} gehört ein Objekt, das mit dem Modifikator `transient` gekennzeichnet ist, nicht zu den persistenten Objekten. Es wird konsequenterweise auch nicht in der Datenbank gespeichert. Ein solches transientes Objekt existiert nur zur Laufzeit des Programms im Arbeitsspeicher. Im Folgenden sei eine Instanzvariable `alter` ein solches transientes Objekt.

```
import java.util.*;
class Person {
  private String name;
  private Date geburtstag;
  private transient int alter;
}
```

Wenn eine Instanz `myP` der Klasse `Person` aus der Datenbank selektiert wird, dann muß beim Rekonstruieren von `myP` auch der Wert von `alter` erzeugt werden. Dazu dient die Methode `postRead()`. Sie wird automatisch vom DBMS nach dem Laden von `myP` aus der Datenbank appliziert. Für die Sicherung der Datenintegrität hält FastObjects drei automatisch applizierte Methoden bereit. Diese werden im Interface `Constraints` vorgegeben.

```
import java.util.*;
class Person implements Constraints {
  private String name;
  private Date geburtstag;
  private transient int alter;
```

```
  // Methode zum Initialisieren
  public void postRead() {
    // Nur als Beispiel --- es geht genauer!
    Date heute = new Date();
    alter = heute.getYear() - geburtstag.getYear();
    // ...
  }

  // Methoden zum Clean-up
  public void preWrite() {
    // ...
  }
  public void preDelete() {
    // ...
  }
}
```

7.7.7 Objekt Resolution

Ein referenziertes Objekt wird vom DBMS erst geladen wenn es tatsächlich benötigt wird. Eine Variable ist daher als spezielles *FastObjects Java reference object* implementiert. Das Laden des referenzierten Objektes in den Arbeitsspeicher wird als *Resolving the Reference* bezeichnet. Anhand eines Beispiels wird dieses *Resolving the Reference* deutlich.

```
class Buch {
  private String titel;
  private Person autor;
  private int jahr;

  public String getTitel() {
    return titel;
  }

  public Person getAutor() {
    return autor;
  }
```

```
  public int getJahr() {
    return jahr;
  }

  public Buch(String titel, String autor, int jahr) {
    this.titel = titel;
    this.autor = new Person(autor);
    this.jahr = jahr;
  }
}
```

Zunächst wird ein Objekt `myBuch` der Klasse `Buch` erzeugt und in der lokalen FastObjects-Datenbank `BuchBase` gespeichert.

```
...
Database myDB = new Database();
myDB.open("poet://LOCAL/BuchBase",
          Database.OPEN_READ_WRITE);
Transaction myT = new Transaction(myDB);
myT.begin();
try {
  Buch myBuch = new Buch(
    "Softwarekonstruktion mit LISP","Bonin",1991);
  myDB.bind(myBuch,"PKS01");
}
catch (ObjectNameNotUniqueException exc) {
  System.out.println("PKS01 gibt es schon!");
}
myT.commit();
myDB.close();
```

Mit einem anderen Programm wird das Buch „Softwarekonstruktion mit LISP“ wieder selektiert.

```
// Datenbank oeffnen und Transaktion starten
// ...
Buch b = (Buch) myDB.lookup("PKS01");
// ...
```

Wenn man jetzt mit Hilfe von `ObjectServices` abfragt, ob das Objekt `b` *resolved* ist, dann erhält man als Wert `false`.

```
// ...
ObjectServices os = ObjectServices.of(myDB);
System.out.println("Buch b  resolved =  " +
  os.isResolved(b));
```

Zu diesem Zeitpunkt ist es für FastObjects nur notwendig eine Referenz zum entsprechenden Buchobjekt `b` in der Datenbank zu erzeugen. Erst wenn man mit diesem Objekt `b` arbeitet, geschieht das *Resolving*.

```
// ...
int buchErscheinungsjahr = b.getJahr();
```

Danach ist der Rückgabewert von `ObjectServices.isResolved(b)` gleich `true`. Auch die Referenz auf den Autor, ein Objekt der Klasse `Person` wird erst aufgelöst, wenn der Wert tatsächlich benötigt wird. Erst nach einer „Benutzung“ der Variablen `autor`, zum Beispiel in der Form:

```
// ...
String inhaltVerantwortlich = b.getAutor();
```

hat `ObjectServices.isResolved(b.getAutor())` den Wert `true`. Das *Resolving*-Konzept beim Ladens eines Objektes aus der Datenbank in den Arbeitsspeicher kann man daher auch als *ondemand*-Laden bezeichnen. Dabei ermöglich FastObjects neben dem automatischen (impliziten) *Resolving* auch ein explizites[13]. Dazu dienen die Methoden `resolve()` und `resolveALL()`.

ondemand

7.7.8 Abfragesprache (*OQL*)

OQL

Für das Arbeiten mit einer objektorientierte Datenbanken hat die ODMG als Abfragesprache OQL (*Object Query Language*) standardisiert. OQL basiert wie SQL[14] auf dem Konstrukt

```
select ... from ... where ...
```

[13] Ein *explizites Resolving* benötigt eine entsprechende Eintragung in der Konfigurationsdatei.

[14] Standard Query Language für eine relationale Datenbank.

FastObjects Java Binding ermöglicht mit Hilfe der Klasse `OQLQuery` Abfragen nach diesem Standard. Die Abfrage selbst wird als `String`-Objekt spezifiziert und dem Konstruktor `OQLQuery(String query)` übergeben. Für das obige Beispiel käme daher folgender Quellcode in Betracht:

```
import com.poet.odmg.util.*;
import com.poet.odmg.*;
import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;
import java.util.*;
.
.
.
String abfrageString  =
  "define extent allePersonen for Person;" +
  "select p from p in allePersonen" +
    "where p.getName() = \"Emma Musterfrau\";";
OQLQuery abfrage = new OQLQuery(abfrageString);
Object result = abfrage.execute();
.
.
.
```

Wenn die Abfrage eine *Collection* von Objekten ergibt, dann sind die einzelnen Objekte mit Hilfe der Klasse `Iterator` zugreifbar.

```
import com.poet.odmg.util.*;
import com.poet.odmg.*;
import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;
import java.util.*;
.
.
.
try
  {
    String query =
      "define extent alleBuecher for Buch;" +
```

```
      "select buch from buch in alleBuecher";
       OQLQuery abfrage = new OQLQuery(query);
       Object result = abfrage.execute();
       Iterator e = ((CollectionOfObject)
                      result).iterator();
         while(e.hasNext())
         {
           Buch buch = (Buch) e.next();
           System.out.println(buch.getTitel());
         }
    }
.
.
.
```

7.7.9 Enhancer `ptj`

Die Firma POET hat ihr *Java ODMG Binding*[15] mit Hilfe des speziellen Programms `ptj`, genannt *Enhancer*, realisiert. Das Programm `ptj` liest Java-Klassen im Bytecode, das heißt die `*.class`-Dateien, und verarbeitet diese. Dazu extrahiert `ptj` die relevanten Daten um die persistenten Klassentypen in der `Dic`tionary-Datenbank zu registrieren. Der *Enhancer* erzeugt zusätzliche `*.class`-Dateien mit dem Namen `_pt_meta.*.class`. Es wird stets eine neue, leere Datenbank erzeugt, wenn die angegebene nicht schon existiert. Der *Enhancer* wird standardmäßig mit folgenden Parametern aufgerufen:

```
ptj -enhance -inplace
```

Die Konfigurationsdatei `ptj.opt` enthält die Daten, welche Klasse persistent ist. Für eine persistente Klasse `Foo` ist eine Eintragung in folgender Form notwendig:

```
[classes\Foo]
persistent = true
```

[15] ODMG Standard Release 2.0

Die Konfigurationsdatei `ptj.opt` enthält auch den Namen der Datenbank und den Namen des Klassenlexikons. Dabei wird das Klassenlexikon als Schema bezeichnet. Beide Namen führen zu entsprechenden Dateien im Filesystem des Betriebssystems. Um die Zugriffsgeschwindigkeit zu verbessern, kann die Datenbank als Indexdatei plus Inhaltsdatei im Filesystem abgebildet werden. Gleiches gilt auch für das Schema Diese Aufsplittung geschieht bei der Eintragung `oneFile = false`.

```
[schemata\myDict]
oneFile = true

[databases\myBase]
oneFile = true
```

7.7.10 Beispiel: `Bind`, `Lookup` und `Delete`

Das Einführungsbeispiel[16] verdeutlich die Unterscheidung in

- *persistence capable class*
  ≡ Klasse, die persistenzfähig ist. Sie hat einen Schema-Eintrag in der Konfigurationsdatei ↪ hier: `MyClass`. Ihre Objekte werden in der Datenbank gespeichert.

- und *persistence aware class*.
  ≡ Klasse, die Kenntnis von der Persistenz hat. Sie nutzt persistente Objekte. Sie ist nicht in der Konfigurationsdatei vermerkt ↪ hier: `Bind`,`Lookup` und `Delete`.

Das hier genutzte FastObjects-System läuft auf einer NT-Plattform (Rechner: 193.174.33.100). Die Umgebungsvariable `CLASSPATH` ist vorab um den Ort der FastObjects-Klassen zu ergänzen.

```
set CLASSPATH=%CLASSPATH%;C:\Programme\POET50\Lib\POETClasses.zip;.
ptjavac *.java
java Bind poet://LOCAL/my_base bonin
```

[16] Ursprüngliche Quelle: Inhalt des POET-Paketes 5.0
`/POET50/Examples/JavaODMG/First/`
— jedoch leicht modifiziert.

```
java Lookup poet://LOCAL/my_base bonin
java Delete poet://LOCAL/my_base bonin
```

Als Konfigurationsdateiname wird der *Default*-Name `ptjavac.opt` verwendet. Die Konfigurationsdatei hat folgenden Inhalt:

```
/**
   ptjavac.opt
 */

[schemata\my_dict]
oneFile = true

[databases\my_base]
oneFile = true

[classes\MyClass]
persistent = true
```

Klasse `MyClass`

```
/**
 *  Persistence capable class MyClass
 *
 *@author     Hinrich Bonin
 *@version    1.1
 */
import com.poet.odmg.*;
import com.poet.odmg.util.*;
import java.util.*;

class MyClass
{
   short s;
   int i;
   float f;
   double d;
   Object obj;
   Date aDate;
   String str;
   SetOfObject set;
   BagOfObject bag;
   ArrayOfObject varray;
```

```
   ListOfObject list;
   MyClass myObj;
   boolean aBool;
   long l;

   // Objekt e aus java.util.Enumeration
   // wird nicht gespeichert

   transient Enumeration e;


   public MyClass()
   {
      set = new SetOfObject();
      bag = new BagOfObject();
      varray = new ArrayOfObject();
      list = new ListOfObject();
      i = 11;
      s = 12;
      d = 3.1415926;
      str = "Mein erstes POET Objekt";
   }


   public String toString()
   {
      return str + " (" +
            Integer.toString(i) + ", " +
            Integer.toString(s) + ", " +
            Double.toString(d) + ", " +
            aDate.toString() + ")";
   }
}
```

Klasse Bind

```
/**
 *  Persistence aware class Bind
 *
 *@author     Hinrich Bonin
 *@version    1.1
 */
```

```
import com.poet.odmg.*;
/*
 *  Exception-Import nicht durch org.odmg.* ersetzen,
 *  da dann 2 mal Klasse Database
 */
import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;
import java.util.*;

public class Bind
{
   Bind(Database db, String name)
          throws ODMGException
   {
      Transaction txn = new Transaction();
      txn.begin();
      try
      {
         MyClass myObject = new MyClass();
         db.bind(myObject, name);
      } catch (ObjectNameNotUniqueException exc)
      {
         txn.abort();
         throw exc;
      } catch (ODMGRuntimeException exc)
      {
         txn.abort();
         throw exc;
      }

      txn.commit();
   }


   public static void main(String[] argv)
          throws ODMGException
   {
      if (argv.length < 2)
      {
         System.out.println(
               "Bitte Datenbank und Objektname nennen!");
         System.exit(1);
```

```
      }
      Database db = new Database();
      db.open(argv[0], Database.OPEN_READ_WRITE);
      try
      {
         new Bind(db, argv[1]);
      } finally
      {
         db.close();
      }
   }
}
```

Klasse Lookup

```
/**
 *  Selektieren und Rekonstruieren eines POET-Objektes
 *
 *@author     Hinrich Bonin
 *@version    1.1
 */
import com.poet.odmg.*;
// Exception-Import nicht durch org.odmg.* ersetzen,
// da dann 2 mal Klasse Database
import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;

public class Lookup
{
   Lookup(Database db, String name)
         throws ODMGException
   {
      Transaction txn = new Transaction();
      txn.begin();
      try
      {
         MyClass myObject = (MyClass) db.lookup(name);
         System.out.println(myObject);
      }
```

```
      // Fuer den Fehlerfall
      catch (ObjectNameNotFoundException exc)
      {
         txn.abort();
         throw exc;
      } catch (ODMGRuntimeException exc)
      {
         txn.abort();
         throw exc;
      }
      txn.commit();
   }


   public static void main(String[] argv)
          throws ODMGException
   {
      if (argv.length < 2)
      {
         System.out.println(
               "Bitte Datenbank und Objektname nennen!");
         System.exit(1);
      }
      Database db = new Database();
      db.open(argv[0], Database.OPEN_READ_WRITE);
      try
      {
         new Lookup(db, argv[1]);
      } finally
      {
         db.close();
      }
   }
}
```

Klasse `Delete`

```
/**
 *  Selektieren und Löschen eines POET-Objektes
 *
 *@author     Hinrich Bonin
 *@version    1.1
```

```
 */
import com.poet.odmg.*;
/*
 *  Exception-Import nicht durch org.odmg.* ersetzen,
 *  da dann 2 mal Klasse Database
 */
import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;

public class Delete
{
   Delete(Database db, String name)
          throws ODMGException
   {
      Transaction txn = new Transaction();
      txn.begin();
      try
      {
         ObjectServices os =
              ObjectServices.of(myDB);
         os.delete(db.lookup(name));
         db.unbind(name);
      }
      // Fuer den Fehlerfall
      catch (ObjectNameNotFoundException exc)
      {
         txn.abort();
         throw exc;
      } catch (ODMGRuntimeException exc)
      {
         txn.abort();
         throw exc;
      }
      txn.commit();
   }


   public static void main(String[] args)
          throws ODMGException
   {
      if (args.length < 2)
      {
```

```
            System.out.println(
                  "Bitte Datenbank und Objektname nennen!");
            System.exit(1);
         }
         Database db = new Database();
         db.open(args[0], Database.OPEN_READ_WRITE);
         try
         {
            new Delete(db, args[1]);
         } finally
         {
            db.close();
         }
      }
}
```

7.8 Zusicherung über Werte

assert

Wenn man sicherzustellen möchte, dass eine Variable einen bestimmten Wert hat oder eine Wertgrenze einhält, dann kann statt eines `if`-Konstruktes dazu das `assert`-Konstrukt[17] genutzt werden. Zur Laufzeit wird dann überprüft, ob die *Assertion* erfüllt ist, wenn nicht, dann wird eine Ausnahme ausgelöst. Soll beispielsweise die Instanzvaribale i nicht negativ werden, dann läßt sich diese Zusicherung wie folgt notieren:

```
int i;
...
assert i >= 0;
...
```

Das `assert`-Konstrukt ist erst ab Java Version 1.4 verfügbar. Aus Gründen der Kompatibilität ist daher beim Compilieren und beim Ausführen auf das `assert`-Konstrukt wie folgt zu verweisen:

```
>javac -source 1.4 file.java
```

[17] „Ein wahrer “Schandfleck” “der” Objektorientierung ist die Art, wie Java um “Zusicherungen” erweitert wurde: das `assert`-Schlüsselwort wirft uns . . . weit . . . zurück.“ (↪ [Jähnichen/Herrmann02] S. 272.)

```
>java -enableassertions file
```

Die Parameter `enableassertions` bzw. `disableassertions` können auch in Kurzschreibweise `ea` bzw. `da` angebegen werden.

Die flogende Klasse `Foo` enthält als Beispiel in ihrem *Setter* die Zusicherung, dass der Wert ihrer Instanzvariablen `slot` stets größer null ist.

Klasse `Foo`

```
/**
 *  Zusicherung einer Wertgrenze
 *
 *@since      9-Jan-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */

package de.fhnon.wert;

public class Foo
{
   private int slot;


   public int getSlot()
   {
      return slot;
   }


   public void setSlot(int slot)
   {
      assert slot >= 0;
      this.slot = slot;

   }


   public static void main(String[] args)
   {
```

```
      Foo myObject = new Foo();
      myObject.setSlot(7);
      System.out.println(
            "Slot-Wert ist: " +
            myObject.getSlot());

      myObject.setSlot(-1);
      System.out.println(
            "Slot-Wert ist: " +
            myObject.getSlot());

   }
}
```

Protokolldatei `Foo.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac -source 1.4 de/fhnon/wert/Foo.java

C:\bonin\anwd\code>java -enableassertions de.fhnon.wert.Foo
Slot-Wert ist: 7
java.lang.AssertionError
at de.fhnon.wert.Foo.setSlot(Foo.java:24)
at de.fhnon.wert.Foo.main(Foo.java:37)
Exception in thread "main"
C:\bonin\anwd\code>java -disableassertions de.fhnon.wert.Foo
Slot-Wert ist: 7
Slot-Wert ist: -1

C:\bonin\anwd\code>javac de/fhnon/wert/Foo.java
javac de/fhnon/wert/Foo.java
de/fhnon/wert/Foo.java:24: warning: as of release 1.4,
  assert is a keyword, and may not be used as an identifier
      assert slot >= 0;
      ^
de/fhnon/wert/Foo.java:24: ';' expected
      assert slot >= 0;
                  ^
```

```
de/fhnon/wert/Foo.java:24: cannot resolve symbol
symbol  : class assert
location: class de.fhnon.wert.Foo
      assert slot >= 0;
      ^
de/fhnon/wert/Foo.java:24:
  slot is already defined in setSlot(int)
      assert slot >= 0;
             ^
3 errors
1 warning

C:\bonin\anwd\code>
```

7.9 Applikation mit großem Speicherbedarf

Als Beispiel für eine Applikation mit einem großen Arbeitsspeicherbedarf dient die Erzeugung eines Bildes im Format `JPEG`[18] (*Joint Photographics Expert Group*) mit einer Auflösung von 6000 x 6000 Pixel. Um die Applikation ausführen zu können werden die *HotSpot Memory Options* genutzt. Die Parameter `-Xms` und `-Xmx` stellen die *Heap Allocation* auf die benötigten Werte ein. Im Beispiel werden 512 MB allokiert:

Heap Allocation

```
>java -Xms512m -Xmx512m de.fhnon.graphic.MyImgStore
```

`-XmsSize` setzt den initialen Java^{TM}-Heap-Wert und `-XmxSize` den maximalen Java^{TM}-Heap-Wert beim Start der *Java Virtual Maschine* (JVM).

Klasse `MyImgStore`

```
/**
 *  Erzeugung einer JPEG-Graphik;
 *
 *@since      16-Jan-2003
 *@version    1.1
 *@author     Hinrich Bonin
 */
package de.fhnon.graphic;
```

[18] Web-Site ↪ `http://www.jpeg.org/`, Zugriff 16-Jan-2003

```
import java.awt.*;
import java.util.Random;

public class MyImgStore extends ImgJpegStore
{
   public static void main(String[] args)
   {
      try
      {
         MyImgStore mis = new MyImgStore();
         mis.store(
               4096,
               4096,
               "./de/fhnon/graphic/myPicture.jpg");
      } catch (Exception ex)
      {
         System.out.println(ex.getMessage());
         System.exit(1);
      }
      System.out.println("Image stored.");
      System.exit(0);
   }


   public void myPaintFunction(
         Graphics2D g,
         int width,
         int height,
         String imgFilename)
   {
      Random generator = new Random();
      String mySymbol = "+";

      g.setFont(new Font("Courier", Font.PLAIN, 10));
      FontMetrics fm =
            g.getFontMetrics(g.getFont());

      /*
       *  Damit zwei Zeilen s
       *  näher zusammenrücken
       */
      int heightStep = fm.getHeight() / 5;
```

```
      int widthStep = fm.stringWidth(mySymbol);

      g.setColor(Color.red);

      for (int i = 0; i < height;
            i = i + heightStep)
      {
         String s = "";

         for (int j = 0; j < width;
               j = j + widthStep)
         {
            if (generator.nextInt(2) == 0)
            {
               s = s + " ";
            } else
            {
               s = s + mySymbol;
            }
         }
         g.drawString(s, 0, i);
      }
   }
}
```

Klasse `ImgJpegStore`

```
/**
 *  Erzeugung einer JPEG-Graphik;
 *
 *@since      16-Jan-2003
 *@version    1.0
 *@author     Hinrich Bonin
 */

package de.fhnon.graphic;

import java.io.*;
import java.awt.*;
import java.awt.image.*;
import com.sun.image.codec.jpeg.*;
```

```
public abstract class ImgJpegStore
{
   public abstract void myPaintFunction(
         Graphics2D g,
         int width,
         int height,
         String imgFilename);


   public void store(
         int width,
         int height,
         String imgFilename)
          throws Exception
   {
      BufferedImage img =
            new BufferedImage(width, height,
            BufferedImage.TYPE_INT_RGB);
      myPaintFunction(
            img.createGraphics(),
            width, height,
            imgFilename);
      try
      {
         FileOutputStream out =
               new FileOutputStream(
               new File(imgFilename));
         JPEGImageEncoder enc =
               JPEGCodec.createJPEGEncoder(out);
         JPEGEncodeParam prm =
               enc.getDefaultJPEGEncodeParam(img);
         prm.setQuality(1.0f, false);
         enc.setJPEGEncodeParam(prm);
         enc.encode(img);
      } catch (Exception e)
      {
         throw new Exception(
               "\nError: Image storing to '" +
               imgFilename + "' failed: " +
               e.getMessage());
      }
```

```
    }
}
```

Protokolldatei `MyImgStore.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

c:\bonin\anwd\code>javac de/fhnon/graphic/*.java

c:\bonin\anwd\code>java -Xms512m -Xmx512m
  de.fhnon.graphic.MyImgStore
Image stored.

c:\bonin\anwd\code>java de.fhnon.graphic.MyImgStore
java.lang.OutOfMemoryError
Exception in thread "main"
c:\bonin\anwd\code>
```

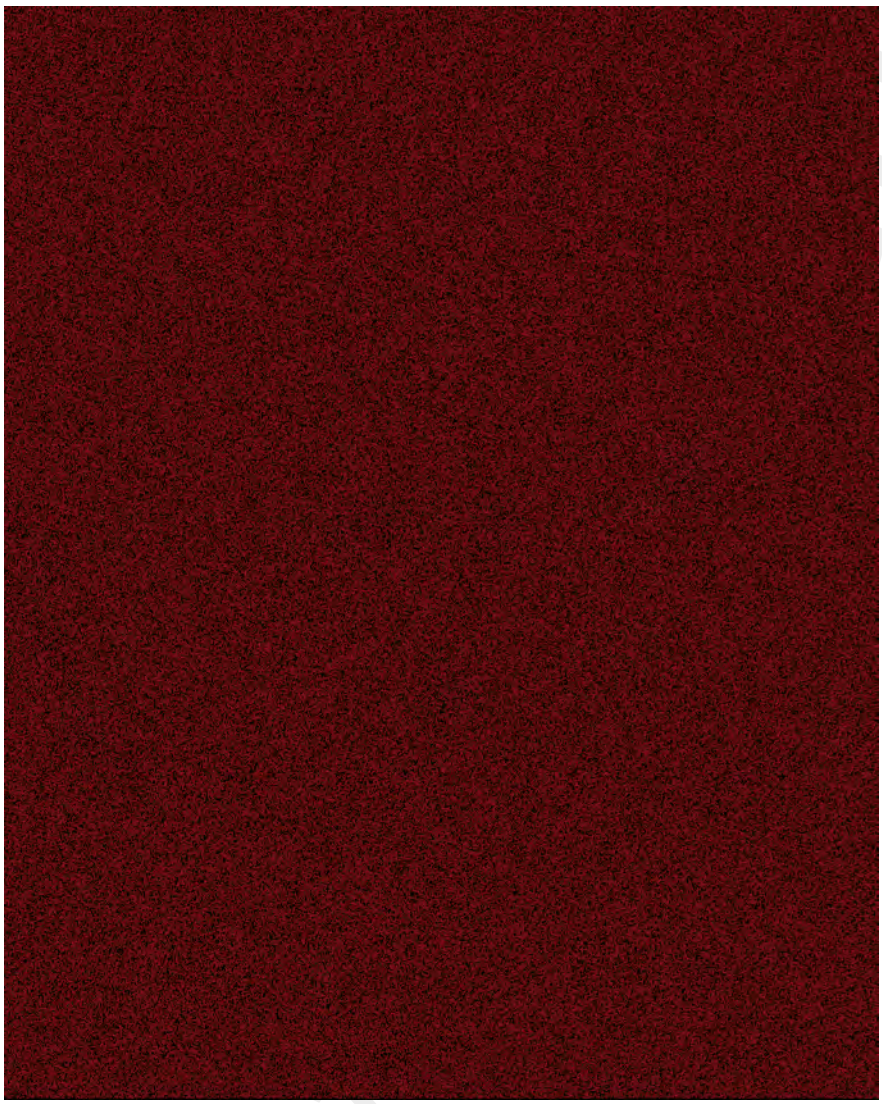
7.10 Verteilte Objekte

Wie funktioniert ein *Distributed Object System*? Mit der Erläuterung dieser Frage befassen sich die beiden Abschnitte:

1. Beispiel `Stub`[19] & `Skeleton`[20]
   ↪ Abschnitt 7.10.1 S. 255
2. Remote Method Invocation Protocol (RMI)
   ↪ Abschnitt 7.10.2 S. 264

Im ersten Schritt wird das grundlegende Prinzip mit einem eigen, primitiven System erläutert. Im zweiten Schritt wird das Prinzip anhand eines RMI-Beispiels vertieft.

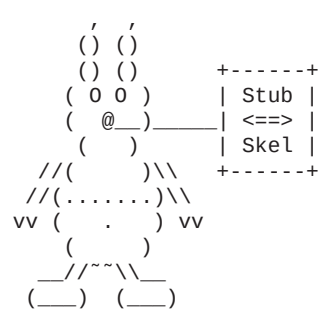
[19] Deutsch ≈ Kontrollabschnitt, (Baum-)Stumpf
[20] Deutsch ≈ Skelett, Rohbau, Rahmen



Legende:
Quellcode der Klasse `MyImgStore` ↪ S. 249 und der Klasse `ImgJpegStore` ↪ S. 251

Abbildung 7.19: JPEG-Bildbeispiel

7.10.1 Beispiel `Stub` & `Skeleton`



Das *Business Object*, dass sich auf einem anderen Rechner befindet (Server), also im Hinblick auf den Objektnutzer (Client) verteilt ist, ist ein einfaches Konto. Das Interface `Konto` (↪ S. 256) spezifiziert die Eigenschaften mit den beiden Methoden `getID()` und `getUmsatz()` für beide Seiten, also für Server und Client. Die Klasse `KontoServer` (↪ S. 259) implementiert dieses Interface und zwar für die Server-Seite. Benötigt wird nun ein Mechanismus um diese Klasse für den *remote* Client verfügbar zu machen. Dazu dienen die Klasse `Konto_Stub` (↪ S. 256) auf der Client-Seite und die Klasse `Konto_Skeleton` (↪ S. 260) auf der Server-Seite. Diese beiden Klassen implemenieren das Interface `Konto` und „kennen“ damit die Eigenschaften des Kontos.

Auf der Client-Seite ist die Klasse `Konto_Stub` quasi ein Ersatz für eine Klasse `Konto`, die ja durch die serverseitige Klasse `KontoServer` abgebildet ist. Die Klasse `Konto_Stub` spezifiziert eine Netzwerkverbindung auf `Socket`-Basis; hier mit der frei gewählten Portnummer 4711. Wird beispielsweise die Methode `getUmsatz()` für eine Instanz der Ersatzklasse `Konto_Stub` aufgerufen, dann wird ein Ausgabestrom mit dem String `umsatz` (Namen der Methode) in die Socketverbindung geschickt und ein Eingabestrom aus dieser Socketverbindung als Wert zurückgegeben. Die Klasse `KontoClient` erzeugt in ihrer `main()` eine Instanz von `Konto_Stub` und wendet darauf die beiden Methoden `getID()` und `getUmsatz()` an.

Socket

Auf der Server-Seite wird die Netzwerkverbindung auf `Socket`-Basis von der Klasse `Konto_Skeleton` „bedient“. Sie umhüllt die eigentliche *Business-Object*-Klasse `KontoServer`. Dieses *Wrapping* wird über den Konstruktor `Konto_Skeleton(KontoServer mySer-`

Wrapping

ver) realisiert. In main() von KontoServerProg wird ein solches umhülltes *Business Object* erzeugt. Der Server selbst läuft als Thread. Zur Aufgabenverteilung zwischen *Stub* und *Skeleton* siehe das Klassendiagramm in Abbildung 7.20 S. 257 und auch die Tabelle 7.6 S. 272.

Interface Konto

```
/**
 *  Beispiel "Own Distributed Object Protocol"
 *
 *@author     Hinrich Bonin
 *@created    16. Dezember 2002
 *@version    1.0
 */
package de.fhnon.distributed;
/*
 *  Konto-Interface zeigt den Ansatz für ein
 *  "Business Object"
 */
public interface Konto
{
   public String getID() throws Throwable;


   public int getUmsatz() throws Throwable;
}
```

Klasse Konto_Stub

```
/**
 *  Beispiel "Own Distributed Object Protocol"
 *
 *@author     Hinrich Bonin
 *@created    16. Dezember 2002
 *@version    1.0
 */
package de.fhnon.distributed;

import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
```

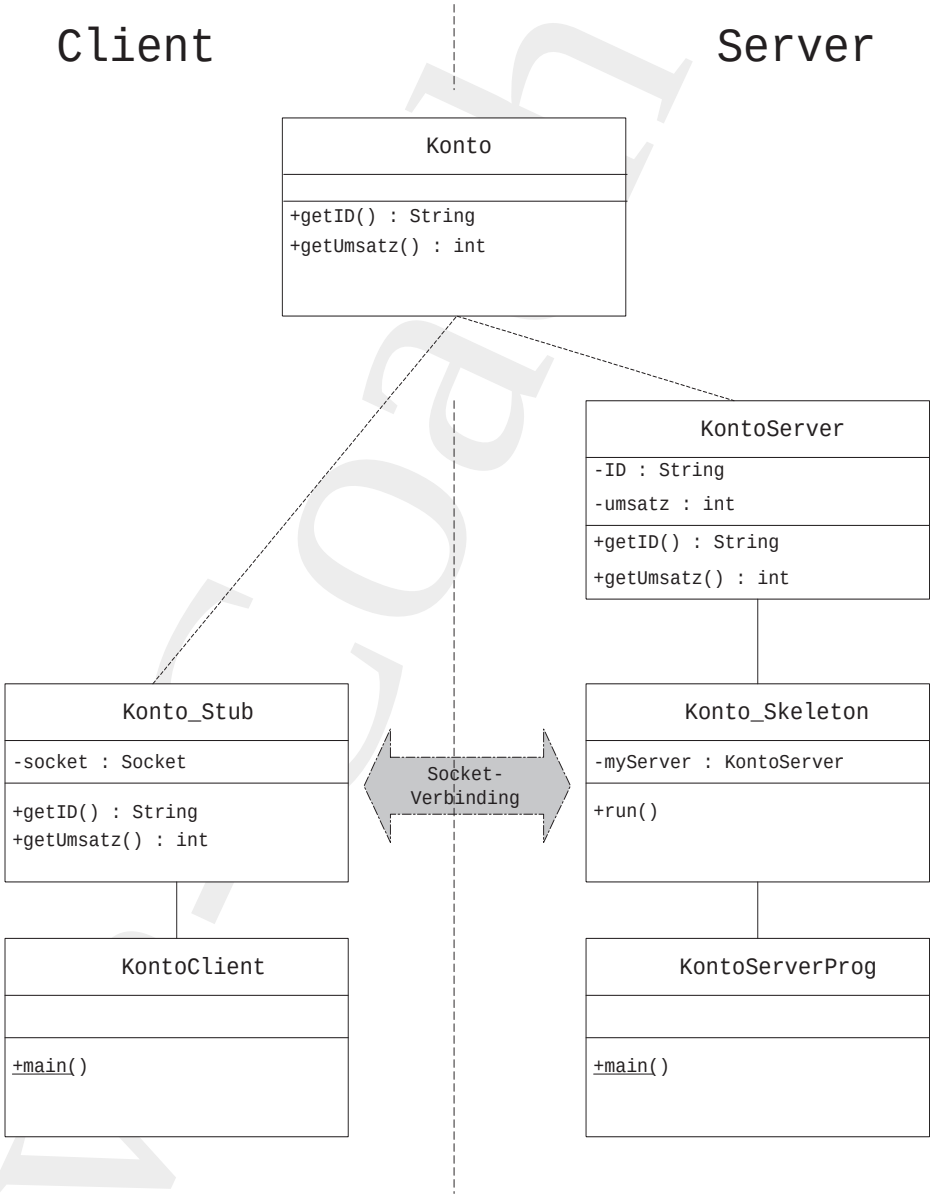


Abbildung 7.20: *Own Distributed Object Protocol* — Klassendiagramm

```
import java.net.Socket;
/*
 *  Konto_Stub implementiert das Interface Konto
 *  damit es wie ein "Business Object" auf den
 *  Client aussieht.
 *  Es leitet ein Anfrage zum Skeleton auf dem Server.
 *  Das Skeleton sendet diese an das "Business Object"
 *  auf dem Server.
 */
public class Konto_Stub implements Konto
{
   private Socket socket;


   public Konto_Stub() throws Throwable
   {
      /*
       *  Erzeugt eine Netzverbindung zum Skeleton.
       *  Hier "localhost" um auf einem
       *  Rechner zu testen;
       *  sonst IP-Adresse.
       */
      socket = new Socket("localhost", 4711);
   }


   /*
    *  Diese Methode schickt einen "Stream"
    *  mit dem Methodennamenzum Skeleton
    */
   public String getID() throws Throwable
   {
      ObjectOutputStream outStream =
            new ObjectOutputStream(
            socket.getOutputStream());
      outStream.writeObject("ID");
      outStream.flush();
      ObjectInputStream inStream =
            new ObjectInputStream(
            socket.getInputStream());
      return
            (String) inStream.readObject();
```

```
   }


   /*
    *  Diese Methode schickt einen "Stream"
    *  mit dem Methodennamenzum Skeleton
    */
   public int getUmsatz() throws Throwable
   {
      ObjectOutputStream outStream =
            new ObjectOutputStream(
            socket.getOutputStream());
      outStream.writeObject("umsatz");
      outStream.flush();
      ObjectInputStream inStream =
            new ObjectInputStream(
            socket.getInputStream());
      return
            inStream.readInt();
   }
}
```

Klasse `KontoServer`

```
/**
 *  Beispiel "Own Distributed Object Protocol"
 *
 *@author     Hinrich Bonin
 *@created    16. Dezember 2002
 *@version    1.0
 */

package de.fhnon.distributed;
/*
 *  KontoServer implementiert die "Business Logic"
 *  und den "State" für ein Konto.
 */
public class KontoServer implements Konto
{
   private String ID;
   private int umsatz;
```

```
   public KontoServer(String ID, int umsatz)
   {
      this.ID = ID;
      this.umsatz = umsatz;
   }


   public String getID()
   {
      return ID;
   }


   public int getUmsatz()
   {
      return umsatz;
   }
}
```

Klasse `Konto_Skeleton`

```
/**
 *  Beispiel "Own Distributed Object Protocol"
 *
 *@author     Hinrich Bonin
 *@created    16. Dezember 2002
 *@version    1.0
 */
package de.fhnon.distributed;

import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;
import java.net.ServerSocket;

public class Konto_Skeleton extends Thread
{
   private KontoServer myServer;


   public Konto_Skeleton(KontoServer myServer)
```

```
{
   /*
    *  Erzeugt eine Referenz zum
    *  "Business Object" das vom Skeleton
    *  umhüllt wird
    */
   this.myServer = myServer;
}


public void run()
{
   try
   {
      ServerSocket serverSocket =
            new ServerSocket(4711);
      /*
       *  Wartet auf eine Socket-Verbindung
       */
      Socket socket = serverSocket.accept();

      while (socket != null)
      {
         ObjectInputStream inStream =
               new ObjectInputStream(
               socket.getInputStream());
         String method =
               (String) inStream.readObject();
         if (method.equals("ID"))
         {
            String id = myServer.getID();
            ObjectOutputStream outStream =
                  new ObjectOutputStream(
                  socket.getOutputStream());
            outStream.writeObject(id);
            outStream.flush();
         } else if (method.equals("umsatz"))
         {
            int umsatz = myServer.getUmsatz();
            ObjectOutputStream outStream =
                  new ObjectOutputStream(
                  socket.getOutputStream());
```

```
                outStream.writeInt(umsatz);
                outStream.flush();
             }
          }
       } catch (Throwable t)
       {
          t.printStackTrace();
          System.exit(1);
       }
    }
}
```

Klasse `KontoClient`

```
/**
 *  Beispiel "Own Distributed Object Protocol"
 *
 *@author     Hinrich Bonin
 *@created    16. Dezember 2002
 *@version    1.0
 */
package de.fhnon.distributed;

public class KontoClient
{

   public static void main(String[] args)
   {
      try
      {
         /*
          *  Erzeugt als Typ Konto eine
          *  Instanz vom Stellvertreter
          */
         Konto konto = new Konto_Stub();
         System.out.println(
               konto.getID() +
               " = " +
               konto.getUmsatz() +
               " EUR");
      } catch (Throwable t)
```

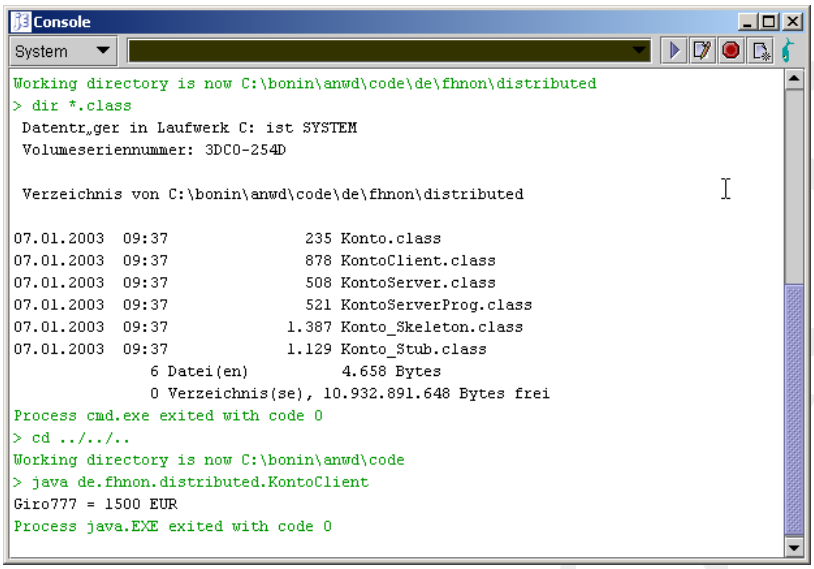
```
      {
         t.printStackTrace();
      }
   }
}
```

Klasse `KontoServerProg`

```
/**
 *  Beispiel "Own Distributed Object Protocol"
 *
 *@author     Hinrich Bonin
 *@created    16. Dezember 2002
 *@version    1.0
 */
package de.fhnon.distributed;

public class KontoServerProg
{

   public static void main(String[] args)
   {
      Konto_Skeleton skel =
            new Konto_Skeleton(
            new KontoServer("Giro777", 1500));
      skel.run();
   }
}
```

Protokolldatei `KontoServer.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/distributed/*.java
```



Legende:
Beispiel *Own Distributed Object Protocol*
Quellecode `KontoClient`↪ S. 262

Abbildung 7.21: Applikation `KontoClient`

```
C:\bonin\anwd\code>java de.fhnon.distributed.KontoServerProg

java.net.SocketException:
  Connection reset by peer:
    JVM_recv in socket input stream read

C:\bonin\anwd\code>
```

7.10.2 Beispiel `RMI`

RPC

Wenn Objekte auf mehreren Rechnern zusammen ein Anwendungssystem bilden, dann muß eine Form von Datenaustausch zwischen ihnen möglich sein. Ein solcher Datenaustausch wird häufig auf der Basis eines *Remote Procedure Call* (RPC) Mechanismus abgebildet. Das *Java Remote Method Invocation Protocol* (RMI) ist ein solches RPC-basiertes Protokoll. RMI ermöglicht einem Objekt eines Client-Systems

vorgegebene Methoden auf einem Server-System genauso aufzurufen als wären es lokale Methoden. RMI löst diese Kommunikation in einer vereinfachten Form des Standards *Common Object Request Broker Architecture* (CORBA). Im Gegensatz zu CORBA setzt RMI eine homogene Welt, also Java-Clients und Java-Server voraus.[21] Man kann sich daher RMI annähernd als ein „pure-Java-CORBA“[22] vorstellen. Das einfachere RMI ist CORBA vorzuziehen, wenn gewährleistet ist, daß es nur Java-Objekte gibt. Bei Mehrsprachigkeit im System ist CORBA erforderlich.

CORBA

RMI ist ein Modell der verteilten Objekte, das allgemein bekannte Lösungen in eine durchgängige Java^{TM} Syntax und Semantik einbaut. Dabei kombiniert RMI Lösungen von *Modula-3 Network Objects System* und von *Spring's Subcontract* (↪ [SunRMI98]). In diesem Modell ist ein *Remote*-Objekt ein Objekt, dessen Methoden aus einer anderen *Java Virtual Maschine* (JVM) aufgerufen werden können. Diese andere JVM läuft üblicherweise auf einem anderen Rechner im Netz. Ein *Remote*-Objekt wird durch ein oder mehrere *Remote Interfaces* beschrieben. Ein solches Interface deklariert die Methoden des *Remote*-Objektes. Der Client referenziert das *Remote*-Objekt anhand einer RMI-URL[23]-Angabe. Diese hat folgende Form:

RMI-URL

```
rmi://hostname[:port]/object
```

Dabei hat der *Default*-Port die Nummer 1099.

Wenn der Server ein Objekt für einen RMI-URL-Zugriff (*lookup*) verfügbar macht, dann muß er eine Objektinstanz an einen Objektnamen binden. Der Objektname ist ein vorgegebener `String`. Die Klassenmethode `lookup(String url)` der Klasse `java.rmi.Naming` verbindet letztlich den Client mit dem entsprechenden Serverobjekt. Dazu sind die Klassen `server_Stub.class` und `server_Skel.class` auf der Serverseite erforderlich. Diese zusätzlichen Kommunikationsklassen werden mit Hilfe des Programms `rmic` erzeugt.

[21] Präziser formuliert: RMI verbindet Systeme, die das *Standard Java Native Method Interface* (JNI) benutzen. Dies könnten prinzipiell auch Systeme in einer anderen Sprache sein.

[22] ↪ [Vanderburg97] p. 525

[23] *Uniform Resource Locator*

rmic

`rmic` ServerClassName

Das Programm `rmic` erzeugt und compiliert die sogenannten *Stub*- und *Skeleton*-Klassen. In der Stub-Klasse sind die Remote-Methoden implementiert. In dem Beispiel „Bank“ sind es die Methoden `abheben()`, `einzahlen()` usw. (↪ Abschnitt 85 S. 272). In der Skeleton-Klasse sind es die Methoden `getOperations()` und `dispatch()`. Die Stub-Klasse dient als eine Art Dummy-Referenz für das Client-System, während die Skeleton-Klasse das eigentliche Server-System verwaltet. Tabelle 7.6 S. 272 skizziert die Zusammenarbeit zwischen Client, Stub, Skeleton und Server.

Bei diesem *Remote Object Model* hält der Server Objekte vor, die der Client „aus der Ferne“ benutzen kann. Der Client wendet eine Methode auf ein entferntes Objekt genauso an, als ob das Remote-Objekt sein lokales Objekt wäre, das in seiner *Java Virtual Maschine* existiert.

```
... MyClientFooClass
⋮
localInstance.lokalMethod(myArgument);
⋮
remoteInstance.remoteMethod(myArgument);
⋮
```

Der RMI-Mechanismus verbirgt die tiefer liegenden Transportmechanismen für das Übermitteln des Methodennamens, der Methodenargumente und des Rückgabewertes. Argumente und Rückgabewert können komplexe Objekte sein, und nicht nur einfache Zeichenketten. Für die Übermittlung müssen sie allerdings serialisiert werden. Daher kommen für RMI alle `Serializable`-Objekte in Betracht (↪ Abschnitt 7.3 S. 182).

Für die Entwicklung einer RMI-Anwendung sind folgende Schritte erforderlich:

Client

1. Festlegen der Methoden, die auf das Remote-Objekt angewendet werden sollen.
   ↪ Definieren eines Subinterfaces von `java.rmi.Remote`. Dieses Interface definiert die exportierbaren Methoden, die das Remote-Objekt implementiert, das heißt, die Methoden, die der Server im-

plementiert und der Client aufrufen kann.

2. Definieren einer Subklasse von `java.rmi.server.UnicastRemoteObject` **Server**
   ↪ Sie implementiert das `Remote`-Interface.

3. Schreiben der Server-Applikation — Erzeugen einer Instanz des Remote-Objekts und „Exportieren“ dieser Instanz, das heißt, Verfügbarmachen für die Nutzung durch den Client. **Server**
   ↪ Registrieren des Objektes anhand seines Namens mit einem Registrierungsservice. Üblicherweise erfolgt diese Registrierung mittels der Klasse `java.rmi.Naming` und dem Programm `rmiregistry` (↪ übernächsten Punkt).

4. Erzeugen von Stub und Skeleton mit dem Programm `rmic` aus der compilierten Server-Klasse. **Server**

5. Registrierung **Server**

   Windosw-NT-Plattform: `start rmiregistry [port]`
   UNIX-Plattform: `rmiregistry [port] &`

6. Schreiben der Client-Applikation **Client**

7. Compilieren und Anwenden der Client-Applikation

Für das Beispiel „Bank“ sehen die Schritte wie folgt aus:

1. Methoden auf dem Bank-Server:

```
public interface RemoteBank extends Remote {
  public void einzahlen
    (String name, String passwort, Euro geld)
    throws RemoteException, BankingException;
  public Euro abheben
    (String name, String passwort, int betrag)
    throws RemoteException, BankingException;
  public int getStand
    (String name, String passwort)
    throws RemoteException, BankingException;
  public Vector getKontoBewegungen
    (String name, String passwort)
```

```
      throws RemoteException, BankingException;
    public void eroeffnenKonto
      (String name, String passwort)
      throws RemoteException, BankingException;
    public Euro aufloesenKonto
      (String name, String passwort)
      throws RemoteException, BankingException;
  }
```

2. Definieren der Klasse `RemoteBankServer` als Unterklasse von `java.rmi.server.UnicastRemoteObject`. Sie implementiert das Interface `RemoteBank`

```
public class RemoteBankServer extends UnicastRemoteObject
  implements RemoteBank {
  class Konto {...}

  public RemoteBankServer() throws RemoteException {
    super();
  }

  public void einzahlen(
    String name, String passwort, Euro geld)
    throws RemoteException, BankingException {...}

  public Euro abheben(
    String name, String passwort, int betrag)
    throws RemoteException, BankingException {...}

  public int getStand(String name, String passwort)
    throws RemoteException, BankingException {...}

  public Vector getKontoBewegungen(
    String name, String passwort)
    throws RemoteException, BankingException {...}

  public synchronized void eroeffnenKonto(
   String name, String passwort)
   throws RemoteException, BankingException {...}

  public Konto pruefen(String name, String passwort)
    throws BankingException {...}
```

```
  public synchronized Euro aufloesenKonto(
    String name, String passwort)
    throws RemoteException, BankingException {...}
...
}
```

3. Schreiben von `RemoteBankServer` — Erzeugen einer Instanz `bank` des Remote-Objekts `RemoteBankServer` und „Exportieren“ dieser Instanz mittels `Naming.rebind(name, bank)`

```
  public static void main(String argv[]) {
    try {
      RemoteBankServer bank = new RemoteBankServer();
      String name
        = System.getProperty("bankname", "BoninRemote");
      Naming.rebind(name, bank);
      System.out.println(name +
        " ist eroeffnet und bereit fuer Buchungen.");
    }
```

Die Klassenmethode `getProperty(String key, String default)` der Klasse `java.lang.System` sucht in der Systemeigenschaftsliste nach dem Wert von `key`. Wird keiner gefunden, dann ist `default` der Rückgabewert. Beim Aufruf einer Applikation kann ein Eintrag in diese Systemeigenschaftsliste mit Hilfe der Option „`-D`“ erfolgen.

```
java -Dkey1=wert1 -Dkey2=wert2 ... javaClass
```

Zum Beispiel:

```
java -Dbank="rmi://myServer:1111/myRemoteObject"
Bank$Client ...
```

4. Erzeugen von `RemoteBankServer_Stub` und `RemoteBankServer_Skel` mit dem Programm `rmic` aus `RemoteBankServer`.

```
javac RemoteBankServer.java
rmic RemoteBankServer
```

```
public final synchronized class RemoteBankServer_Stub
  extends java.rmi.server.RemoteStub
  implements Bank$RemoteBank, java.rmi.Remote {
  // Feld(er)
  private static java.rmi.server.Operation[] operations;
  private static final long interfaceHash;
  // Konstruktor(en)
  public RemoteBankServer_Stub();
  public RemoteBankServer_Stub(java.rmi.server.RemoteRef);
  // Methode(n)
  public Bank$Euro abheben(java.lang.String, java.lang.String, int)
    throws Bank$BankingException, java.rmi.RemoteException;
  public Bank$Euro aufloesenKonto(java.lang.String, java.lang.String)
    throws Bank$BankingException, java.rmi.RemoteException;
  public void einzahlen(java.lang.String, java.lang.String, Bank$Euro)
    throws Bank$BankingException, java.rmi.RemoteException;
  public void eroeffnenKonto(java.lang.String, java.lang.String)
    throws Bank$BankingException, java.rmi.RemoteException;
  public java.util.Vector getKontoBewegungen
    (java.lang.String, java.lang.String)
     throws Bank$BankingException, java.rmi.RemoteException;
  public int getStand(java.lang.String, java.lang.String)
    throws Bank$BankingException, java.rmi.RemoteException;
}
```

```
public final synchronized class RemoteBankServer_Skel
  extends java.lang.Object
  implements java.rmi.server.Skeleton {
  // Feld(er)
  private static java.rmi.server.Operation[] operations;
  private static final long interfaceHash;
  // Konstruktor(en)
  public RemoteBankServer_Skel();
  // Methode(n)
  public java.rmi.server.Operation[] getOperations();
  public void dispatch
   (java.rmi.Remote, java.rmi.server.RemoteCall, int, long)
   throws java.rmi.RemoteException, java.lang.Exception;
}
```

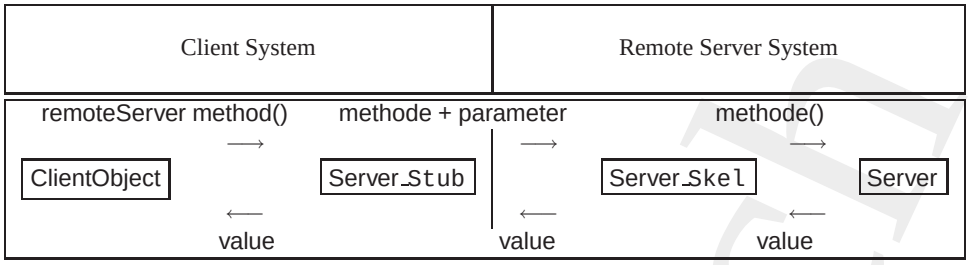
5. Registrierung mit Hilfe des Programms `rmiregistry` auf einer UNIX-Plattform bei Nutzung des *Default-Ports* 1099 und Starten des Servers.

```
rmiregistry&
java RemoteBankServer
BoninRemote ist eroeffnet und bereit fuer Buchungen.
```

6. Schreiben der Client-Applikation `Bank$Client`

```
public static class Client {
  public static void main(String argv[]) {
  try {
    System.setSecurityManager(new RMISecurityManager());
    String url = System.getProperty(
        "bank", "rmi:///BoninRemote");
    RemoteBank bank = (RemoteBank) Naming.lookup(url);

    if (aktion.equals("einzahlen")) {
      Euro geld = new Euro(
          Integer.parseInt(argv[3]));
      bank.einzahlen(argv[1], argv[2], geld);
      System.out.println("Eingezahlt: " +
        geld.betrag + " Euro");
    }
    else if (aktion.equals("abheben")) {
      Euro geld = bank.abheben(argv[1], argv[2],
        Integer.parseInt(argv[3]));
      System.out.println("Abgehoben: " +
        geld.betrag + " Euro");
    }
    ...

  }
  catch (RemoteException e)  {...}
  catch (BankingException e) {...}
  catch (Exception e)        {...}
  ...
```

7. Compilieren der Datei `Bank.java` und Anwenden der Applikation, das heißt, Aufruf von `main()` in der Klasse `Bank$Client`.



Legende:

`Stub`	≡	lokaler Stellvertreter
`Skel`	≡	Skeleton, Rahmen

Das RMI-System gliedert sich in die *Layer* — Näheres ↪ [SunRMI98]:

1. *Stub/Skeleton Layer* — eine Proxy-Funktion auf der Client-Steite (Stub)
2. *Remote Reference Layer* — Verhalten bei einem einzelnen Objekt oder bei replizierten Objekten
3. *Transport Layer* — Verbindungsmanagement und Remote Objekt Verfolgung

Tabelle 7.6: RMI: *Stub/Skeleton, Remote-Reference* und *Transport*

Die Klasse `Bank.class` dient nur als ein Sammelbehälter für das Interface `RemoteBank` und die Klassen `Euro`, `BankingException` und `Client`.[24]

```
javac Bank.java
java Bank$Client eroeffnen otto kh234g
Konto eroeffnet!
...
```

Clientklasse `Bank`

```
/**
 *  RMI-Client-Beispiel Idee von David Flanagan; Java
 *  Examples in a Nutshell, 1997, p. 294 Quellcode stark
 *  modifiziert.
 *
 *@author     Hinrich Bonin
```

[24] Diese sind mit dem Modifikator `static` versehen, um als Toplevel-Interface bzw. Toplevel-Klassen nutzbar zu sein.

```
 *@version    1.0
 *@since      13-Jun-1998
 */
import java.rmi.*;
import java.util.Vector;

/*
 *  Bank enthält alle Interfaces und Klassen (top-level)
 */
public class Bank
{
   /*
    *  Methoden auf dem Bankserver einzahlen abbheben
    *  getStand getKontoBewegungen eroeffnenKonto
    *  aufloesenKonto
    */
   public interface RemoteBank extends Remote
   {
      public void einzahlen
            (String name, String passwort, Euro geld)
             throws RemoteException, BankingException;


      public Euro abheben
            (String name, String passwort, int betrag)
             throws RemoteException, BankingException;


      public int getStand
            (String name, String passwort)
             throws RemoteException, BankingException;


      public Vector getKontoBewegungen
            (String name, String passwort)
             throws RemoteException, BankingException;


      public void eroeffnenKonto
            (String name, String passwort)
             throws RemoteException, BankingException;


      public Euro aufloesenKonto
            (String name, String passwort)
             throws RemoteException, BankingException;
```

```
}


/*
 *  Einfache Klasse die Geld repraesentiert
 */
public static class Euro implements
      java.io.Serializable
{
   public int betrag;


   public Euro(int betrag)
   {
      this.betrag = betrag;
   }
}


/*
 *  Bankspezifische Ausnahmen
 */
public static class BankingException
       extends Exception
{
   public BankingException(String nachricht)
   {
      super(nachricht);
   }
}


/*
 *  Bank$Client kommuniziert mit dem RMI-Server
 */
public static class Client
{
   public static void main(String argv[])
   {
      try
      {
         /*
          *  Sicherheit gegen untrusted stub code
          *  über das Netz
          */
         System.setSecurityManager(
```

```
      new RMISecurityManager());
/*
 *  Default-Wert BoninRemote
 */
String url = System.getProperty(
      "bank", "rmi:///BoninRemote");
/*
 *  Naming Objekt kontaktet rmiregistry
 */
RemoteBank bank =
      (RemoteBank) Naming.lookup(url);

String aktion = argv[0].toLowerCase();

if (aktion.equals("einzahlen"))
{
   Euro geld =
         new Euro(Integer.parseInt(argv[3]));
   bank.einzahlen(argv[1], argv[2], geld);
   System.out.println("Eingezahlt: " +
         geld.betrag + " Euro");
} else if (aktion.equals("abheben"))
{
   Euro geld = bank.abheben(argv[1], argv[2],
         Integer.parseInt(argv[3]));
   System.out.println("Abgehoben: " +
         geld.betrag + " Euro");
} else if (aktion.equals("stand"))
{
   System.out.println("Kontostand : " +
         bank.getStand(argv[1], argv[2])
          + " Euro");
} else if (aktion.equals("bewegungen"))
{
   Vector bewegungen = bank.getKontoBewegungen(
         argv[1], argv[2]);
   for (int i = 0; i < bewegungen.size(); i++)
   {
      System.out.println(bewegungen.elementAt(i));
   }
} else if (aktion.equals("eroeffnen"))
{
   bank.eroeffnenKonto(argv[1], argv[2]);
   System.out.println("Konto eroeffnet!");
} else if (aktion.equals("aufloesen"))
{
```

```
                Euro geld =
                      bank.aufloesenKonto(argv[1], argv[2]);
                System.out.println(geld.betrag +
                      " Euro erhalten Sie noch ausgezahlt!");
             } else
             {
                System.out.println("Unbekannte Aktion!");
             }
          } catch (RemoteException e)
          {
             System.err.println(e);
          } catch (BankingException e)
          {
             System.err.println(e.getMessage());
          } catch (Exception e)
          {
             System.err.println(e);
             System.err.println(
                   "Usage: java [-Dbank=<url>] Bank$Client " +
                   "<aktion> <name> <passwort> [<betrag>]");
             System.err.println(
                   "wobei <aktion> einer der folgenden Wert ist: " +
                   "\nEinzahlen, Abheben, Stand," +
                   " Bewegungen, Eroeffnen, Aufloesen");
          }
       }
    }
}
```

Serverklasse `RemoteBankServer`

```
/**
 *  RMI-Server-Beispiel Quellidee von David Flanagan; Java
 *  Examples in a Nutshell, 1997, p. 297 Quellcode
 *  modifiziert.
 *
 *@author     Hinrich Bonin
 *@version    1.0
 *@since      13-Jun-1998
 */
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import Bank.*;
```

```
public class RemoteBankServer
       extends UnicastRemoteObject
       implements RemoteBank
{
   class Konto
   {
      int stand;
      String passwort;
      Vector bewegungen = new Vector();


      Konto(String passwort)
      {
         this.passwort = passwort;
         bewegungen.addElement(
               "Kontoeroeffnung am: " +
               new Date());
      }
   }


   Hashtable kontos = new Hashtable();


   public RemoteBankServer()
          throws RemoteException
   {
      super();
   }


   public void einzahlen(
         String name, String passwort, Euro geld)
          throws RemoteException, BankingException
   {
      Konto myK = pruefen(name, passwort);
      synchronized (myK)
      {
         myK.stand += geld.betrag;
         myK.bewegungen.addElement("Eingezahlt: " +
               geld.betrag +
               " am " + new Date());
      }
   }
```

```
public Euro abheben(
      String name, String passwort, int betrag)
       throws RemoteException, BankingException
{
   Konto myK = pruefen(name, passwort);
   synchronized (myK)
   {
      if (myK.stand < betrag)
      {
         throw new BankingException("Keine Deckung!");
      }
      myK.stand -= betrag;
      myK.bewegungen.addElement("Abgehoben: " +
            betrag + " am " + new Date());
      return new Euro(betrag);
   }
}


public int getStand(String name, String passwort)
       throws RemoteException, BankingException
{
   Konto myK = pruefen(name, passwort);
   synchronized (myK)
   {
      return myK.stand;
   }
}


public Vector getKontoBewegungen(
      String name, String passwort)
       throws RemoteException, BankingException
{
   Konto myK = pruefen(name, passwort);
   synchronized (myK)
   {
      return myK.bewegungen;
   }
}


public synchronized void eroeffnenKonto(
      String name, String passwort)
       throws RemoteException, BankingException
{
```

```
      if (kontos.get(name) != null)
      {
         throw new BankingException(
               "Konto gibt es schon!");
      }
      Konto myK = new Konto(passwort);
      kontos.put(name, myK);
   }


   public Konto pruefen(String name, String passwort)
          throws BankingException
   {
      synchronized (kontos)
      {
         Konto myK = (Konto) kontos.get(name);
         if (myK == null)
         {
            throw new BankingException(
                  "Kein solches Konto!");
         }
         if (!passwort.equals(myK.passwort))
         {
            throw new BankingException(
                  "Falches Passwort!");
         }
         return myK;
      }
   }


   public synchronized Euro aufloesenKonto(
         String name, String passwort)
          throws RemoteException, BankingException
   {
      Konto myK;
      myK = pruefen(name, passwort);
      kontos.remove(name);
      synchronized (myK)
      {
         int wert = myK.stand;
         myK.stand = 0;
         return new Euro(wert);
      }
   }
```

```
   public static void main(String argv[])
   {
      try
      {
         RemoteBankServer bank
                = new RemoteBankServer();
         String name
                = System.getProperty(
               "bankname", "BoninRemote");
         Naming.rebind(name, bank);
         System.out.println(
               name +
               " ist eroeffnet und" +
               " bereit fuer Buchungen.");
      } catch (Exception e)
      {
         System.err.println(e);
         System.err.println(
               "Usage: java [-Dbankname=<name>]" +
               " RemoteBankServer");
         System.exit(1);
      }
   }
}
```

Protokoll einer Session

```
--- Windows-NT-Rechner 193.174.33.100
>java -fullversion
java full version "JDK1.1.5k"
>javac RemoteBankServer.java
>rmic RemoteBankServer
>start rmiregistry
>java RemoteBankServer
BoninRemote ist eroeffnet und bereit fuer Buchungen.

--- UNIX-Rechner 193.174.33.106

cl3:/home/bonin/myjava:>java -fullversion
java full version "JDK 1.1.6 IBM build a116-19980529" (JIT: jitc)
>javac Bank.java
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client doof otto geheim
```

```
Unbekannte Aktion!
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client eroeffnen otto geheim
Konto eroeffnet!
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client einzahlen otto geheim 100
Eingezahlt: 100 Euro
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client einzahlen otto geheim 200
Eingezahlt: 200 Euro
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client stand otto geheim
Kontostand : 300 Euro
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client abheben otto geheim 50
Abgehoben: 50 Euro
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client stand otto geheim
Kontostand : 250 Euro
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client bewegungen otto geheim
Kontoeroeffnung am: Sat Jun 13 14:14:13 CEST 1998
Eingezahlt: 100 am Sat Jun 13 14:15:03 CEST 1998
Eingezahlt: 200 am Sat Jun 13 14:15:14 CEST 1998
Abgehoben: 50 am Sat Jun 13 14:16:00 CEST 1998
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
Bank\$Client aufloesen otto geheim
250 Euro erhalten Sie noch ausgezahlt!
cl3:/home/bonin/myjava:>
```

7.11 XML-Daten aggregieren

Als Beispiel für die Verarbeitung von XML-Daten in Java gehen wir von folgenden Anforderungen (*Requirements*) aus:

R01 Das Programm `AggregationProg` liest eine Lieferantendatei `lief-dat` ein.

R02 Beim ordnungsgemäßen Ende von `AggregationProg` wird die Nachricht „`Alles verarbeitet!`“ ausgegeben.

R03 Die Datensätze der `lief-dat` haben ein Merkmal `m`.

R03.1 Die Datensätze mit `m = A` sind nicht bedeutsam. Sie werden übersprungen.

R03.2 Bei Datensätzen mit `m = B` wird die Nachricht „`Dubios!`“ ausgegeben.

R03.3 Bedeutsam sind die Datensätze mit `m = C`. Für sie gilt folgendes:

R03.3.1 Jeder Datensatz der `lief-dat` enthält für jeden Monat ein Monatsumsatzfeld.

R03.3.2 Für jeden der 12 Monate wird geprüft, ob der Wert im Monatsumsatzfeld numerisch ist;

R03.3.3 wenn ja, dann wird der Monatsumsatz in das entsprechende Feld der Jahrestabelle `jahr-tab` addiert.

R04 Vor Beginn der Verarbeitung wird `jahr-tab` auf den Anfangswert 0 gesetzt.

Ausgangspunkt ist der Entwurf einer XML-Struktur für `lief-dat` und `jahr-tab`. Die Klasse `Aggregation` (↪ S. 289) nutzt JDOM, eine *Open Source Libery* für die Java-optimierte Verarbeitung von XML-Daten. JDOM stellt Klassen für SAX (*Simple API for XML*) und DOM (*Document Object Model*) bereit. Die benötigten Klassen lassen sich von der Web-Homepage `http://www.jdom.org` (online 19-May-2004) herunterladen.[25]

SAX
DOM

Während das DOM von W3C (*World Wide Web Consortium*) sprachunabhängig konzipiert wurde und ursprünglich primär für die Manipulation von HTML-Dokumenten mit JavaScript genutzt wurde, ist JDOM konsequent auf die Java-Möglichkeiten hin entwickelt worden. Holzschnittartig formuliert verhält sich JDOM zu W3C's DOM wie RMI (↪ Abschnitt 7.10.2 S. 264) zu CORBA (*Common Object Request Broker Architecture*).

Für die beispielhaften Lieferantendaten `lief-dat.xml` (↪ S. 285) wird eine *Document Type Definition* entworfen. Diese DTD ist in der Datei `lief-dat.dtd` notiert (↪ S. 284). Für die die Ergebnisdatei

[25] Hinweis: Nach Durchführung des Batchlaufes `jdom-b10/build.bat` ist die Java-Umgebungsvariable `CLASSPATH` zu ergänzen, hier um: `c:/programme/jdom-b10/build/jdom.jar;`.

`jahr-tab.xml` wird ebenfalls eine *Document Type Definition* entworfen. Diese DTD ist in der Datei `jahr-tab.dtd` notiert (↪ S. 288).

Aus JDOM nutzen wir die folgende Klasse:

```
org.jdom.DocType
org.jdom.Document
org.jdom.Element
org.jdom.input.SAXBuilder
org.jdom.input.JDOMParseException
org.jdom.output.Format
org.jdom.output.XMLOutputter
```

Der jeweilige Klassenname vermittelt schon intuitiv die Funktion. Ist eine Instanz der Klasse `Document` erzeugt, dann kann davon das Root-Element selektiert werden. Davon dann wiederum die Nachfolgeknoten mit Hilfe der Methode `getChildren()`.

Beim Einlesen der XML-Lieferantendaten validieren wir den Inhalt von `lief-dat.xml` gegen ihre DTD. Gäbe es beispielsweise das nicht vorgesehene Element `<jaenner>...</jaenner>` würde es vom Parser erkannt (↪ S. 297). Das Einlesen der `lief-dat.xml` basiert auf folgender Konstruktion:

Detail XML-Input

```
SAXBuilder builder = new SAXBuilder();
builder.setValidation(true);

Document docInput = builder.build(new File(
      this.getInputXMLFile()));

Element rootInput =
      docInput.getRootElement();

lieferantenListe =
      rootInput.getChildren(
      this.getInputXMLFileChild());
```

Zur Ausgabe der XML-Datei `jahr-tab.xml` starten wir mit dem `dtdRootElement`, hier `umsatz`, und geben die dazugehörende DTD als `dtdOutputXMLFile` an, hier `jahr-tab.dtd`. Die Details zeigt der folgende Quellcodeausschnitt:

Detail XML-Output

```
Document docOutput = new Document(
      new Element(
      this.getDtdRootElement()));
DocType docType = new DocType(
      this.getDtdRootElement(),
      this.getDtdOutputXMLFile());

docOutput.setDocType(docType);

Element rootOutput = docOutput.getRootElement();
addElementeMonate(rootOutput);

XMLOutputter out = new XMLOutputter();

Format format = Format.getPrettyFormat();
out.setFormat(format);

BufferedWriter bw = new BufferedWriter(
      new FileWriter(outputXMLFile));

out.output(docOutput, bw);

bw.close();
```

Document Type Definition `lief-dat.dtd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD zur Inputvalidation von lief-dat.xml -->
<!-- Bonin   May 2004                  -->
<!ELEMENT lief-dat (lieferant+) >
<!ELEMENT lieferant (
  januar,
  februar,
  maerz,
  april,
  mai,
  juni,
  juli,
  august,
  september,
  oktober,
```

```
  november,
  dezember
  )>
  <!ATTLIST lieferant id ID #REQUIRED
                      m ( A | B | C ) #REQUIRED>
<!ELEMENT januar    (#PCDATA)>
<!ELEMENT februar   (#PCDATA)>
<!ELEMENT maerz     (#PCDATA)>
<!ELEMENT april     (#PCDATA)>
<!ELEMENT mai       (#PCDATA)>
<!ELEMENT juni      (#PCDATA)>
<!ELEMENT juli      (#PCDATA)>
<!ELEMENT august    (#PCDATA)>
<!ELEMENT september (#PCDATA)>
<!ELEMENT oktober   (#PCDATA)>
<!ELEMENT november  (#PCDATA)>
<!ELEMENT dezember  (#PCDATA)>
```

Lieferantendaten `lief-dat.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lief-dat SYSTEM "lief-dat.dtd">
<lief-dat>
        <lieferant id="OttoAG" m="A">
                <januar>10</januar>
                <februar>100</februar>
                <maerz>10</maerz>
                <april>10</april>
                <mai>100</mai>
                <juni>100</juni>
                <juli>100</juli>
                <august>100</august>
                <september>100</september>
                <oktober>200</oktober>
                <november>60</november>
                <dezember>100</dezember>
        </lieferant>
        <lieferant id="MuellerGmbH" m="C">
                <januar>100</januar>
                <februar>100</februar>
                <maerz>30</maerz>
                <april>100</april>
```

```
            <mai>10</mai>
            <juni>200</juni>
            <juli>100</juli>
            <august>45</august>
            <september>100</september>
            <oktober>67</oktober>
            <november>10</november>
            <dezember>500</dezember>
    </lieferant>
    <lieferant id="KrauseOHG" m="A">
            <januar>100</januar>
            <februar>100</februar>
            <maerz>500</maerz>
            <april>10</april>
            <mai>100</mai>
            <juni>10</juni>
            <juli>100</juli>
            <august>400</august>
            <september>100</september>
            <oktober>100</oktober>
            <november>100</november>
            <dezember>100</dezember>
    </lieferant>
    <lieferant id="SchulzeAG" m="C">
            <januar>100</januar>
            <februar>100</februar>
            <maerz>100</maerz>
            <april>10</april>
            <mai>100</mai>
            <juni>100</juni>
            <juli>unbekannt</juli>
            <august>100</august>
            <september>100</september>
            <oktober>60</oktober>
            <november>100</november>
            <dezember>800</dezember>
    </lieferant>
    <lieferant id="HausmmannKG" m="B">
            <januar>100</januar>
            <februar>100</februar>
            <maerz>100</maerz>
            <april>100</april>
```

```
            <mai>100</mai>
            <juni>100</juni>
            <juli>100</juli>
            <august>100</august>
            <september>100</september>
            <oktober>100</oktober>
            <november>100</november>
            <dezember>100</dezember>
    </lieferant>
    <lieferant id="MeyerGmbH" m="C">
            <januar>100</januar>
            <februar>100</februar>
            <maerz>100</maerz>
            <april>100</april>
            <mai>100</mai>
            <juni>100</juni>
            <juli>100</juli>
            <august>100</august>
            <september>100</september>
            <oktober>100</oktober>
            <november>100</november>
            <dezember>100</dezember>
    </lieferant>
    <lieferant id="WilhelmEG" m="B">
            <januar>100</januar>
            <februar>100</februar>
            <maerz>100</maerz>
            <april>100</april>
            <mai>100</mai>
            <juni>100</juni>
            <juli>100</juli>
            <august>100</august>
            <september>100</september>
            <oktober>100</oktober>
            <november>100</november>
            <dezember>100</dezember>
    </lieferant>
    <lieferant id="GutknechtGmbH" m="C">
            <januar>100</januar>
            <februar>100</februar>
            <maerz>100</maerz>
            <april>30</april>
```

```
                <mai>100</mai>
                <juni>100</juni>
                <juli>100</juli>
                <august>100</august>
                <september>100</september>
                <oktober>100</oktober>
                <november>100</november>
                <dezember>100</dezember>
        </lieferant>
</lief-dat>
```

Document Type Definition `jahr-tab.dtd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD zur XML-Ausgabe von jahr-tab -->
<!-- Bonin   May 2004                 -->
<!ELEMENT umsatz (
  januar,
  februar,
  maerz,
  april,
  mai,
  juni,
  juli,
  august,
  september,
  oktober,
  november,
  dezember)>
<!ELEMENT januar    (#PCDATA)>
<!ELEMENT februar   (#PCDATA)>
<!ELEMENT maerz     (#PCDATA)>
<!ELEMENT april     (#PCDATA)>
<!ELEMENT mai       (#PCDATA)>
<!ELEMENT juni      (#PCDATA)>
<!ELEMENT juli      (#PCDATA)>
<!ELEMENT august    (#PCDATA)>
<!ELEMENT september (#PCDATA)>
<!ELEMENT oktober   (#PCDATA)>
<!ELEMENT november  (#PCDATA)>
<!ELEMENT dezember  (#PCDATA)>
```

Klasse `Aggregation`

```
/**
 *  Example "Lieferantendaten" selektieren und aggregieren
 *
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.as.xmldata;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.List;

import org.jdom.DocType;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.input.JDOMParseException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;

public class Aggregation
{
   int januar = 0;
   int februar = 0;
   int maerz = 0;
   int april = 0;
   int mai = 0;
   int juni = 0;
   int juli = 0;
   int august = 0;
   int september = 0;
   int oktober = 0;
   int november = 0;
   int dezember = 0;
```

```
String inputXMLFile = new String();
String inputXMLFileChild = new String();

String outputXMLFile = new String();

String dtdRootElement = new String();
String dtdOutputXMLFile = new String();


String getInputXMLFile()
{
   return inputXMLFile;
}


String getInputXMLFileChild()
{
   return inputXMLFileChild;
}


String getOutputXMLFile()
{
   return outputXMLFile;
}


String getDtdRootElement()
{
   return dtdRootElement;
}


String getDtdOutputXMLFile()
{
   return dtdOutputXMLFile;
}


Aggregation(String inputXMLFile, String inputXMLFileChild,
      String outputXMLFile,
      String dtdRootElement, String dtdOutputXMLFile)
```

```
{
   this.inputXMLFile = inputXMLFile;
   this.inputXMLFileChild = inputXMLFileChild;
   this.outputXMLFile = outputXMLFile;
   this.dtdRootElement = dtdRootElement;
   this.dtdOutputXMLFile = dtdOutputXMLFile;
}


/**
 *  [R01]
 *
 *@return    Liste der Lieferanten
 */
List readXMLInput()
{
   /*
    *  zur Vermeidung von
    *  java.lang.NullPointerException
    *  nicht mit null initialisiert
    */
   List lieferantenListe = new ArrayList();

   try
   {
      SAXBuilder builder = new SAXBuilder();
      builder.setValidation(true);

      Document docInput = builder.build(new File(
          this.getInputXMLFile()));

      Element rootInput =
            docInput.getRootElement();

      lieferantenListe =
            rootInput.getChildren(
            this.getInputXMLFileChild());
   } catch (JDOMParseException e)
   {
      System.err.println(e);
      System.exit(1);
   } catch (Exception e)
```

```
      {

         System.err.println(e);
         System.exit(1);

      }

      return lieferantenListe;
   }


   /**
    *@param  lieferantenListe
    *@return                  Description of the Return
    *      Value
    */
   Aggregation process(List lieferantenListe)
   {

      /*
       *  [R03]
       */
      String m;

      String id;

      try
      {
         for (int i = 0; i < lieferantenListe.size(); i++)
         {
            Element lieferant =
                  (Element) (lieferantenListe.get(i));
            id = lieferant.getAttributeValue("id");
            m = lieferant.getAttributeValue("m");
            /*
             *  [R03.2]
             */
            if (m.equals("B"))
            {
               System.out.println(
                     "Der Datensatz des Lieferanten " +
                     id +
```

```
                " ist dubios!");
        }
        /*
         *  [R03.3]
         */
             else if (m.equals("C"))
        {

           januar += numerisch(
                 lieferant.getChildText("januar"));
           februar += numerisch(
                 lieferant.getChildText("februar"));
           maerz += numerisch(
                 lieferant.getChildText("maerz"));
           april += numerisch(
                 lieferant.getChildText("april"));
           mai += numerisch(
                 lieferant.getChildText("mai"));
           juni += numerisch(
                 lieferant.getChildText("juni"));
           juli += numerisch(
                 lieferant.getChildText("juli"));
           august += numerisch(
                 lieferant.getChildText("august"));
           september += numerisch(
                 lieferant.getChildText("september"));
           oktober += numerisch(
                 lieferant.getChildText("oktober"));
           november += numerisch(
                 lieferant.getChildText("november"));
           dezember += numerisch(
                 lieferant.getChildText("dezember"));

        } else
        {
           /*
            *  [R03.1]
            *  Satz wird übersprungen
            */
        }

    }
```

```
      } catch (Exception e)
      {
         e.printStackTrace();
      }
      return this;
   }


   /**
    *  Schreibt die XML-datei im Pretty-Print-Format.
    */
   void writeXMLOutput()
   {
      try
      {

         Document docOutput = new Document(
               new Element(
               this.getDtdRootElement()));
         DocType docType = new DocType(
               this.getDtdRootElement(),
               this.getDtdOutputXMLFile());

         docOutput.setDocType(docType);

         Element rootOutput = docOutput.getRootElement();
         addElementeMonate(rootOutput);

         XMLOutputter out = new XMLOutputter();

         Format format = Format.getPrettyFormat();
         out.setFormat(format);

         BufferedWriter bw = new BufferedWriter(
               new FileWriter(outputXMLFile));

         out.output(docOutput, bw);

         bw.close();

      } catch (Exception e)
      {
```

```
            e.printStackTrace();
        }
    }


    /**
     *@param  rootOutput  Addiert die Monatselement zum
     *      Root-Element
     */
    private void addElementeMonate(Element rootOutput)
    {
        rootOutput.addContent(new Element("januar").
              setText(String.valueOf(januar)));
        rootOutput.addContent(new Element("februar").
              setText(String.valueOf(februar)));
        rootOutput.addContent(new Element("maerz").
              setText(String.valueOf(maerz)));
        rootOutput.addContent(new Element("april").
              setText(String.valueOf(april)));
        rootOutput.addContent(new Element("mai").
              setText(String.valueOf(mai)));
        rootOutput.addContent(new Element("juni").
              setText(String.valueOf(juni)));
        rootOutput.addContent(new Element("juli").
              setText(String.valueOf(juli)));
        rootOutput.addContent(new Element("august").
              setText(String.valueOf(august)));
        rootOutput.addContent(new Element("september").
              setText(String.valueOf(september)));
        rootOutput.addContent(new Element("oktober").
              setText(String.valueOf(oktober)));
        rootOutput.addContent(new Element("november").
              setText(String.valueOf(november)));
        rootOutput.addContent(new Element("dezember").
              setText(String.valueOf(dezember)));

    }

    /**
     *  R[03.3.2]
     *
```

```
     *@param  s  String der eine int-Zahl repräsentiert
     *@return    int-Zahl; im Fehlerfall int-Null.
     */
    private int numerisch(String s)
    {
       int x = 0;
       try
       {

          x = Integer.parseInt(s);

       } catch (NumberFormatException e)
       {
          System.err.println(e);
          x = 0;
       }
       return x;
    }

}
```

Klasse `AggregationProg`

```
/**
 *  Example "Lieferantendaten" selektieren und aggregieren
 *  Java Application
 *
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.as.xmldata;

public class AggregationProg
{

   public static void main(String[] args)
   {
      final String inputXMLFile =
            "de/fhnon/as/xmldata/lief-dat.xml";
      final String inputXMLFileChild =
            "lieferant";
```

```
        final String outputXMLFile =
              "de/fhnon/as/xmldata/jahr-tab.xml";

        final String dtdRootElement = "umsatz";
        final String dtdOutputXMLFile =
              "jahr-tab.dtd";

        Aggregation foo = new Aggregation(
              inputXMLFile, inputXMLFileChild,
              outputXMLFile,
              dtdRootElement, dtdOutputXMLFile);

        foo.process(foo.readXMLInput()).writeXMLOutput();

        /*
         *  [R02]
         */
        System.out.println("Alles verarbeitet!");
    }
}
```

Protokolldatei `Aggregation.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\anwd\code>javac
  de/fhnon/as/xmldata/*.java

D:\bonin\anwd\code>java
  de.fhnon.as.xmldata.AggregationProg
java.lang.NumberFormatException:
  For input string: "unbekannt"
Der Datensatz des Lieferanten HausmmannKG ist dubios!
Der Datensatz des Lieferanten WilhelmEG ist dubios!
Alles verarbeitet!

D:\bonin\anwd\code>
```

```
  REM Udate in lief-dat.xml beim Lieferanten
D:\bonin\anwd\code>
  REM OttoAG statt <januar>10</januar>
D:\bonin\anwd\code>
  REM <jaenner>10</jaenner>

D:\bonin\anwd\code>java
  de.fhnon.as.xmldata.AggregationProg
org.jdom.input.JDOMParseException:
  Error on line 5 of document file:
  /D:/bonin/anwd/code/de/fhnon/as/xmldata/lief-dat.xml:
     In Element "lieferant" ist hier "jaenner" nicht zulässig.

D:\bonin\anwd\code>
```

Ergebnis der Auswertung `jahr-tab.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE umsatz SYSTEM "jahr-tab.dtd">

<umsatz>
  <januar>400</januar>
  <februar>400</februar>
  <maerz>330</maerz>
  <april>240</april>
  <mai>310</mai>
  <juni>500</juni>
  <juli>300</juli>
  <august>345</august>
  <september>400</september>
  <oktober>327</oktober>
  <november>310</november>
  <dezember>1500</dezember>
</umsatz>
```

Einige Hinweise zum Arbeiten mit JDOM

JDOM

JDOM erleichtert das Navigieren im Baum der Elemente. Wie in der Beispielklasse `Aggregation` schon genutzt, erhält man das Root-Element folgendermaßen:

```
SAXBuilder builder = new SAXBuilder();
```

```
Document document = builder.build(new File(...));
Element root = document.getRootElement();
```

Eine Liste seiner Kind-Elemente erhält man mit:

```
List allChildren = root.getChildren();
```

Alle Kind-Elemente mit einem vorgegebenen Bezeichner `name` erhält man mit:

```
List namedChildren = root.getChildren("name");
```

Das erste Kind-Eelement mit einem vorgegebenen Bezeichner `name` erhält man mit:

```
List namedChild = root.getChild("name");
```

Das 3. Kind-Element wird gelöscht — sowohl in der `List`-Instanz wie in der `Document`-Instanz — mit:

```
allChildren.remove(2);
```

Alle Kind-Elemente mit dem Bezeichner `name` werden gelöscht mit:

```
allChildren.removeAll(root.getChildren("name"));
```

oder vereinfacht notiert auf der Basis des Root-Elements mit:

```
root.removeChildren("name");
```

Eingefügt wird ein neues Kind-Element mit dem Bezeichner `name` am Anfang mit:

```
allChildren.add(0, new Element("name"));
```

und am Ende mit:

```
allChildren.add(new Element("name"));
```

oder auf der Basis des Root-Elements mit:

```
root.addContent(new Element("name"));
```

Um ein Element im Baum zu verschieben, ist es an der alten Stelle zu „löschen“ (`detach`) und an der neuen Stelle zu positionieren. Beide Punkte sind bedeutsam. Wird das Element vorher nicht „gelöscht“, dann führt JDOM zu einer Ausnahme (`Exception`). Daher notiert man das Verschieben eines Elementes im Baum wie folgt:

```
Element movable = new Element("name"));
parentOld.addContent(movable);
...
parentNew.addContent(movable.detach());
```

Wir nehmen folgendes Element an:

```
<myelement id="1" m="7">Alles klar?</myelement>
```

Die Attribute erhält man beispielsweise mit:

```
List mylist = root.getChilden("myelement");
Element myelement = (Element) (mylist.get(0));
Attribute idAttribute = myelement.getAttribute("id");
int id = idAttribute.getIntValue();
int m = lieferant.getAttributeValue("m");
```

Attribute werden wie folgt gesetzt, modifiziert oder entfernt:

```
myelement.setAttribute("m", "0");
myelement.setAttribute("neu", "OK");
myelement.removeAttribute("m");
```

7.12 Komponentenmodelle

Der Begriff *Component Model* wird in vielfältigen Zusammenhängen verwendet auch in der Java^{TM}-Welt. Hier sind besonders zu unterscheiden:

GUI

1. (ürsprungliche) *JavaBeans*^{TM} (↪ Abschnitt 7.12.1 S. 301)
   Sie werden primär verwendet um GUI-Komponenten zu kombiniern. Sie sind jedoch kein Server-seitiges Modell.

CTM

2. EJB *Enterprise JavaBeans*^{TM} (↪ Abschnitt 7.12.2 S. 307)
   Sie werden primär verwendet als Server-seitiges Modell zum Kombinieren von Komponenten mit einer Transaktionsteuerung im Sinne eines (*Transaction Processing Monitor*[26](s)) verwendet.

Die ursprünglichen *JavaBeans*^{TM} wurden als Komponenten zur *Intra*-Prozessgestaltung konzipiert. Im Unterschied dazu dienen EJB als Komponenten zur *Inter*-Prozessgestaltung (↪ [Monson01] p. 12). Das EJB-Konzept ist daher keine Erweiterung der ursprünglichen *JavaBeans*^{TM}, wie manchmal formuliert wird, sondern hat eine andere Aufgabe, nämlich die eines *Component Transaction Monitor*(s) (CTM[27]).

In diesem Kontext definiert ein Komponentenmodell einen Vertrag zwischen dem Komponentenentwickler und dem System auf dem die Komponente „laufen“ soll. Der Vertrag spezifiziert wie die Komponente zu entwickeln und zusammenzufügen ist. Wenn die Komponente entsprechend gestaltet ist, wird sie zu einem unabhängigen Softwarestück, das verteilt und von anderen Applikationen genutz werden kann.

Der eigentliche Traum, den das objekt-orientierte Paradigma vermittelt, ist die Entwicklung von problemlos, überall wiederverwendbaren Komponenten. Mit einer anwendungsfeldspezifischen Bibliothek von solchen Komponenten soll sich das Entwickeln der gewünschten Anwendung auf das „Zusammenstecken von Bauteilen“ reduzieren. Programmieren im engeren Sinne ist dann nur noch ein „*plugging in*-Prozeß von genormten Bausteinen“ in den eigenen Rest-Quellcode.

7.12.1 JavaBeans^{TM}

Die *Plug-In*-Bausteine, konzipiert für GUI-Zwecke, sind die ursprünglichen *JavaBeans*^{TM}. Ihr Vertrag zielt auf eine einfache Anpassbarkeit und zwar mit Hilfe von Werkzeugen.

> „*A JavaBean is a reusable software component that can be manipulated visually in a builder tool.*“ ([Vanderburg97] p. 578 oder [Flanagan97] p. 233).

[26] Ein weit verbreiteter Transaktionsmonitor ist das *Customer Information Control System* (CICS) der IBM Corporation. CICS wurde 1968 eingeführt und hat sich zu einer *Host*-basierten Plattform für zeitkritsche (*mission-critical*) Massenanwendungen entwickelt.

[27] Der Begriff wurde 1999 von Anne Thomas (jetzt Ms. Manes) geprägt. (↪ [Monson01] p. 4)

BDK

Wenn zur Anpassung kein leistungsfähiges, kommerzielles *Builder*-Werkzeug[28] verfügbar ist, kann man *Beans Development Kit*[29] (BDK) von Sun Microsystems, Inc. USA, mit seiner Testbox genutzt werden.

Ein JavaBean ist ein normales Objekt[30], das Eigenschaften, Ereignisse und Methoden exportiert und zwar nach vorgegebenen Konstruktionsmustern und (Namens-)Regeln. Diese Vorgaben umfassen primär folgende Punkte:

1. Eine Eigenschaft (*property*) des Objektes ist ein Teil des inneren Zustandes eines JavaBean. Sie wird über öffentliche Zugriffsmethoden verfügbar. Diese `get`- und `set`-Methoden, salopp auch als „Getter“ und „Setter“ bezeichnet, haben eine fest vorgegebene Signatur. Für eine Eigenschaft mit dem Namen `Foo` sind es folgende Methoden:

   Getter

   - `public FooType getFoo(){...}`
   - `public boolean isFoo(){...}`

   Setter

   - `public void setFoo(FooType wert){...}`
   - `public void setFoo(boolean wert){...}`

   Zusätzlich gibt es für Eigenschaften auch einen indizierten Zugriff. Dieser Zugriff läßt sich für ein Element mit dem Namen PropertyName und dem Typ PropertyElement wie folgt beschreiben:

   - `public PropertyElement getPropertyName (int index){...}`
   - `public void setPropertyName (int index, PropertyElement wert){...}`

   Listener

2. Die Ereignisbehandlung basiert auf dem Delegationsmodell (*listener classes* für *events*, ↪ Abschitt 7.2 S. 168). Dazu ist folgendes Paar von Methoden zu definieren:

[28] Liste der Hersteller: `http://splash.javasoft.com/beans/tools.html` (Zugriff: 24-Mai-1998)

[29] BDK Quelle: `http://splash.javasoft.com/beans/bdk_download.html` (Zugriff: 24-Mai-1998)

[30] Auf unterstem Level können beispielsweise alle AWT-Komponenten als *Beans* bezeichnet werden.

- `public void addEventListenerType (EventListenerType l){...}`
- `public void removeEventListenerType (EventListenerType l){...}`

Dabei muß EventListenerType abgeleitet sein von `java.util.EventListener`. Sein Name muß mit dem Wort `Listener` enden, also zum Beispiel:

```
addFooListener(FooListener l);
```

3. Persistente Objekte basieren auf dem Interface `java.io.Serializable` (↪ Abschitt 7.3 S. 182).

4. Verfügbar wird ein JavaBean als ein *Java Archiv* (JAR) mit einem sogenannten Manifest (JAR-Parameter `-m`, ↪ Seite 189). Eine solche Manifest-Datei hat folgende Eintragungen:

Manifest

```
Name: className
Java-Bean: trueOrFalse
Name: nextClassName
Java-Bean: trueOrFalse
⋮
```

Beispielsweise hat `MyBean` dann folgendes Manifest[31]:

```
Name: myjava/AllBeans/MyBean.class
Java-Bean: true
```

Das folgende JavaBean-Beispiel skizziert grob eine übliche Konstruktion.[32]

[31] Hinweis: Auch auf Windows-Plattformen gilt hier der Schrägstrich (*slash*) und nicht der *Backslash*.

[32] Für weitere Informationen zum Schreiben von eigenen *JavaBeans*^{TM} siehe zum Beispiel [Vanderburg97].

bound

Beispiel `SimpleBean` Dieses Beispiel hat eine Eigenschaft `Witz` mit dem Getter `getWitz()` und dem Setter `setWitz()`. Es informiert automatisch „interessierte Parteien“ **bevor** sich diese Eigenschaft ändert. Man spricht daher von einer *bound*-Eigenschaft. Dazu dient die Klasse
`java.beans.PropertyChangeSupport`.
Sie stellt die Methode `firePropertyChange()` bereit, die ein Objekt von `PropertyChangeEvent` an alle registrierten Listener sendet.

constrained

Mit der Klasse `java.beans.VetoableChangeSupport` und deren Methoden wird die Eigenschaftsänderung von einer Bedingung abhängig gemacht. Man spricht daher von einer *constrained*-Eigenschaft. Bevor eine Eigenschaftsänderung erfolgt, werden alle Listener angefragt, indem ein `PropertyChangeEvent`-Objekt gesendet wird. Wenn ein Listener ein Veto sendet, dann schickt die Methode `fireVetoableChange()` wieder an alle Listener ein `PropertyChangeEvent`-Objekt um mitzuteilen, daß die Eigenschaft wieder den ursprünglichen Wert hat.

Selbst wenn man nicht beabsichtigt, eigene Klassen als *JavaBeans*^{TM} zu verbreiten, so ist es doch sinnvoll die Konstruktionsmuster und (Namens-)Regeln direkt zu übernehmen.

Klasse `SimpleBean`

```
/**
 *  Grundstruktur für JavaBean mit Kontrollmechanismus für
 *  eine Änderung: ,,bound'' und ,,constrained'' Idee aus
 *  Glenn Vanderburg; MAXIMUM Java 1.1, 1997, p. 597
 *
 *@since      23-Mai-1998 29-May-1998
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.beans;
import java.beans.*;
import java.awt.*;

public class SimpleBean extends Canvas
{
   String myWitz = "Piep, piep ... lieb";
```

```
/*
 *  bound-Eigenschaft
 *  --- Automatisches Informieren über eine Änderung
 */
private PropertyChangeSupport
      changes = new PropertyChangeSupport(this);

/*
 *  constrained-Eigenschaft
 *  --- Vetomechanismus für eine Änderung
 */
private VetoableChangeSupport
      vetos = new VetoableChangeSupport(this);


public SimpleBean()
{
   setBackground(Color.green);
}


public String getWitz()
{
   return myWitz;
}


public void setWitz(String neuerWitz)
       throws PropertyVetoException
{
   String alterWitz = myWitz;
   vetos.fireVetoableChange(
         "Witz", alterWitz, neuerWitz);

   /*
    *  Kein Veto für die Änderung
    */
   myWitz = neuerWitz;

   /*
    *  Nachträgliche Information über die Änderung
    */
```

```
      changes.firePropertyChange(
            "Witz", alterWitz, neuerWitz);
   }


   /*
    *  Vetomechanismus für die Änderung
    *  mit VetoableChangeListener
    */
   public void addVetoableChangeListener(
         VetoableChangeListener l)
   {
      vetos.addVetoableChangeListener(l);
   }


   public void removeVetoableChangeListener(
         VetoableChangeListener l)
   {
      vetos.removeVetoableChangeListener(l);
   }


   /*
    *  Änderungsinformation mit
    *  PropertyChangeListener
    */
   public void addPropertyChangeListener(
         PropertyChangeListener l)
   {
      changes.addPropertyChangeListener(l);
   }


   public void removePropertyChangeListener(
         PropertyChangeListener l)
   {
      changes.removePropertyChangeListener(l);
   }


   /*
```

```
     *  Sonstige exportierte Methode
     */
    public Dimension getMinimuSize()
    {
       return new Dimension(100, 150);
    }
}
```

Protokolldatei `SimpleBean.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
    Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
    (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/FHNON/beans/SimpleBean.java

C:\bonin\anwd\code>dir de\FHNON\beans\*
    1.654 SimpleBean.class
    2.176 SimpleBean.java

C:\bonin\anwd\code>
```

7.12.2 EJB (*Enterprise JavaBeans*^{TM})

Enterprise JavaBeans
is a standard server-side component model
for component transaction monitors.
(↪ [Monson01] p. 5)
.

Das EJB-Komponentenmodell in der Form von EJB 2.0 unterscheidet drei unterschiedliche Bean-Typen:

1. *Entity Beans*
   Es sind RMI-basierte Server-seitige Komponenten.

2. *Session Beans*
   Es sind RMI-basierte Server-seitige Komponenten.

3. *Message-driven Beans*
   Es sind JMS[33]-basierte Server-seitige Komponenten, die asynchrone Nachrichten bearbeiten.

Die Typunterscheidung ist aufgrund der Flexibilität der Beans unscharf. Eine charakteristisches Unterscheidungsmerkmal ist die Persistenz[34]. *Entity Beans* sind geprägt durch ihren persistenten Zustand, während die anderen *Beans* auf das Modellieren von Interaktionen abzielen. Die Funktionen werden weiter verdeutlicht durch die Klassen und Interfaces, die die Beans erweitern bzw. implementieren (↪ Abbildung 7.22 S. 309):

- RemoteInterface `extends javax.ejb.EJBObject`
  Entity Beans und Session Beans definieren mit diesem Interface die *Business Methods* für Applikationen außerhalb des EJB-Containers.
  Namensbeispiel: `RaumRemote`

- RemoteHomeInterface `extends javax.ejb.EJBHome`
  Entity Beans und Session Beans definieren mit diesem Interface die *Life-cycle Methods* eines Beans für Applikationen außerhalb des EJB-Containers. Es geht also um Methoden für das Erzeugen, Löschen und Finden von Beans.
  Namensbeispiel: `RaumHomeRemote`

- LocalInterface `extends javax.ejb.EJBLocalObject`
  Entity Beans und Session Beans definieren mit diesem Interface die die *Business Methods* für andere Beans im gleichen EJB-Container.
  Namensbeispiel: `RaumLocal`

- HomeLocalInterface `extends javax.ejb.EJBLocalHome`
  Entity Beans und Session Beans definieren mit diesem Interface die die *Life-cycle Methods* für andere Beans im gleichen EJB-Container.
  Namensbeispiel: `RaumHomeLocal`

[33] JMS ≡ *Java Messaging Service*
[34] Zum Begriff *Persitenz* ↪ Abschnitt 7.3 S. 182.

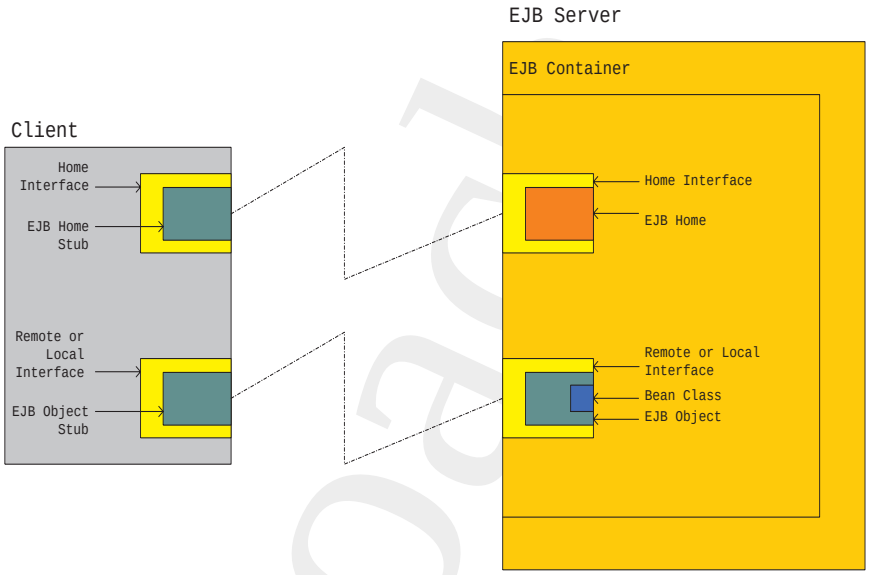


Abbildung 7.22: Skizze der *Enterprise JavaBeans*^{TM}-Architektur

- EntityBeanClass `implements javax.ejb.EntityBean`
  Ein Entity Bean muß dieses Interface implemenieren.
  Namensbeispiel: `RaumBean`

- SessionBeanClass `implements javax.ejb.SessionBean`
  Ein Session Bean muß dieses Interface implemenieren.
  Namensbeispiel: `WartungBean`

- MessageDrivenBean `implements javax.ejb.MessageDrivenBean, javax.jms.MessageListener`
  Ein Message-driven Bean muß beide Interfaces implemenieren.

- BeanClass `extends javax.ejb.EnterpriseBean`
  Ein Entity Bean, Session Bean und ein Message-driven Bean erben jeweils von dieser Klasse.
  Namensbeispiel: `RaumBean`

Bezeichnung

Üblicherweise[35] bezeichnet der Begriff *Enterprise Bean* (oder auch kurz *Bean*) jeden der obigen *Bean*-Typen. Dieser Begriff wird wird oft als EJB notiert, so auch im JAVA^{TM}–COACH. Zum Beispiel bezeichnet „Raum EJB“ ein *Enterprise Bean* mit allen Teilen, also mit den *Component Interfaces* und Klassen. Soll nur das *Remote Interface* angeben werden, dann wird „`RaumRemote`“ notiert. Geht es um das *Local Component Interface* dann wird „`RaumLocal`“ notiert. Bei *Home Interfaces* wird das Wort `Home` hinzugefügt, also „`RaumHomeRemote`“ und „`RaumHomeLocal`“. Zur Bezeichnung der *Bean*-Klasse wird das Wort `Bean` angehängt, beispielsweise „`RaumBean`“.

EJB-Beispielskizze Raumbewirtschaftung

„EJB ist hervorragend geeignet
für Prototypen innerhalb von Diplomarbeiten,
aber nur sehr eingeschränkt in großen Systemen
mit hohem Transaktionsvolumen.“
(↪ [Broy/Siedersleben02] S. 57)

Die *Business Methods* in diesem Beispiel können einen Raum bezeichnen und die Anzahl der Plätze in dem Raum setzen. Natürlich bedarf es bei einer konkreten Anwendung mehr Methoden, aber für diese EJB-Beispielsskizze sind sie ausreichend.

Interface `RaumRemote` Die *Business Methodes* sind in der Form von Signaturen im *Remote Interface* angegeben.

```
/**
 *  Einfaches EJB Beispiel
 *
 *@since      19-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.0
 */

package de.fhnon.ejb.raum;

import java.rmi.RemoteException;
```

[35] Siehe zum Beispiel ↪ [Monson01].

```
public interface RaumRemote
       extends javax.ejb.EJBObject
{
   public String getBezeichnung()
          throws RemoteException;


   public void setBezeichnung(String bezeichnung)
          throws RemoteException;


   public int getAnzahlPlaetze()
          throws RemoteException;


   public void setAnzahlPlaetze(int anzahlPlaetze)
          throws RemoteException;


   public abstract Integer getId()
          throws RemoteException;


   public abstract void setId(Integer id)
          throws RemoteException;

}
```

Interface `RaumHomeRemote` Die *Life-cycle Methods* sind im *Remote Home Interface* definiert. Dabei ist die Methode `create()` zuständig für die Initialisierung einer Bean-Instanz.

```
/**
 *  Einfaches EJB Beispiel
 *
 *@since      19-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.0
 */
```

```
package de.fhnon.ejb.raum;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface RaumHomeRemote
       extends javax.ejb.EJBHome
{
   public RaumRemote create(Integer id)
          throws CreateException, RemoteException;


   public RaumRemote findByPrimaryKey(
         Integer primaryKey)
          throws FinderException, RemoteException;
}
```

Interface `RaumLocal`

```
/**
 *  Einfaches EJB Beispiel
 *
 *@since      19-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.0
 */

package de.fhnon.ejb.raum;

public interface RaumLocal
       extends javax.ejb.EJBLocalObject
{
}
```

Interface `RaumHomeLocal`

```
/**
 *  Einfaches EJB Beispiel
 *
```

```
 *@since      19-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.0
 */

package de.fhnon.ejb.raum;

import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface RaumHomeLocal
       extends javax.ejb.EJBLocalHome
{
   public RaumLocal create(Integer id)
          throws CreateException;


   public RaumLocal findByPrimaryKey(
         Integer primaryKey)
          throws FinderException;
}
```

Interface `RaumBean` Die Klasse `RaumBean` ist `abstract` wie auch einige Methoden, die auf den persistenten Zustand des EJB zugreifen oder diesen ändern. Der Grund liegt in der Nutzung von EJB 2.0 *Container-managed Etity Bean*[36].

```
/**
 *  Einfaches EJB Beispiel
 *
 *@since      19-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.0
 */

package de.fhnon.ejb.raum;

import javax.ejb.EntityContext;

public abstract class RaumBean implements
```

[36] Näheres dazu siehe EJB-Literatur zum Beispiel ↪ [Monson01],

```
      javax.ejb.EntityBean
{

   public Integer ejbCreate(Integer id)
   {
      setId(id);
      return null;
   }


   public void ejbPostCreate(Integer id)
   {
      // do nothing
   }


   public abstract String getBezeichnung();


   public abstract void setBezeichnung(
         String bezeichnung);


   public abstract int getAnzahlPlaetze();


   public abstract void setAnzahlPlaetze(
         int anzahlPlaetze);


   public abstract Integer getId();


   public abstract void setId(Integer id);


   public void unsetEntityContext()
   {
      // not implemented
   }
```

```
    public void ejbActivate()
    {
       // not implemented
    }


    public void ejbPassivate()
    {
       // not implemented
    }


    public void ejbLoad()
    {
       // not implemented
    }


    public void ejbStore()
    {
       // not implemented
    }


    public void ejbRemove()
    {
       // not implemented
    }

}
```

Client-Skizze `ClientA`

Der Client wird mit dem EJB Server verbunden. Um auf ein *Entprise Bean* zugreifen zu können nutzt `ClientA` das JNDI-Paket (*Java Naming and Directory Interface*). Ähnlich wie bei einem Treiber für *Java Database Connectivity* (JDBC) ist ein solches Paket herstellerabhängig. Die herstellerspezifischen Angaben enthält die Klassenmethode `getInitialContext()`, die dazu die Klasse `java.util.Properties` nutzt.

```
/**
```

```
 *  Einfaches EJB Beispiel
 *
 *@since      19-Dec-2002
 *@author     Hinrich Bonin
 *@version    1.0
 */

package de.fhnon.ejb.raum;

import java.rmi.RemoteException;
import java.util.Properties;
import de.FHNON.ejb.raum.RaumHomeRemote;
import de.FHNON.ejb.raum.RaumRemote;

import javax.ejb.CreateException;
import javax.ejb.FinderException;

import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;

import javax.rmi.PortableRemoteObject;

public class ClientA
{
   public static void main(String[] args)
   {
      try
      {
         Context jndiContext =
               getInitialContext();
         Object ref =
               jndiContext.lookup("RaumHomeRemote");
         RaumHomeRemote home = (RaumHomeRemote)
               PortableRemoteObject.narrow(
               ref, RaumHomeRemote.class);
         RaumRemote raum1 =
               home.create(new Integer(1));
         raum1.setBezeichnung("Grosser Saal");
         raum1.setAnzahlPlaetze(120);
         raum1.setId(new Integer(1));
```

```
            RaumRemote raum2 =
                  home.findByPrimaryKey(new Integer(2));
            System.out.println(raum2.getId() + ": " +
                  raum2.getBezeichnung() + " hat " +
                  raum2.getAnzahlPlaetze() +
                  " Plaetze.");

         } catch (RemoteException re)
         {
            re.printStackTrace();
         } catch (NamingException ne)
         {
            ne.printStackTrace();
         } catch (CreateException ce)
         {
            ce.printStackTrace();
         } catch (FinderException fe)
         {
            fe.printStackTrace();
         }

      }


      public static Context getInitialContext()
            throws NamingException
      {

         Properties p = new Properties();
         /*
          *  Define the JNDI properties
          *  specific to the vendor
          *  here for example: IBM WebSphere
           */
         p.put(Context.PROVIDER_URL, "iiop:///");
         p.put(Context.INITIAL_CONTEXT_FACTORY,
               "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
         return new InitialContext(p);
      }

   }
```

Skizze *Deployment Descriptors*-**File:**
`de/fhnon/ejb/raum/META-INF/ejb-jar.xml`

Die Verteilung (englisch *Deployment*) und Zusammenstellung wird durch spezielle XML-Elemente in dieser Datei beschrieben. Die *Deployment*--Deskriptoren ermöglichen ähnlich wie *Property files* eine Anpassung der *Enterprise Beans* ohne die Software selbst ändern zu müssen. Diese Datei wird zusammen mit dem Bean (Klassen und Interfaces) zu einem *Java Archiv* (JAR-File ↪ Abschnitt 7.3.3 S. 189) gepackt.

JAR

Zu Beginn einer XML-Datei wird der Dokumententyp angegeben und zwar mit der `<!DOCTYPE>`-Angabe für das Wurzelelement, hier `<ejb-jar>`. Diese Angabe umfaßt:

DTD

- die Art des Standards, hier öffentlich (`PUBLIC`) und von keiner amtlichen Standardisierungsinstanz („-“)
- die Organisation, die für diese *Document Type Definition* (DTD) verantwortlich ist, hier `Sun Microsystems, Inc.`,
- die DTD mit Version, hier `DTD Enterprise JavaBeans 2.0` und
- eine DTD-Quelle als URL, hier `http://java.sun.com/dtd/ejb-jar_2_0.dtd`

```
<!DOCTYPE eib-jar PUBLIC
  "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
  "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>RaumEJB</ejb-name>
      <home>de.fhnon.ejb.raum.RaumHomeRemote</home>
      <remote>de.fhnon.ejb.raum.RaumRemote</remote>
      <local-home>de.fhnon.ejb.raum.RaumHomeLocal</local-home>
      <local>de.fhnon.eib.raum.RaumLocal</local>
      <ejb-class>de.fhnon.ejb.raum.RaumBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Integer</prim-key-class>
      <reentrant>False</reentrant>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

Kapitel 8

Konstruktionsempfehlungen

Das Primärziel der Softwarekonstruktion ist eindeutig und unstrittig. Es gilt:

- *nützliche* Software zu konstruieren,
  das heißt Software, die eine nützliche Aufgabe erledigt, wobei die Erledigung vereinfacht oder erst ermöglicht wird,
- und zwar *hinreichend fehlerfrei* und *termingerecht*.

Von diesem Primärziel leiten sich eine Vielzahl von Sekundärzielen ab, wie zum Beispiel:

1. Konstruiere Programm(teil)e in gewünschter Qualität.
2. Konstruiere Programm(teil)e, die wiederverwendbar sind.
3. Konstruiere Programm(teil)e, die sich leicht pflegen und ergänzen lassen.
4. Konstruiere Programm(teil)e, die dokumentierbar unddurchschaubar sind.
5. . . .

Für eine Java-basierte Systemarchitektur bei betrieblichen, transaktionsorientierten Kernsystemen lassen sich die Hauptanforderungen mit den Stichworten:

1. Performance (große Menge von Transaktionen pro Sekunde)
2. Realtimefähigkeit (Einhaltung von definierten Verarbeitungszeiten)
3. Unterbrechungsfreier Betrieb ($365\frac{Tage}{Jahr}$ mit $24\frac{h}{Tag}$)
4. Restart-Fähigkeit (schnelle Arbeitswiederaufnahme im Katastrophenfall)
5. Komponentenorientierung (Austauschbarkeit von Altversionen im laufenden Betrieb)

charakterisieren.

Eine Software, die diese Anforderungen hinreichend erfüllt, kann nur im Team erarbeitet werden. Eine effektive Teamarbeit bedingt jedoch den Einsatz geeigneter Werkzeuge und die strikte Einhaltung von Konventionen (Standards). Exemplarisch für ein leistungsfähiges Teamwerkzeug wird hier der Einsatz der *Integrated Development Environment* (IDE) *Eclipse* mit dem *Concurrent Versions System* (CVS) näher erläutert. Exemplarisch für Konventionen werden hier Hinweise zur Code-Gestaltung skizziert.

Trainingsplan

Das Kapitel „Konstruktionsempfehlungen“ erläutert:

- den Einsatz der teamfähigen IDE am Beispiel *Eclipse*
  ↪ Seite 321 . . .
- den Einsatz von Werkzeugen für spezielle Aufgaben
  ↪ Seite 79 . . .
- die Notwendigkeit von Code-Konventionen und gibt Empfehlungen,
  ↪ Seite 345 . . .
- die Idee von vorgefertigten Geschäftsobjekten (*Common Business Objects*) und Geschäftsvorgängen (*Core Business Processes*) und
  ↪ Seite 369 . . .
- Alternativen für die Quellcodegestaltung aus Sicht der Performance.
  ↪ Seite 351 . . .

8.1 Einsatz einer teamfähigen IDE

IBM

Im November 2001 brachte IBM, Object Technology International (OTI) mit acht anderen Unternehmen *Eclipse* auf den Markt. Eclipse versucht das properitäre Muster zu überwinden. Eclipse ist eine Integrationsplattform und zwar als ein frei fügbares Werkzeug. Eclipse kann problemlos mit anderen Entwicklunswerkzeugen arbeiten. Vielfältige Programmierschnittstellen bilden die Möglichkeiten zur Integration von anderen Werkzeugen. Eclipse läuft auf den marktüblichen Betriebssystemen und

ist sprachneutral. Als *Open Source Project* wird der Quellcode kostenlos bereitgestellt.

Eclipse

Der Name *Eclipse* (deutsch „Verdunkelung“) umfaßt drei Aspekte:

- eine Entwicklungsumgebung für Java,
- eine Plattform zur Integration von Werkzeugen und
- eine Gemeinschaft für die Entwicklung des frei verfügbaren Quellcodes

Der dritte Aspekt bezieht sich auf den Zusammenschluss von Softwareexperten, die das gemeinsame Interesse an einer Software zur Werkzeugintegration vereint. Sie wollen Eclipse nutzen und dazu beitragen, dass Eclipse ein bedarfsgerechtes, leistungsfähiges Produkt wird. Wer mitwirken will informiere sich unter der URL `www.eclipse.org` (online 19-Oct-2003).

8.1.1 Eclipse — Überblick

Eclipse hält Projekte, Verzeichnise und Dateien (≡ Ressourcen) mit denen man arbeitet im eigenen Arbeitsraum (Workspace). Standardmäßig ist dieser bei Windows Betriebssystemen im Verzeichnis `workspace` unter dem Hauptordner von Eclipse, zum Beispiel für das Projekt `MyFirstJavaProject` unter:

`work-space`

```
C:/Programme/eclipse/workspace/MyFirstJavaProject/
```

Informationen über den Workspace enthält das Verzeichnis `.metadata` (Bitte nicht ändern!). Die Datei `.project` enthält spezifische Informationen für das jeweilige Projekt, zum Beispiel Referenzen zu anderen Projekten.

Sehr hilfreich ist die Möglichkeit Ressourcen zu vergleichen und durch vorhergehende Versionen der Ressource zu ersetzen. Dazu wählt man aus dem Kontextmenue (rechte Maustaste) in der Navigatorsicht **Local History** > **Compare With**. Man kann auch zwei Projekte oder zwei Verzeichnisse auswählen indem man vom Kontextmenue in der Navigatorsicht **Compare With** > **Each Other** auswählt.

Lokale Historieinformationen werden für eine Datei gehalten, selbst wenn man sie vom Projekt löscht. Daher können Dateien wiederhergestellt werden, die in Projekten oder Verzeichnissen gelöscht wurden. Zur

Wiederstellung der gelöschte Datei wählt man das Projekt oder das Verzeichnis welches sie enthielt und wählt **Restore From Local History** . . . im Kontextmenue.

Das Hilfesystem benutzt die eingebaute Servlet-Maschine Tomact aus dem Apache Projekt um das gewünschte Dokument auszuliefern. Daher kann man auf das Hilfesystem von einem Web Browser außerhalb von Eclipse zugreifen. Das ist zweckmäßig wenn man eine Menge von Hilfethemen hat, auf die man häufiger zugreift. Um die URL zum jeweiligen Thema zu erhalten, klickt man die rechte Maustaste im Inhaltsfenster. Dies zeigt dann das Browser Kontextmenue und nicht das von Eclipse. Vom Browser Kontextmenue wählt man **Create Shortcut**.

Tomcat

Hinweis: In der Deutschen Fassung des Internet Explorer wählt man **Zu Favoriten hinzufügen** . . .). Die URL sieht dann wie folgt aus:
`http://127.0.0.1:6969/help/index.jsp?topic=/org.eclipse.help/doc/help_home.html`
Meine Eclipse-Dokumentation ist erreichbar unter:
`http://193.174.33.66:6969/help/index.jsp`

Wenn das Hilfesystem läuft, dann übernimmt man diese URL in seinen (externen) Web Browser um die zugehörende Information zu sehen. Die URL enhält eine Portnummer, in unserem Fall die Portnummer 6969, die zur Kommunikation mit der Servletmaschine benutzt wird. Eclipse konfiguriert die Servletmaschine für eine dynamsiche Zuweisung einer freien Portnummer. Man kann dies ändern, so das stets die gleiche Portnummer benutzt wird. Die URLs bleiben dann stets die gleichen. Dazu setzt man in der Datei `preference.ini`, die sich im Verzeichnis `org.eclipse.tomcat_4.0.6` im Verzeichnis `plugins` befindet, die Portnummer. Um Tomcat so zu konfigurieren, das stets die gleiche Portnummer verwendet wird, ändert man `port=0` zu `port=xxxx`, wobei `xxxx` eine gültige unbenutzte TCP/IP-Portnummer ist.

Hinweis: In Eclipse 2.1 wählt man **Window** > **Preferences** und dann **Help** und dann **Help Server**. Das Menu ermöglicht dann das Setzen der Portnummer.

Help Server

Eclipse ist ein relativ kompaktes, effektives Programm. Es nimmt sich zur Laufzeit nicht viel Plattenspeicher oder Arbeitsspeicher. Mit intensiver Benutzung, also mit der Anzahl und Größe der Projekte so-

wie ihren Abhängigkeiten, und auch wenn man zusätzliche Eclipse-basierte Angebote in die Installation einbaut, wachsen natürlich die Bedarfsanforderungen. Abhängig von dem Leistungsvermögen des Computers wird dann ein Punkt erreicht bei dem die Geduld überstrapaziert wird. Wenn dies eintritt, gibt es eine Anzahl von Gegenmaßnahmen, die man ergreifen kann, um wieder hinreichend schnell arbeiten zu können.

Hinweis: Meine Eclipse 2.1 Installation umfasst zur Zeit ca. $85MB$ Plattenspeicher und benötigt zur Laufzeit ca. $1,7MB$ Arbeitsspeicher (ohne den Bedarf zusätzlicher Java-Prozesse).

Wenn Eclipse gestartet wird, baut Eclipse den Zustand des letzten Herunterfahrens wieder auf. Man hat daher folgende Möglichkeiten um die Startdauer zu verkürzen:

- Man schließt offene Sichten und Editoren bevor man Eclipse verläßt.
- Man reduziere die Anzahl der Projekte, die man im Workspace geöffnet hat. Man kann ein Projekt schließen, indem man in der Navigatorsicht aus dem Kontextmenue **Close** auswählt. Die Informationen für dieses Projekt werden dann beim Start von Eclipse nicht geladen; ausgenommen die Projektdefinition und die Ressourcen.

8.1.2 Eclipse — Edieren

Debug

Der Java-Editor stellt eine große Menge von Funktionen bereit. Sie helfen den Java Code mit weniger Aufwand zu schreiben, insbesondere weniger Flüchtigkeits- und Schreibfehler zu machen. Unter den Editor-Funktionen umfassen einen Inhalts-Assistent zur Vervollständigung von Java-Ausdrücken, zur Codegenerierung, zur unmittelbaren Fehlererkennung, zur schnellen Korrekturen für Codefehler. Darüberhinaus bestehen Möglichkeiten mit verschiedenen Laufzeitumgebungen und Java-Dokumentationsgenerationen zu arbeiten.

Typenhierarchie

Die Super- und Subtypen einer Klasse oder eines Interfaces können in einer Hierarchie angezeigt werden. Dazu markiert man ein Element, beispielsweise eine Java-Quellcodedatei, eine Klasse, eine Methode oder ein Feld, und wählt dann **Open Type Hierarchy** aus dem Kontextmenue oder drückt die Funktionstaste **F4**. Diese Funktion kann aus jeder Java-Sicht aktiviert werden.

8.1.3 Eclipse — Fehleranalyse

Die Java-Entwicklungswerkzeuge (JDT) mit der Fähigkeit Fehler zu erkennen umfassen die Debug-Perspektive, verschiedene Sichten und attraktive Erweiterungen des Java-Editors und zwar so, dass man Laufzeitfehler findet und diese zur Laufzeit im Programm korrigiert. Man kontrolliert die Ausführung des Java-Programms indem Unterbrechungs- und Beobachtungspunkte setzt, den Inhalt von Feldern und Variablen prüft und ändert, Schritt für Schritt durch die Ausführung des Programmes geht und Threads anhält und wieder startet.

Der einfache Weg einen Unterbrechungspunkt zu setzen ist der Doppelklick in der Markierungsleiste des Editors und zwar bei der Zeile bei der man diesen Punkt definieren will. Man kann auch den Einfügungszeiger auf die Zeile setzen und dann die Tasten **Ctrl+Shift+B** drücken.

Man steuert die Programmausführung aus der Debug-Sicht mit den folgenden Aktionen:

F6 **Step Over** führt eine Anweisung aus und setzt die Ausführung der nachfolgenden Anweisung aus.

F5 **Step Into** gilt für einen Methodenaufruf wie folgt. Es wird ein neuer Stackrahmen erzeugt, die Methode in der Anweisung aufgerufen und die Ausführung der ersten Anweisung in der Methode ausgesetzt. Für alle anderen Anweisungen, wie Zuweisungen und Alternativen, ist die Wirkung die gleiche wie bei **Step Over**.

F7 **Step Return** setzt die Ausführung bis zum Ende der laufenden Methode fort und setzt die Ausführung bei der nächsten Anweisung nach dem Methodenaufruf oder vorher wenn eine Unterbrechungspunkt erreicht wurde aus.

F8 **Resume** bewirkt die Ausführung fortzuführen bis das Programm beendet ist oder vorher ein anderer Unterbrechungspunkt erreicht wurde.

□ **Terminate** beendet die laufende Ausführung ohne mehr weitere Anweisungen auszuführen.

Ein **Beobachtungspunkt** (oder Feld-Unterbrechungspunkt) setzt die Ausführung auf der Codezeile aus, die zuständig ist für den Zugriff auf

Watch-point

das Feld für den der Beobachtungspunkt definiert ist. Wenn man beobachten will wie auf ein Feld zugegriffen wird, dann ist es oft einfacher einen Beobachtungspunkt für das Feld zu verwenden als es mit einer Menge von Unterbrechungspunkten auf allen Zeilen zu versuchen, die auf das Feld möglicherweise zugreifen. Man kann Beobachtungspunkte auf Felder in Laufzeitbibliotheken oder JAR-Dateien setzen; das sind Felder in Klassen von denen man die Quelle nicht hat.

Um einen Beobachtungspunkt zu definieren wählt man ein Feld in einer der Java Sichten und dann wählt man **Add/Remove Watchpoint** aus dem Kontextmenue. Man editiere die Eigenschaften eines Unterbrechungspunktes wenn der Beobachtungspunkt die Ausführung nur beim Zugiff, nur bei der Modifikation oder in beiden Fällen ausetzen soll. Beobachtungspunkte können zusätzlich Trefferzähler und Randbedingungen haben.

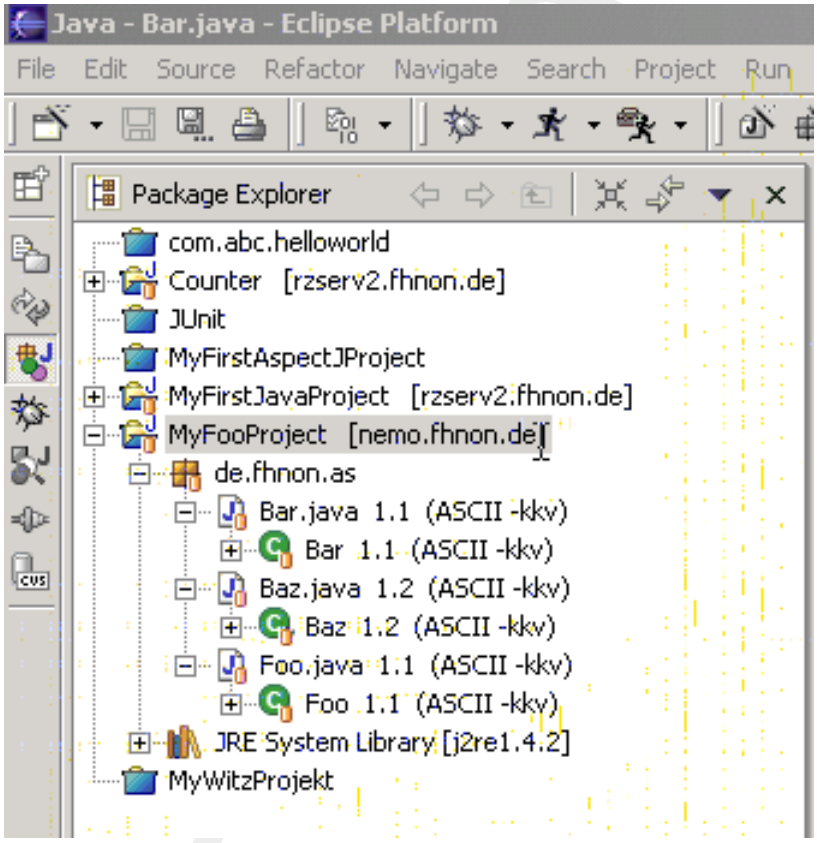
8.1.4 Eclipse — CVS (Concurrent Versions System)

CVS

Eclipse stellt eine direkte Unterstützung für ein spezielles Repository bereit, das sogenannte *Concurrent Versions System* (CVS). CVS ist ein häufig genutztes Repository mit offenem Quellcode[1]. Man kann CVS ohne Installationsarbeiten in Eclipse zu nutzen, wenn ein CVS-Repository auf einem Rechner im Netz verfügbar ist. Man verbindet sich nur über eine Dialogprozedur mit dem CVS-Server. Das Eclipseteam hat CVS für seine eigene Entwicklungsarbeit genutzt; selbst Buchautoren greifen auf diese einfach einsetzbare Versionsverwaltung zurück. Zum Beispiel wurde [Shavor+03] mit CVS über Eclipse erarbeitet. Die Abbildung 8.2 S. 328 zeigt exemplarisch zwei CVS-Repositories mit denen Eclipse verbunden ist. Die Abbildung 8.1 S. 327 zeigt die CVS-Verknüpfung der einzelnen Projekt.

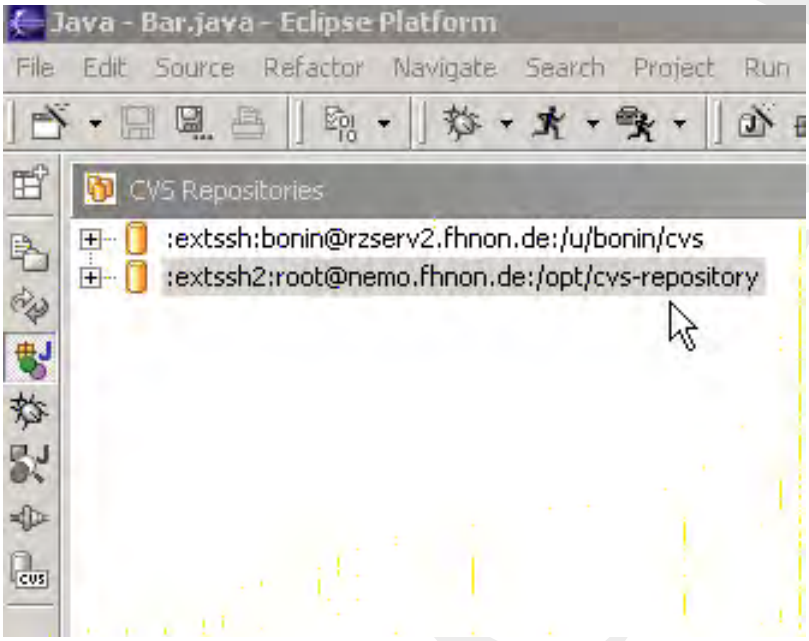
CVS is ein Projekt mit offenem Quellcode und läßt sich zurückführen auf eine einfache Menge von UNIX-Shell-Skripten, die von Dick Grune bereitgestellt wurden. Brian Berliner entwarf und codierte im Jahre 1989 CVS. Mit der Zeit entwickelte es sich durch Beiträge von vie-

[1] CVS unterliegt der *GNU General Public License*. Eine Kopie der Lizenzbedingungen enthält der *CVS distribution kit*.



Legende:
Dieses *Package-Explorer*-Beispiel zeigt mehrere Projekt. Das Projekt `MyFoo-Project` setzt sich aus Klassen mit underschiedlichen Versionen zusammen. So hat die Klasse `Bar` die Versionsnummer 1.1 während die Klasse `Baz` die Versionsnummer 1.2 hat. Außerdem zeigt das Beispiel, dass in diesem Workspace Projekte mit unterschiedlichen Repositories (↪ Abbildung 8.2 S. 328) verwaltet werden.

Abbildung 8.1: Eclipse — Package Explorer



Legende:
Dieses *CVS Repositories*-Beispiel verweist auf zwei Reopsitories:

1. Repository auf dem Rechner `rzserv2.fhnon.de` im Verzeichnis `/u/bonin/cvs`. Zugegriffen wird vom Benutzer `bonin` über die *Secure Shell* in der Version 1.
2. Repository auf dem Rechner `nemo.fhnon.de` im Verzeichnis `/opt/cvs-repository`. Zugegriffen wird vom Benutzer `root` über die *Secure Shell* in der Version 2.

Abbildung 8.2: Eclipse — CVS Repositories

len anderen. So enthält CVS das *Revision Control System*[2] (RCS). Dieses verwaltet mehrere Versionen einer Datei in einer Datei mit dem Suffix `,v`. RCS wurde in den frühen 80iger Jahren von Walter Tichy an der Purdue University entwickelt. Das RCS-Konzept stellte eine wesentliche Verbesserung gegenüber seinem Vorgänger *Source Code Control System* (SCCS) dar. Der konzeptionelle Vorteil betrifft die Performance, weil die jeweils neuste Version einer Datei komplett und die Vorgängerversionen nur durch die Differenzen zur neusten Version gespeichert werden. Dies bezeichnet man als *reverse deltas*.

RCS

CVS ist verfügbar auf verschiedenen Plattformen, dazu gehören Windows-, UNIX- und Linux-Systeme. Für weitere Informationen über CVS siehe `ccvs.cvshome.org`.

Wie jedes *Repository* (≈ Verwahrungsort, Fundgrube, Lager) weist auch CVS die Historie von Änderungen der Dateien nach. Die Dateien, die es verwaltet, sind für andere Teammitglieder verfügbar. Es gibt sicherlich leitungsfähigere Repositories[3] als CVS, aber für moderate Teamgrößen mit genauen Zuordnungen der Code-Besitzverhältnisse ist CVS hervorragend geeignet.

Als ein Beispiel dient der CVS-Server in der Version `1.11.1p1` (client/server) auf dem Rechner `nemo.fhnon.de` mit der IP `193.174.33.63`. Hier liegt das Repository im folgenden Verzeichnis:

```
nemo:/opt/cvs-repository/MyFooProject/de/fhnon/as# ls -l
-r--r--r--    1 root      829 Nov  6 09:51 Bar.java,v
-r--r--r--    1 root     1371 Nov  6 09:52 Baz.java,v
-r--r--r--    1 root      817 Nov  6 09:51 Foo.java,v
nemo:/opt/cvs-repository/MyFooProject/de/fhnon/as#
```

CVSROOT

Dieses Repository wurde im Betriebssystem Linux (Debian) mit folgenden Kommandos eingerichtet:[4]

[2] RCS-Web-Site `http://www.gnu.org/software/rcs/rcs.html` (online 11-Nov-2003).

[3] Beispielsweise hat CVS keine Funktionen wie Defekt- oder Funktionsnachweis und auch keine Zusammenbaufähigkeit oder Unterstützung des Entwicklungsprozesses.

[4] Hinweis: Für Studierende der Wirtschaftsinformatik der FHNON wurde ein Repository zum Experimentieren auf dem Rechner `nemo.fhnon.de` unter dem Benutzer `anwend` (Anwendungsentwicklung) im Verzeichnis `/home/anwend/cvs-repository` angelegt. Dieses Repository darf über Eclipse unter strikter Einhaltung der allgemeinen Benutzerregelungen des Rechenzen-

```
>mkdir /opt/cvs-repository
>cvs -d /opt/cvs-repository init
```

Man kann den Ort des Repository auch über die `CVSROOT`-Umgebungsvariable spezifizieren. Beispielsweise indem man im Fall einer *C-Shell* folgende Zeile in die `.cshrc`-Datei einbaut:

```
setenv CVSROOT /opt/cvs-repository
```

Im Fall einer Shell vom Typ *Bash* können folgende Zeilen in die Datei `.profile` oder die Datei `.bashrc` eingebaut werden:

```
CSVROOT=/opt/cvs-repository
export CVSROOT
```

Mit der erfolgreichen Repository-Initiierung ist ein Verzeichnis:

```
/opt/cvs-repository/CVSROOT
```

entstanden.

SSH2

Für den Zugriff auf diesen CVS-Server benutze ich in Eclipse das *CVS-SSH2 plug-in*[5]. Es liegt unter:

```
C:\Programme\eclipse\plugins\com.jcraft.eclipse.cvsssh2_0.0.7>dir
01.09.2003  22:04             4.494 about.html
10.09.2003  19:31             1.377 ChangeLog
28.01.2003  16:56            15.231 cpl-v10.html
10.09.2003  19:20            21.873 cvsssh2.jar
29.08.2003  21:05            84.144 jsch-0.1.7.jar
28.01.2003  17:09                45 plugin.properties
10.09.2003  18:54             1.560 plugin.xml
10.09.2003  23:57             3.068 README
06.11.2003  09:26    <DIR>          src
C:\Programme\eclipse\plugins\com.jcraft.eclipse.cvsssh2_0.0.7>
```

Die Tabelle 8.1 S. 331 gibt die CVS-Kommandos an und demonstriert damit das CVS-Leistungsspektrum.

8.1.5 Eclipse — *Refactoring*

Refactoring

Unter *Refactoring* versteht man eine Überarbeitung von vorhandenem

trums der FHNON eingesetzt werden. Für andere Zwecke darf die Kennung auf dem Rechner nicht genutzt werden.

[5] CVS-SSH2 plug in von ymnk@jcraft.com, JCraft,Inc., Web-Site: `http://www.jcraft.com/eclipse-cvsssh2/` (online 8-Nov-2003)

add	Add a new file/directory to the repository
admin	Administration front end for rcs
annotate	Show last revision where each line was modified
checkout	Checkout sources for editing
commit	Check files into the repository
diff	Show differences between revisions
edit	Get ready to edit a watched file
editors	See who is editing a watched file
export	Export sources from CVS, similar to checkout
history	Show repository access history
import	Import sources into CVS, using vendor branches
init	Create a CVS repository if it doesn't exist
log	Print out history information for files
login	Prompt for password for authenticating server
logout	Removes entry in `.cvspass` for remote repository
pserver	Password server mode
rannotate	Show last revision where each line of module was modified
rdiff	Create 'patch' format diffs between releases
release	Indicate that a Module is no longer in use
remove	Remove an entry from the repository
rlog	Print out history information for a module
rtag	Add a symbolic tag to a module
server	Server mode
status	Display status information on checked out files
tag	Add a symbolic tag to checked out version of files
unedit	Undo an edit command
update	Bring work tree in sync with repository
version	Show current CVS version(s)
watch	Set watches
watchers	See who is watching a file

Legende: CVS (*Concurrent Versions System*) Version `1.11.1p1 (client/server)` — Quelle: `cvs -version`

Tabelle 8.1: CVS-Repository — Kommandos

Quellcode (↪ [Fowler99]). Die *Refactoring*-Option in Eclipse unterstützt beispielsweise das Umbenennen von Klassen sowie Änderungen der Klassenstruktur.

Wenn eine *Refactoring*-Operation veranlasst wird, kann man sich die Änderungen vorab anzeigen lassen ehe man über ihre Durchführung dann entscheidet. Auf Probleme dabei wird man weitgehend automatisch hingewiesen.

8.2 Einsatz von speziellen Werkzeugen

8.2.1 JUnit — *Unit Tests*

JUnit ist ein einfaches Rahmenwerk um wiederholbare Tests zu schreiben. JUnit ist eine Ausprägung der xUnit-Architektur. Sie hilft beim sogenannten Unit-Testen. In der Java-Welt handelt es sich primär um Tests der Methoden einer Klasse. JUnit, geschrieben von Erich Gamma and Kent Beck, ist von der Web-Site:

`http://www.junit.org/index.htm`
(online 10-Jun-2004)

als Open Source Software herunterladbar. In der ZIP-Datei[6] befindet sich die JAR-Datei `junit.jar`. Sie ist in den Klassenpfad einzufügen (siehe Protokolldatei `TestString.log`, S. 335).

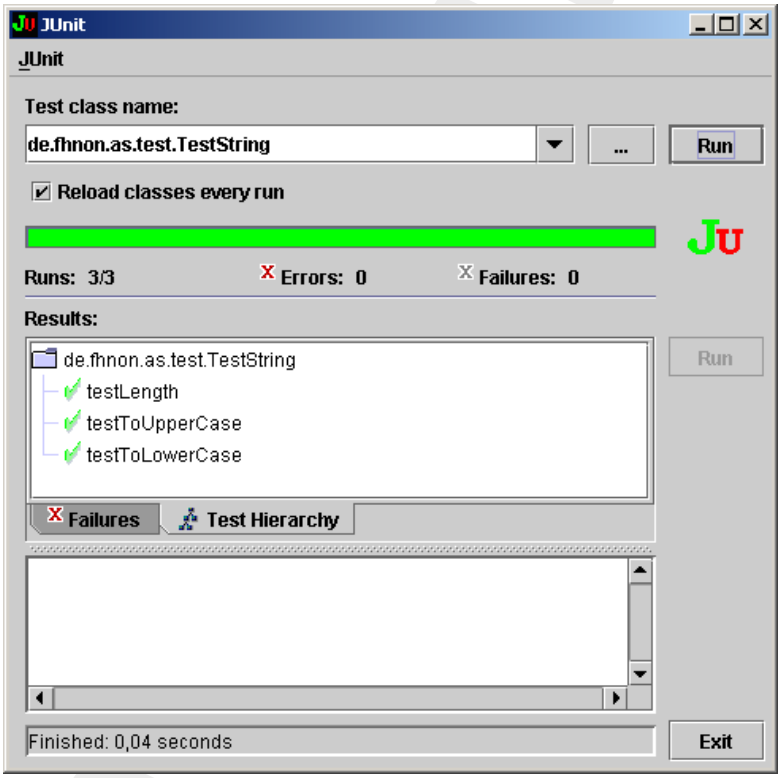
Im folgenden Beispiel wird exemplarisch für die vielfältigen JUnit-Möglichkeiten, diejenige genutzt, die alle Testmethoden, deren Name mit `test` beginnen, ausführen läßt. Wir testen hier die bekannten Mehoden `lenth()`, `toUpperCase()` und `toLowerCase()` der Klasse `String` aus dem Java-Paket `java.lang`.

Zunächst nutzen wir das einfache Interface auf Basis von Textausgaben in Kommandozeilen. Anschließend aktivieren wir das graphische Interface auf Basis von Swing (↪ `TestString.log` S. 335). Das Swing-Ergebnis zeigt Abbildung 8.3 S. 333.

Klasse `TestString`

```
/**
 *  JUint Beispiel: Test der Klasse java.lang.String
```

[6] hier `junit3.8.1.zip`



Legende:
JUnit von Kent Beck und Erich Gamma.

Abbildung 8.3: JUnit — graphische Komponente

```
 *
 *@author     Bonin
 *@version    1.0
 */

package de.fhnon.as.test;

import junit.framework.TestCase;
import junit.textui.TestRunner;
import junit.framework.TestSuite;

public class TestString extends TestCase
{
   static String myText =
         new String("Alles klar?");
   static int myLength = 11;


   public TestString(String name)
   {
      super(name);
   }


   public void testLength()
   {
      assertEquals(myText.length(), myLength);
   }


   public void testToUpperCase()
   {
      assertTrue(myText.toUpperCase().equals(
            new String("ALLES KLAR?")));
   }


   public void testToLowerCase()
   {
      assertTrue(myText.toLowerCase().equals(
            new String("alles klar?")));
   }


   static TestSuite suite() throws Exception
   {
```

```
      return new TestSuite(TestString.class);
   }


   public static void main(String[] args) throws Exception
   {
      TestRunner.run(TestString.suite());
   }
}
```

Protokolldatei `TestString.log`

```
D:\bonin\anwd\code>echo %CLASSPATH%
echo %CLASSPATH%
.;c:\programme\aspectj1.1\lib\aspectjrt.jar;
c:\programme\jdom-b10\build\jdom.jar;
c:\programme\junit3.8.1\junit.jar;
c:\programme\junit3.8.1\src.jar;

D:\bonin\anwd\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/as/test/TestString.java

D:\bonin\anwd\code>java de.fhnon.as.test.TestString
...
Time: 0,01

OK (3 tests)


D:\bonin\anwd\code>REM myLength = 13;

D:\bonin\anwd\code>javac de/fhnon/as/test/TestString.java

D:\bonin\anwd\code>java de.fhnon.as.test.TestString
.F..
Time: 0,01
There was 1 failure:
1) testLength(de.fhnon.as.test.TestString)
  junit.framework.AssertionFailedError:
```

```
    expected:<11> but was:<13>
at de.fhnon.as.test.TestString.testLength(
  TestString.java:28)
at sun.reflect.NativeMethodAccessorImpl.invoke0(
  Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(
  Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(
  Unknown Source)
at de.fhnon.as.test.TestString.main(
  TestString.java:54)

FAILURES!!!
Tests run: 3,  Failures: 1,  Errors: 0

D:\bonin\anwd\code>REM myLength = 11;

D:\bonin\anwd\code>java junit.swingui.TestRunner

D:\bonin\anwd\code>
```

Vorgehensweise für einen einfachen Unit-Test:

1. Schritt: Programmieren einer Unterklasse der Klasse `TestCase`, zum Beispiel:

   ```
   package de.fhnon.as.test;
   public class TestString extends TestCase
   { public TestString(String name)
        { super(name); }
   }
   ```

2. Schritt: Programmieren der Testmethoden mit den Zusicherungen für die erwarteten Ergebnisse, zum Biepiel:

   ```
   public void testLength()
     { assertEquals(myText.length(), myLength); }
   ```

3. Schritt: Programmieren einer `suite()`-Methode. Diese nutzt die *Reflection*-Option von Java um dynamisch eine Test-Suite zu erzeugen, die alle Methoden mit dem Namens-Präfix `testxxx` enthält. Zum Beispiel:

```
static TestSuite suite() throws Exception
  { return new TestSuite(TestString.class); }
```

4. Schritt: Programmieren der üblichen `main()`-Methode um den Test mit dem textuellen Interface durchführen zu können, zum Beispiel:

```
public static void main(String[] args)
  throws Exception
 { TestRunner.run(TestString.suite()); }
```

8.2.2 Ant — *Build Tool*

Apache Ant („Ameise“) ist ein Java-basiertes *Build Tool*. Es ist ein Open Source Produkt. Es ist von der Web-Site:

`http://ant.apache.org/`
(online 10-Jun-2004)

herunterladbar. Holzschnittartig betrachtet ist Ant eine Art von Make[7] ohne die „Runzeln“ (*wrinkles*) von Make. Während übliche Make-Werkzeuge sich auf Shell-Scripts stützen sowie mit diesen erweitert werden können und damit stets Betriebssytem bezogen sind, vermeidet Ant diese Betriebsystemabhängigkeit. Statt Shell-Kommandos erweitert Ant Java-Klassen. Die eigentliche Konfigurationsdatei ist im XML-Format. Sie spezifiziert einen Zielbaum in dem die einzelnen Aufgaben ausgeführt werden.

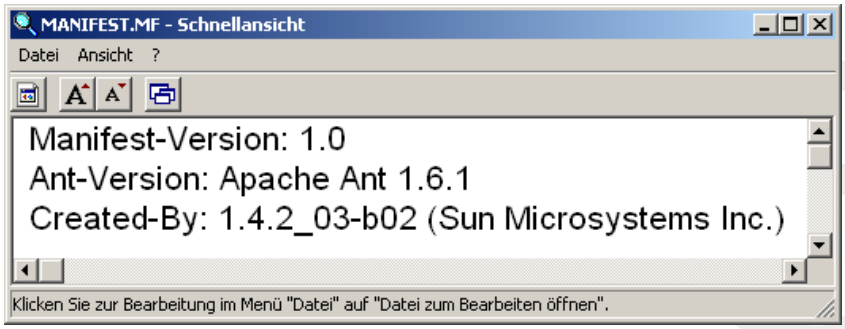
Zum Betrieb von Ant muss die Systemvariable `ANT_HOME` gesetzt sein und der `Pfad` im System auch das zutreffende Verzeichnis enthalten, zum Beispiel folgendermaßen:

```
ANT_HOME c:/programme/apache-ant-1.6.1
Path .;c:/programme/apache-ant-1.6.1/bin
```

Die Lauffähigkeit von Ant läßt sich wie folgt feststellen:

```
D:\bonin\anwd>ant -version
Apache Ant version 1.6.1
  compiled on February 12 2004
D:\bonin\anwd>
```

[7] Steht hier beispielhaft für `make`, `gnumake`, `nmake` oder `jam`



Legende:
Manifest in JAR-Datei `dist/lib/MyTestString-20040613.jar`

Abbildung 8.4: Mit Ant erzeugtes Mainfest

Als Beispiel verwenden wir die obige Klasse `TestString` (↪ S. 332). Das Kompilieren und das Bilden einer Java-Archiv-Datei (JAR) spezifizieren wir in der Datei `build.xml`. Das Ant-Ergebnis zeigt die Protokolldatei `Ant.log` (↪ S. 339). Das dabei erzeugte Manifest zeigt Abbildung 8.4 S. 338.

Datei `build.xml`

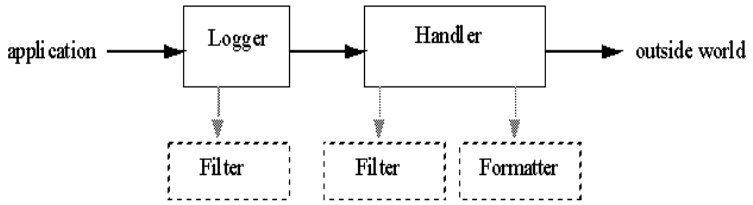
```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Bonin 12-Jun-2004  -->
<project name="MyTestString" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
  <!-- set global properties for this build -->
  <property name="src" location="de/fhnon/as/test"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure
      used by compile -->
```

```
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Compile the java code
      from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
        description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into
      the MyTestString-${DSTAMP}.jar file -->
    <jar jarfile=
      "${dist}/lib/MyTestString-${DSTAMP}.jar"
      basedir="${build}"/>
  </target>

  <target name="clean"
        description="clean up" >
    <!-- Delete the ${build} and ${dist}
      directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Protokolldatei `Ant.log`

```
D:\bonin\anwd\code>dir build.xml

1.266 build.xml

D:\bonin\anwd\code>ant -version
Apache Ant version 1.6.1 compiled on February 12 2004
D:\bonin\anwd\code>ant
Buildfile: build.xml

init:
    [mkdir] Created dir:
```

```
      D:\bonin\anwd\code\build

compile:
    [javac] Compiling 1 source file to
      D:\bonin\anwd\code\build

dist:
    [mkdir] Created dir: D:\bonin\anwd\code\dist\lib
      [jar] Building jar:
D:\bonin\anwd\code\dist\lib\MyTestString-20040612.jar

BUILD SUCCESSFUL
Total time: 2 seconds

D:\bonin\anwd\code>ant
Buildfile: build.xml

init:

compile:
    [javac] Compiling 1 source file to D:\bonin\anwd\code\build

dist:
      [jar] Building jar:
D:\bonin\anwd\code\dist\lib\MyTestString-20040613.jar

BUILD SUCCESSFUL
Total time: 2 seconds

D:\bonin\anwd\code>set classpath=%CLASSPATH%d:/bonin/anwd/
  code/dist/lib/MyTestString-20040612.jar;

D:\bonin\anwd\code>echo %CLASSPATH%
.;c:\programme\aspectj1.1\lib\aspectjrt.jar;
  c:\programme\jdom-b10\build\jdom.jar;
  c:\programme\junit3.8.1\junit.jar;
  c:\programme\junit3.8.1\src.jar;
  d:/bonin/anwd/code/dist/lib/MyTestString-20040612.jar;

D:\bonin\anwd\code>java de.fhnon.as.test.TestString
...
Time: 0

OK (3 tests)
```



Legende:
Quelle ↪ `http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html` (online 16-Jun-2004)

Abbildung 8.5: *Logging*-Übersicht — `java.util.logging`

```
D:\bonin\anwd\code>
```

8.2.3 Logging — `java.util.logging`

Eine Applikation macht *Logging*-Aufrufe bezogen auf `Logger`-Objekte. Diese `Logger`-Objekte sind *LogRecord*-Objekten zugeordnet, die einem `Handler`-Objekt zur Ausgabe übergeben werden. Beide, *Loggers* und *Handlers*, verwenden sogenannte *Logging Levels* und (optional) Filter mit denen entschieden wird, ob sie in einen bestimmten *LogRecord* kommen. Vor der *LogRecord*-Ausgabe (— in den I/O-Stream —) kann der *Handler* noch eine Formatierung (*Formatter*) durchführen. Die Abbildung 8.5 S. 341 skizziert diesen Zusammenhang.

Ein einfaches Beispiel für die Nutzung des Logging-Paketes

`java.util.logging`

stellt die Klasse `LogSample` (S. 341) dar. Die Ausgabe erfolgt dabei im XML-Format in die Datei `loggingResult.xml` (S. 342). Diese XML-Datei entspricht der *Document Type Definition* (DTD) `logger.dtd` (S. 343).

Klasse `LogSample`

```
/**
 *  Logging Example
 *
```

```
 *@author     Bonin
 *@version    1.0
 */

package de.fhnon.as.logging;

import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.FileHandler;
import java.io.IOException;

public class LogSample
{

   private static Logger logger =
         Logger.getLogger("de.fhnon.as.logging");
   private static String resultFile =
         "./de/fhnon/as/logging/loggingResult.xml";


   public String working()
   {
      return new String("Doing something!");
   }


   public static void main(String[] args)
          throws IOException
   {
      System.out.println(
            new LogSample().working());

      FileHandler fh =
            new FileHandler(resultFile);
      logger.addHandler(fh);
      logger.setLevel(Level.ALL);
      logger.log(Level.FINEST,
            "Testing - Level = FINEST");
   }
}
```

Datei `loggingResult.xml`

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
```

```
<log>
<record>
  <date>2004-06-16T09:22:09</date>
  <millis>1087370529034</millis>
  <sequence>0</sequence>
  <logger>de.fhnon.as.logging</logger>
  <level>FINEST</level>
  <class>de.fhnon.as.logging.LogSample</class>
  <method>main</method>
  <thread>10</thread>
  <message>Testing - Level = FINEST</message>
</record>
</log>
```

Datei `logger.dtd`

```
<?xml version="1.0" encoding="utf-8"?>

<!-- DTD used by the java.util.logging.XMLFormatter -->
<!-- This provides an XML formatted log message. -->


<!-- The document type is "log"
which consists of a sequence
of record elements -->
<!ELEMENT log (record*)>

<!-- Each logging call is
described by a record element. -->
<!ELEMENT record (date, millis, sequence, logger?, level,
class?, method?, thread?, message, key?, catalog?, param*,
exception?)>

<!-- Date and time when LogRecord was
created in ISO 8601 format -->
<!ELEMENT date (#PCDATA)>

<!-- Time when LogRecord was
created in milliseconds since
midnight January 1st, 1970, UTC. -->
<!ELEMENT millis (#PCDATA)>

<!-- Unique sequence number within source VM. -->
<!ELEMENT sequence (#PCDATA)>
```

```
<!-- Name of source Logger object. -->
<!ELEMENT logger (#PCDATA)>

<!-- Logging level, may be either one of the constant
names from java.util.logging.Constants (such as "SEVERE"
or "WARNING") or an integer value such as "20". -->
<!ELEMENT level (#PCDATA)>

<!-- Fully qualified name of class that issued
logging call, e.g. "javax.marsupial.Wombat". -->
<!ELEMENT class (#PCDATA)>

<!-- Name of method that issued logging call.
It may be either an unqualified method name such as
"fred" or it may include argument type information
in parenthesis, for example "fred(int,String)". -->
<!ELEMENT method (#PCDATA)>

<!-- Integer thread ID. -->
<!ELEMENT thread (#PCDATA)>

<!-- The message element contains the text string
of a log message. -->
<!ELEMENT message (#PCDATA)>

<!-- If the message string was localized, the key element provides
the original localization message key. -->
<!ELEMENT key (#PCDATA)>

<!-- If the message string was localized,
the catalog element provides
the logger's localization resource bundle name. -->
<!ELEMENT catalog (#PCDATA)>

<!-- If the message string was localized,
each of the param elements
provides the String value (obtained using Object.toString())
of the corresponding LogRecord parameter. -->
<!ELEMENT param (#PCDATA)>

<!-- An exception consists of an optional
message string followed
by a series of StackFrames. Exception elements are used
for Java exceptions and other java Throwables. -->
<!ELEMENT exception (message?, frame+)>
```

```
<!-- A frame describes one line in a Throwable backtrace. -->
<!ELEMENT frame (class, method, line?)>

<!-- an integer line number within a class's source file. -->
<!ELEMENT line (#PCDATA)>
```

Protokolldatei `LogSample.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/as/logging/LogSample.java

D:\bonin\anwd\code>java de.fhnon.as.logging.LogSample
Doing something!

D:\bonin\anwd\code>
```

8.3 Konventionen zur Transparenz

8.3.1 Code-Konventionen

Code-Konventionen sind aus einer Menge von Gründen bedeutsam[8]:

- Der wesentlich Anteil der Kosten eines Softwareproduktes (≈ 80%), die im gesamten Lebenszyklus des Produktes anfallen, verschlingt die Wartung (*Maintenance*).
- Der Schöpfer (ursprüngliche Programmierer) des Produktes pflegt nur in Ausnahmefällen das Produkt während der gesamten Lebenszeit.
- Code-Konventionen, wenn sie strikt eingehalten werden, verbessern die Durchschaubarkeit des Programms. Dritte können sich

[8] Die Empfehlungen sind weitgehend abgeleitet von den *Code Conventions for the Java™ Programming Language* Revised April 20, 1999, ↪ `http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html` (online 13-Oct-2003)

wesentlich schneller einfinden und das Programm leichter verstehen.

Allgemeinde Regeln

Suffixes Eine Java-Quellcodedatei hat grundsätzlich den Suffix `.java`. Eine Java-Bytecodedatei hat den Suffix `.class`.

Struktur

Reihenfolge Eine Java-Quellcodedatei ist wie folgt gegliederte:

1. Anfangskommentar
2. `package`-Statement
3. `import`-Statements
4. `class`-Deklaration(en)
5. `interface`-Deklaration(en)

Kommentar

Kommentare Der Anfangskommentar enthält Angaben zur Version, zum Autor und zum Copyright. Primär werden Kommentare zur Klärung der Frage „Warum“ und nicht zur Frage „Wie“ notiert. Kommentare verlängern die Quellcodedatei und müssen gepflegt werden. Ihre Reichweite ist nicht direkt ersichtlich, sondern sie ergibt sich aus ihrem Inhalt. Die Pflege wird aufwendig, weil bei einer Programmänderung quasi alle Kommentare überprüft werden müssen.

Groß/Kleinschreibung von Bezeichnern Ein Bezeichner ist ein Name, der eine Klasse, Methode, Parameter, Variable (Slot), Schnittstelle (Interface) oder ein Paket bezeichnet. Ein Bezeichner in Java^{TM} besteht aus einer beliebig langen Folge[9] von Unicode-Buchstaben und -Ziffern. Er beginnt mit einem Buchstaben oder einem Unterstrich (_) oder Dollarzeichen. Die Schreibweise von Bezeichnern erfolgt entsprechend der Regeln in Abschnitt 4.6 S. 68ff. Ein Paketname ist stets in kleinen Buchstaben definiert. Damit ergibt sich auch der entsprechende Pfad (*directories*) in kleinen Buchstaben.

[9] Die reservierten Schlüsselwörter (↪ Tabelle 6.4 S. 149) dürfen nicht als Bezeichner genutzt werden.

Deklarationen Eine Deklaration steht in einer eigenen Zeile, also zum Beispiel: **Deklaration**

```
int slot1; // first slot
int slot2; // second slot
```

Sie steht stets am Anfang eines Blocks, also zum Beispiel:

```
public Object myMethod()
{
    int i = 0;  // beginning of method block
    if (condition) {
        int j = 0; // beginning of "if" block
        ...
    }
    ...
}
```

Zu vermeiden sind lokale Deklarationen mit dem gleichen Namen von Deklarationen auf höherem Level, zum Beispiel:

```
int count;
...
public void myMethod()
{
    if (condition) {
        int count; // Vermeiden!
        ...
    }
    ...
}
```

Alternative Das `if-then-else`-Konstrukt ist wie folgt zu notieren: **if** . . . **then** . . . **else** . . .

```
if (condition) {
    ...
}
if (condition) {
    ...
```

```
} else {
    ...
}
if (condition1) {
    ...
} else if (condition2) {
    ...
} else {
    ...
}
```

Stets wird ein Block gebildet. Auch eine einzelne Anweisung wird aus Gründen der Fehlervermeidung in Klammern notiert.

switch

`switch`-**Konstrukt** Aus Gründen der Fehlervermeidung enthält jede Klausel enthält ein `break`-Konstrukt oder einen entsprechenden Kommentar, wie zum Beipiel:

```
switch (condition) {
case a:
    ...
    /* falls through */
case b:
    ...
    break;
case c:
    ...
    break;
default:
    ...
    break;
```

return

`return`-**Werte** Der Rückgabewert soll möglichst einfach direkt beim `return`-Konstrukt erkennbar sein, zum Beispiel nicht:

```
if (booleanExpression) {
    return true;
} else {
```

```
    return false;
}
```

sondern anstatt:

```
return booleanExpression;
```

Ein weiteres Negativbeispiel, also nicht so:

```
if (condition) {
    return x;
}
return y;
```

sondern besser:

```
return (condition ? x : y);
```

Java Source File Beispiel

Klasse `SourceFileExample`

```
/*
 *  @(#) SourceFileExample 1.3 12-Oct-2003
 *  Copyright (c) 2002-2003 Nemo, Inc.
 *  An der Eulenburg 6, D-21391 Reppenstedt, Germany
 */
package de.fhnon.as.codeconvention;
import java.awt.*;
/**
 *  Class JavaSourceFileExample description goes
 *  here.
 *
 *@author     Firstname Lastname
 *@created    12. Oktober 2003
 *@version    1.3 12-Oct-2003
 */
public class SourceFileExample extends Object
{
   /*
    *  A class implementation
    *  comment can go here.
    */
   /**
    *  Class variable1 documentation comment
```

```
 */
public static final int CLASSVAR1;

/*
 *  Class variable2 documentation comment
 */
private static int classVar2;
/*
 *  Instance variable 1 documentation comment
 */
public Object instanceVar1;
/*
 *  Instance variable 2 documentation comment
 */
protected Object instanceVar2;
/*
 *  Instance variable 3 documentation comment
 */
private Object instanceVar3;


/**
 *  Getter description
 *
 *@return    The instanceVar1 value
 */
public Object getInstanceVar1()
{
   return instanceVar1;
}


/**
 *  Setter description
 *
 *@param  instanceVar1  The new value of
 *      instanceVar1 value
 */
public void setInstanceVar1(Object instanceVar1)
{
   this.instanceVar1 = instanceVar1;
}


/**
 *  Constructor SourceFileExample documentation
```

```
     *  comment
     */
    public SourceFileExample() { }


    /**
     *  Method doSomething() documentation comment
     *
     *@param  parameter1  description
     *@param  parameter2  description
     *@return             returnValue description
     */
    public boolean doSomething(
          Object parameter1, Object parameter2)
    {
       // ... implementation goes here ...
       return true;
    }


    /**
     *  Method main() documentation comment
     *
     *@param  args  The command line arguments
     */

    public static void main(String[] args)
    {
       // ... implementation goes here ...
       System.out.println("End of SourceFileExample");
    }

}
```

8.3.2 Tipps zur Kodierung

`javadoc` nutzen — Bezeichner überlegen

Die Wahl der Bezeichner sollte wohl überlegt werden. Es gilt möglichst viel Semantik einzubauen. Auch die Reihenfolge der gewählten Begriffe ist bedeutsam. Es ist beispielsweise nicht „egal“ ob man eine Klasse `TestString` oder `StringTest` nennt. Will man sich auf die Thematik des Testen konzentrieren, dann wählt man die erste Lösung, wie

hier im Abschnitt 104 S. 332. Hat man ein konkretes Projekt, dann ist in der Regel die zweite Lösung zweckmäßiger. Mit einem Muster Begriff-BetroffeneKlasseTest läßt sich besser überschauen, welche Klassen bereits durch Testklassen abgedeckt sind, weil im Pfad zu der jeweiligen Klasse dann gleich die Testklasse folgt. Die Zusammenfassung in Blöcke durch eine alphabetische Sortierung ist bei Wahl der Bezeichner zu bedenken.

Die Dokumentation im Quellcode ist entsprechend den Regeln von `javadoc` zu gestalten. Dabei ist zu beachten, dass `javadoc` leider nicht alle gültigen Java-Konstrukte verarbeiten kann. Im folgenden Beispiel ist es das `assert`-Konstrukt. Um trotzdem eine Dokumentation zu generieren, wird vor einer Anwendung von `javadoc` dieses Konstrukt auskommentiert (↪ Protokolldatei 8.3.2 S. 361). Die Abbildung 8.6 S. 363 zeigt einen Ergebnisausschnitt.

Anhand des folgenden Beispiels einer rekursiv definierten Funktion werden die Quellecodeunterschiede verdeutlicht.

$$g = \begin{cases} 0 & : \quad x \leq 0 \\ \frac{a}{\frac{b}{2}} * x & : \quad 0 < x \leq \frac{b}{2} \\ -\frac{a}{\frac{b}{2}} * x + 2a & : \quad \frac{b}{2} < x \leq b \\ g(x-b) & : \quad b < x \end{cases}$$

Die schnelle, scheinbar sehr transparente Lösung mit der Klasse `F` (↪ S. 352) berücksichtigt nicht die Bedingung, dass die Werte für `a` und `b` größer als null sein sollten, wenn die Funktion eine nach oben zeigende „Säge“ (engl. `saw`) abbilden soll. Ein Aufruf `g(4.5, -2.0)` führt zu einem nicht abgefangenen Fehler (`java.lang.StackOverflowError`).

Klasse `F` — So nicht!

```
package de.fhnon.as.function;

public abstract class F
{
   public static double g(double x, double a, double b)
   {

      if (x <= 0.0)
      {
```

```
            return 0.0;
         } else if (x <= b / 2)
         {
            return (a / (b / 2)) * x;
         } else if (x <= b)
         {
            return (-a / (b / 2)) * x + 2 * a;
         } else
         {
            return g(x - b, a, b);
         }

      }


      public static double g(double x)
      {
         return g(x, 1.0, 2 * java.lang.Math.PI);
      }


      public static double g(double x, double a)
      {
         return g(x, a, 2 * java.lang.Math.PI);
      }


      public static void main(String[] args)
      {
         System.out.println(g(4.5) + "\n" +
               g(4.5, 2.0, 2.0));
      }
}
```

Die Aufteilung in die folgenden drei Klassen ist zweckmäßig:

- `Math` (↪ S. 357) bildet prinzipiell die obige Funktion `g` (↪ S. 352) ab.
  Der Methodenname `saw` und die Parameternamen `toothHeight` und `toothWidth` vermitteln ihre Bedeutung unmittelbar — haben also mehr Semantik. Die Tabelle 8.2 S. 356 ist daher der bessere Ausgangspunkt.

- `MathValue` enthält die Prüfmethoden und die Ersatzwerte bei

fehlenden Angaben (↪ S. 359).

- `MathProg` enthält die Testbeispiele (↪ S. 360).

Nützlich ist die schriftliche Dokumentierung der Anforderungen (*Requirements*), die von der Software, hier primär von der Metode `saw(x)`, erfüllt werden. Die Requirements werden mit einem eindeutigen Identifier, zum Beispiel Rn, notiert. Im Quellcode bilden diese Identifier dann Verknüpfungspunkte (*Links*) zur Dokumentation. Bei der Formulierung der Requirements wird unterstellt, dass die zu beschreibende Software schon existiert. Man formuliert daher nicht in der Art das Softwareprodukt „`XYZ` wird, soll, könnte usw.“, sondern „`XYZ` tut, leistet, bewirkt usw.“. So kann man später in allen Phasen des Lebenszykluses der Software diesen ersten Text über die Requirements direkt, also ohne fehleranfällige Umformulierungen, nutzen (sogenannte *Cut&Paste*-Technik).

Requirements für `Math.saw(x)`:

`R01` `Math.saw(x)` berechnet die Zahnhöhe an der Stelle `x` nach der Formel in Tabelle 8.2 S. 356.

`R02` Der Parameter `toothHeight` erhält seinen Wert:

- `R02.1` über den Aufruf in der Form: `saw(x, toothHeight)` oder `saw(x, toothHeight, toothWidth)`
- `R02.2` oder als Konstante `defaultToothHeight` aus der Klasse `MathValue`, gesetzt auf 1.0.

`R03` Der Parameter `toothWidth` erhält seinen Wert:

- `R03.1` über den Aufruf in der Form: `saw(x, toothHeight, toothWidth)`
- `R03.2` oder als Konstante `defaultToothWidth` aus der Klasse `MathValue`, gesetzt auf $2 * \pi$.

`R04` Die Werte der Parameter `toothHeight` und `toothWidth` müssen ≥ 0 sein. Ist das nicht der Fall, ist die Zahnhöhe `NaN`, als *Not a Number*.

`R05` Tritt ein Fehler bei der Berechnung auf, weil für die rekursive Formel in Tabelle 8.2 S. 356 der Rechner zu wenig Speicherplatz hat, ist die Zahnhöhe `NaN`, als *Not a Number*.

$$saw(x) = \begin{cases} 0 & : \quad x \leq 0 \\ \frac{toothHight}{\frac{toothWidth}{2}} * x & : \quad 0 < x \leq \frac{toothWidth}{2} \\ -\frac{toothHight}{\frac{toothWidth}{2}} * x + 2 * toothHight & : \quad \frac{toothWidth}{2} < x \leq toothWidth \\ saw(x - toothWidth) & : \quad toothWidth < x \end{cases}$$

Tabelle 8.2: `Math.saw(x)`-Formel

Klasse `Math`

```
/**
 *  <code>Math</code> class with the recursive specified
 *  <code>saw</code> method The saw has some tooth. A tooth
 *  is symetric construct. It has a height and a width. The
 *  method computes the height at a tooth position. The
 *  default values for tooth height and tooth width contains
 *  the class <code>MathValue</code>
 *
 *@author     Bonin 6-Apr-2004
 *@version    1.0
 */
package de.fhnon.as.function;

public abstract class Math
{
   /**
    *  <code>saw</code> computes the value of the height at
    *  a tooth position x
    *
    *@param  x           position of the saw from 0.0 to
    *       ...
    *@param  toothHeight  should be greater zero
    *@param  toothWidth   should be greater zero
    *@return              height value at the position x. If
    *      impossible, it returns "not a number" (<code>NaN</code>
    *      ).
    */
   public static double saw(
         double x,
         double toothHeight,
         double toothWidth)
   {
      /*
       *  implements R01, R02.1, R03.1, R05
       */
      try
      {

         if (!MathValue.assertGreaterZero(toothHeight) ||
               !MathValue.assertGreaterZero(toothWidth))
         {
            return Double.NaN;
         }

         if (x <= 0.0)
         {
            return 0.0;
         } else if (x <= toothWidth / 2)
         {
            return (toothHeight / (toothWidth / 2)) * x;
         } else if (x <= toothWidth)
         {
```

```
            return (-toothHeight / (toothWidth / 2)) * x +
                  2 * toothHeight;
         } else if (toothWidth < x)
         {
            return saw(x - toothWidth, toothHeight, toothWidth);
         } else
         {
            return Double.NaN;
         }
      } catch (java.lang.StackOverflowError eS)
      {
         System.out.println(eS + " x: " + x + "\n");
         return Double.NaN;
      }
   }


   /**
    *  <code>saw</code> computes the value of the height at
    *  a tooth position x
    *
    *@param  x  position of the saw from 0.0 to ...
    *@return    height value at the position x. If
    *      impossible, it returns "not a number" (<code>NaN</code>
    *      ).
    */
   public static double saw(double x)
   {
      /*
       *  implents  R02.2, R03.2
       */
      return saw(
            x,
            MathValue.defaultToothHeight,
            MathValue.defaultToothWidth);
   }


   /**
    *  <code>saw</code> computes the value of the height at
    *  a tooth position x
    *
    *@param  x           position of the saw from 0.0 to
    *      ...
    *@param  toothHeight  should be greater zero
    *@return              height value at the position x. If
    *      impossible, it returns "not a number" (<code>NaN</code>
    *      ).
    */
   public static double saw(double x, double toothHeight)
   {
      /*
       *  implements R03.2
```

```
       */
      return saw(
            x,
            toothHeight,
            MathValue.defaultToothWidth);
   }

}
```

Klasse `MathValue`

```
/**
 *  <code>MathValue</code> class contains the default values
 *  for the class <code>Math</code> and a method to validate
 *  the parameter values for <code>Math.saege(...)</code>
 *
 *@author     Bonin 6-Apr-2004
 *@version    1.0
 */

package de.fhnon.as.function;

public abstract class MathValue
{

   final static double defaultToothHeight = 1.0;

   final static double defaultToothWidth = 2 * java.lang.Math.PI;



   /**
    *  <code>assertGreaterZero</code> checks the value of
    *  its parameter. by using the <code>assert</code>
    *  construct
    *
    *@param  value  will bee checked if it is <code>>= 0.0</code>
    *@return        <code>true</code> or <code>false</code>
    */

   protected static boolean assertGreaterZero(double value)
   {
      /*
       *  implements R04
       */
      try
      {

        assert value >= 0.0;

      } catch (java.lang.AssertionError eA)
```

```
      {
         System.err.println(eA + " value: " + value);
         return false;
      }
      return (value >= 0.0 ? true : false);
   }
}
```

Klasse `MathProg`

```
/**
 *  <code>MathProg</code> class contains test cases for
 *  <code>Math.saw(...)</code>
 *
 *@author     Bonin 6-Apr-2004
 *@version    1.0
 */

package de.fhnon.as.function;

public class MathProg
{

   /**
    *@param  args  are not used
    */

   public static void main(String[] args)
   {
      // x is to great --> stack overflow
      System.out.println(Math.saw(1000000.0));

      // values are OK!
      System.out.println(
            Math.saw(4.5) + "\n" +
            Math.saw(4.5, 2.0, 2.0));

      // negative toothHeight and toothWidth
      System.out.println(Math.saw(4.5, -2.0) + "\n" +
            Math.saw(4.5, 2.0, -3.0));

   }

}
```

Protokolldatei `MathProg.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\anwd\code>javac -source 1.4
                    de/fhnon/as/function/*.java

D:\bonin\anwd\code>java de.fhnon.as.function.F
0.567605512172942
1.0

D:\bonin\anwd\code>java -enableassertions
                    de.fhnon.as.function.MathProg
java.lang.StackOverflowError x: 972542.48020767

NaN
0.567605512172942
1.0
java.lang.AssertionError value: -2.0
java.lang.AssertionError value: -3.0
NaN
NaN

D:\bonin\anwd\code>java -disableassertions
                    de.fhnon.as.function.MathProg
java.lang.StackOverflowError x: 972542.48020767

NaN
0.567605512172942
1.0
NaN
NaN

D:\bonin\anwd\code>javadoc de/fhnon/as/function/*.java
javadoc de/fhnon/as/function/*.java
Loading source file de/fhnon/as/function/F.java...
Loading source file de/fhnon/as/function/Math.java...
Loading source file de/fhnon/as/function/MathProg.java...
Loading source file de/fhnon/as/function/MathValue.java...
de/fhnon/as/function/MathValue.java:36:
  warning: as of release 1.4, assert is a keyword,
  and may not be used as an identifier
        assert value >= 0.0;
        ^
de/fhnon/as/function/MathValue.java:36: ';' expected
        assert value >= 0.0;
                     ^
1 error
1 warning
```

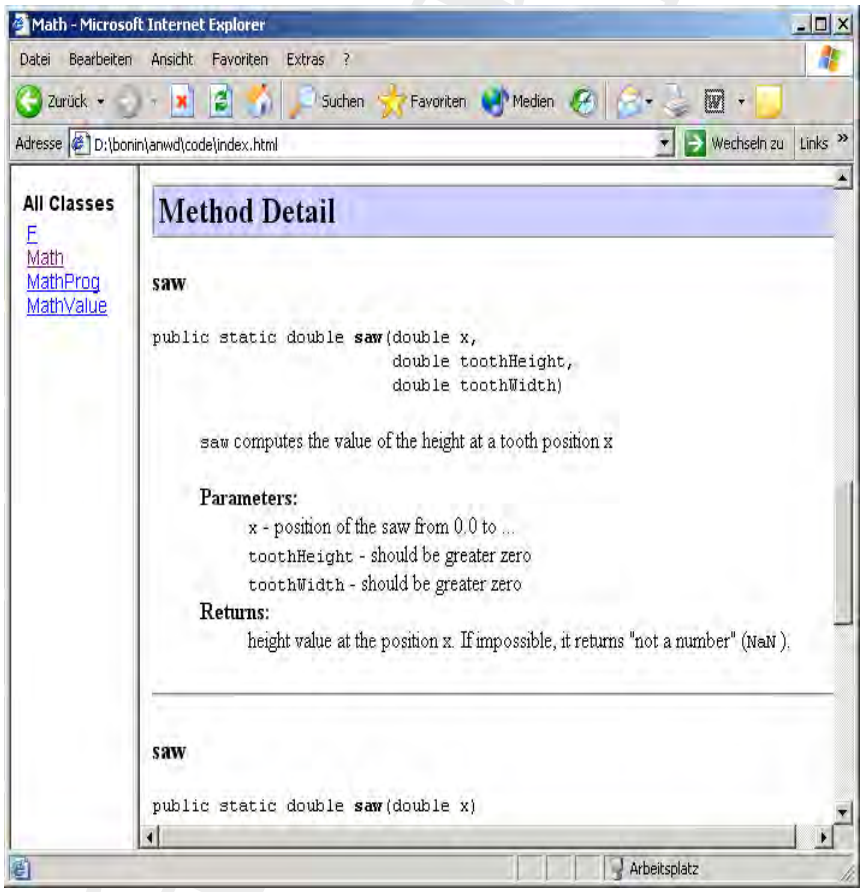
```
D:\bonin\anwd\code>javadoc de/fhnon/as/function/*.java
javadoc de/fhnon/as/function/*.java
Loading source file de/fhnon/as/function/F.java...
Loading source file de/fhnon/as/function/Math.java...
Loading source file de/fhnon/as/function/MathProg.java...
Loading source file de/fhnon/as/function/MathValue.java...
Constructing Javadoc information...
Standard Doclet version 1.4.2_03
Generating constant-values.html...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating packages.html...
Generating de\fhnon\as\function\package-frame.html...
Generating de\fhnon\as\function\package-summary.html...
Generating de\fhnon\as\function\package-tree.html...
Generating de\fhnon\as\function\F.html...
Generating de\fhnon\as\function\Math.html...
Generating de\fhnon\as\function\MathProg.html...
Generating de\fhnon\as\function\MathValue.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...

D:\bonin\anwd\code>
```

Begrenzung der Reichweite

Stets sollte man sich präzise die Reichweite seiner Konstrukte überlegen. Die Reichweite ist möglichst auf das zwingend notwendige Maß zu begrenzen. Dazu eignen sich zusätzliche Blöcke, also {...}-Konstrukte, sowie das *Inner-Class*-Konzept (↪ Abschnitt 7.4 S. 191 und z. B. Übungsaufgabe A.19 S. 453). Die folgende Beispielklasse `Reichweite` skizziert solche Begrenzungen der Reichweiten mit Blöcken in der `main()`-Methode und mit zwei lokalen Klassen `Work` im *if-then-else*-Konstrukt der Methode `internalWork()`.

Klasse `Reichweite`

```
/**
 *  Beispiel: Reichweite durchdenken Mittel: Block
```



Legende:
Dargestellt mit *Microsoft Internet Explorer*, Version 6.0.2600, auf einer Windows-XP-Plattform.

Abbildung 8.6: Dokumentation mittels `javadoc`

```
 *  und Innere Klasse
 *
 *@author     Hinrich Bonin
 *@version    1.0 18-Mar-2004
 */

package de.fhnon.scope;

public class Reichweite
{
   private void internalWork(boolean working)
   {
      if (working)
      {
         class Work
         {
            private String id;


            Work(String s)
            {
               id = s;
            }


            String getId()
            {
               return id;
            }


            void setId(String s)
            {
               id = s;
            }
         }
         Work w = new Work("Emma");
         w.setId(w.getId() + " " + "Mustermann");
         System.out.println(w.getId());
      } else
      {
         class Work
         {
            String id = "Nothing to do!";
```

```
            String text()
            {
               return "No Working: ";
            }
         }
         System.out.println((new Work()).id);
         System.out.println((new Work()).text() + "\n" +
               "  No Specifikation Work & no Objekt w");
         /*
          *  Compiler error if something like
          *  Work  foo = new Work("Nemo");
          */
      }
   }

   public static void main(String[] args)
   {
      {
         /*
          *  Block 1 --- the scope of object r1
          */
         Reichweite r1 = new Reichweite();
         r1.internalWork(true);
      }
      {
         /*
          *  Block 2 --- the scope of object r2
          */
         Reichweite r2 = new Reichweite();
         r2.internalWork(false);
      }
      {
         /*
          *  Block 3 --- no r1 & r2 reachable
          *  For Example:
          *  Compile error for the following statement
          *  r2.internalWork(false);
          */
      }

   }
}
```

Protokolldatei `Reichweite.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/scope/Reichweite.java

D:\bonin\anwd\code>java de.fhnon.scope.Reichweite
Emma Mustermann
Nothing to do!
No Working:
  No Specifikation Work & no Objekt w

D:\bonin\anwd\code>cd de/fhnon/scope

D:\bonin\anwd\code\de\fhnon\scope>dir Reichweite*
   595 Reichweite$1Work.class
   529 Reichweite$2Work.class
 1.208 Reichweite.class
 1.892 Reichweite.java

D:\bonin\anwd\code\de\fhnon\scope>
```

Konstante statt Pseudo-Variable ↪ Performance

Immer wenn unstrittig feststeht, daß ein Wert für alle Objekte gleich bleibt, sich also garantiert nicht ändert, ist eine Konstante statt einer Variablen zu wählen. Die Angabe der Modifieren `static` und `final` ermöglicht dem Compiler einen wesentlich effizienteren Code zu erzeugen.

Langsame Lösung

```
String myString = "Bleibt immer so!";
```

Effizientere Lösung

```
static final String myString = "Bleibt immer so!";
```

Anordnen von Ausnahmen (*Exceptions*)

Sind mehrere Ausnahmen zu programmieren, dann stellt sich die Frage ihrer Anordnung. Die Praxis, jeden Methodenaufruf, der eine Ausnahme bewirken kann, in ein eigenes `try-catch`-Konstrukt einzuschließen, macht den Quellcode schwer durchschaubar. Zusätzlich erschwert es eine Ablaufoptimierung des Compilers. Es ist daher besser, in einem größeren `try`-Block die Methodenaufrufe zusammen zu fassen und die `catch`-Blöcke danach zu notieren.

Schwer durchschaubare `try-catch`-Anordnung

```
private void irgendEtwas() {
  try {
    foo.methodA();
  }
  catch (methodAException eA) {
    // Code zur Behandlung der Ausnahme eA
  }
  try {
    foo.methodB();
  }
  catch (methodBException eB) {
    // Code zur Behandlung der Ausnahme eB
  }
  try {
    foo.methodC();
  }
  catch (methodCException eC) {
    // Code zur Behandlung der Ausnahme eC
  }
}
```

Bessere `try-catch`-Anordnung

```
private void irgendEtwas() {
  try {
    foo.methodA();
```

```
    foo.methodB();
    foo.methodC();
  }
  catch (methodAException eA) {
    // Code zur Behandlung der Ausnahme eA
  }
  catch (methodBException eB) {
    // Code zur Behandlung der Ausnahme eB
  }
  catch (methodCException eC) {
    // Code zur Behandlung der Ausnahme eC
  }
}
```

Ersatz durch `throws`-Konstrukt In manchen Fällen kann das `try-catch`-Konstrukt durch `throws` ersetzt werden. Die Aufgabe wird dann dem *Caller* übertragen.

```
private void irgendEtwas()
  throws
    methodAException,
    methodBException,
    methodCException {
  foo.methodA();
  foo.methodB();
  foo.methodC();
}
```

Zeichenmodifikationen mit `StringBuffer`

Weil ein Objekt vom Typ `String` per Definition nicht änderbar (*immutable*) ist, wird bei jeder Modifikation ein neues String-Objekt erzeugt. Die Zwischenresulate bei mehreren Manipulationen sind dann alles Objekte, deren Speicherplatz wieder freizugeben ist, also Arbeit für den *Garbage Collector*. Der `StringBuffer` ist ein modifizierbares Objekt. Er sollte daher stets benutzt werden, wenn viele Manipulationen an einer Zeichenkette erforderlich sind.

Aus dem gleichen Grund sollte auch eine Konstruktion mit `StringBuffer` und `append` einer Konstruktion mit dem Konstrukt „+“ vorgezogen werden.

Ausreichende Lösung

```
String myString = "Alles";
String klar     = "klar?";
myString += " ";
myString += klar;
```

Gute Lösung

```
StringBuffer myBuffer = new StringBuffer(11);
myBuffer.append("Alles ");
myBuffer.append("klar?");
String myString = myBuffer.toString();
```

Sehr gute Lösung

```
String myString = "Alles" + " " + "klar?";
```

da vom Compiler in denselben Bytecode verwandelt wie (↪ [IBM-Francisco98]):

```
String myString = "Alles klar?";
```

8.3.3 Rahmen für Geschäftsobjekte und -prozesse

„Es gibt wenige Frameworks,
die wirklich funktionieren,
. . . die Zahl der unvollendeten,
vorzeitig verschrotteten Frameworks ist Legion.“
(↪ [Broy/Siedersleben02] S. 58)

Francisco

Das *San Francisco Project*[10] der IBM Corporation stellt bewährte „Musterobjekte“ (*Common Business Objects*) und „Musterprozesse“ (*Core Business Processes*) für verschiedene Anwendungsfelder in der kommerziellen Datenverarbeitung bereit. Dazu zählen zum Beispiel:

[10] Aktuelle Informationen zum San Francisco Projekt der IMB Corporation: `http://www.ibm.com/java/sanfrancisco` (Zugriff:08-Jul-1998)

Objekte

- Geschäftsobjekte (*Common Business Objects*):
  - Adresse
  - Geschäftspartner (Kunde, Lieferant)
  - Kalender (zum Beispiel perioden-basiert)
  - Indentifierungs-Serien für Dokumente, Konten usw.
  - Währungen

Prozesse

- Geschäftsvorgänge (*Core Business Processes*):
  - Zahlungsverkehr, Finanzwesen (*Business Financials*))
  - Verkaufs- und Angebotsverwaltung (*Order Management*
  - Empfangen und Versenden von Waren *Warehouse Management*

Solche gut getesteten *Common Business Objects* und *Core Business Processes* können direkt in den Quellcode einer Java-Anwendung importiert werden. Sie bilden die eigentliche Basis für den Entwickler. Die Java-Konstrukte des `J2SE SDK`s dienen nur noch als „Mörtel“ für das Zusammenpassen der vorgefertigten Bausteine.

Hinweis: Die IBM hat inzwischen die Arbeiten am *San Francisco Project* eingestellt.

Es gibt eine Vielzahl kritischer Stimmen zur Frage der generellen Machbarkeit von solchen Ansätzen. Heute läßt sich (noch ?) feststellen: „. . . in summa bisher keineswegs die versprochene Linderung eines “Ewigkeitsproblems” der Wirtschaftsinformatik, der inner- und zwischenbetrieblichen Integration heterogener IV-Systeme, gebracht.“(↪ [Hau/Mertens02] S. 339).

Kapitel 9

Dokumentieren mit HTML

HTML (*HyperTextMarkup Language*) ist das Esperanto im Web. Hervoragend ist die Auspägung XHMTL (*Extensible* HTML) für das Dokumentieren eines Softwaresystems geeignet. Auf einfache Art und Weise sind verschiedene Dokumenttypen (zum Beispiel: Modell-, Test- und Quellcode-Dokumenten) verknüpfbar.
Mit dem CSS-Möglichkeiten (*Cascading Style Sheets*) können die vielen Dokumente aus den verschiednenen Quellen (Entwicklungswerkzeugen) einheitlichen und damit leichter überschaubar präsentiert werden.

Trainingsplan

Das Kapitel „Dokumentation mit HTML“ erläutert:

- Möglichkeiten von XHTML und
  ↪ Seite 372 . . .
- die Layout-Gestaltung mit Hilfe von *Cascading Style Sheets* (CSS).
  ↪ Seite 376 . . .

9.1 XHTML

XHTML, die XML-konforme Spezifikation von HTML Version 4.0, ergänzt die ursprünglichen, einfachen HTML-Konstrukten (↪ Versionen 1 & 2) um vielfältige Mechanismen zur Softwaredokumentation. Zu nennen sind hier beispielsweise:

- einheitliche Layoutgestaltung über mehrere Dokumente (*Cascading Style Sheet* (CSS) (↪ Abschnitt 9.2 S. 376)
- Einbindung von Script-Sprachen (*Scripting* — zum Beispiel Javascript)
- Bildflächenaufteilung (*Frames*)
- Integrierte Objekte (*Embedding Objects*)
- Maskengestaltung (*Forms*)
- geschachtelte Tabellen (*Tables*)
- Textausrichtung nach rechts, links, mittig.

Solche Gestaltungsmöglichkeiten gab es schon ansatzweise in der HTML Version 3.2 und/oder durch die browser-spezifischen Konstrukte von *Netscape Communications Corporation* und *Microsoft*. Mit XHTML sind die Mechanismen als SGML[1]-Konstrukte strikter definiert und in ein konsistentes Gesamtkonzept integriert.

Diese vielfältigen Möglichkeiten können hier nicht dargestellt werden. Präzise Beschreibungen sind den von W3C publizierten Spezifikationen (zum Beispiel HTML 4.0 Spezifikation ↪ [HTML4.0]) entnehmbar. Im folgenden wird nur das CSS-Konzept behandelt. Es ermöglicht, Daten aus verschiedenen Quellen im Softwareerstellungsprozeß einheitlich zu präsentieren. Für die Teamarbeit besteht damit eine leicht umsetzbare Konvention für das Definieren und Einhalten eines gemeinsamen Projekt-Layouts.

Das folgende Beispiel `index.php3` soll daher nur einige XHTML--Optionen skizzieren.

PHP-Datei `index.php3`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Bonin 09-Feb-2000 ...  21-Nov-2003                -->
<!-- Hinweis: Kein xml-Prolog weil PHP Fehler meldet    -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head>
<link rel="shortcut icon" href="/as111.ico" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="robots" content="index,follow" />
<meta http-equiv="Last-Modified" content="21-Nov-2003 13:00:00 GMT" />
<meta http-equiv="Expires" content="31-Dec-2010 00:00:00 GMT" />
<meta name="DESCRIPTION" content="Bonin's Hompepage" />
<meta name="KEYWORDS"
  content="Wirtschaftsinformatik,Web-Technologie,Aspect-Oriented Programming,
    Aspect-Oriented Softwaredevelopment,Verwaltungsinformatik" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<base href="http://as.fhnon.de/" />
<link href="/myStyle.css" rel="stylesheet" type="text/css" />
<link rev="owns" title="Hinrich E.G. Bonin" href="mailto:bonin@fhnon.de" />
<script type="text/javascript">
<!--
  var browserName    = navigator.appName;
  var browserVersion = parseInt(navigator.appVersion);
```

[1] SGML ≡ *Standard Generalized Markup Language*, ISO-Standard 8879:1986

```
  var checkBrowser    = "NichtOK";
  if ((browserName == "Netscape" && browserVersion >= 3) ||
    (browserName.indexOf('Explorer') != -1 && browserVersion >=4))
    {
    checkBrowser = "OK";
    }
  function goMyWeb(){
    if (checkBrowser == 'OK')
      setTimeout('top.location="/content.html"',10000);
  }
//
// Hinweis: Alte Browser koennen nicht:
//    script type="text/javascript" src="/myScript.js"
//  und nicht XHTML-konform
//    ![CDATA[ ... ]]
//-->
</script>
<title>as: applied sciences / aspect-oriented software</title>
</head>
<body onload="goMyWeb()">
<p class="klein">
<?php
  if(strstr(date("a"),"am")) {
    echo "Morning, ";
   } else {
    echo "Greetings, ";
   }
?>dear
 <?php require("tcounter.php3");
    echo tcounter();
    echo ". visitor to my home page! You are " ;
    if(strstr($REMOTE_ADDR,"193.174.32.")) {
         echo "in the Computer Center of the University ... . ";
     } elseif(strstr($REMOTE_ADDR,"193.174.33.")) {
         echo "in the Department of Business Administration ... . ";
     } elseif(strstr($REMOTE_ADDR,"193.174.20.")) {
         echo "in the University of Applied Sciences at ... . ";
     } elseif(strstr($REMOTE_ADDR,"193.174.21.")) {
         echo "in the  University of Applied Sciences at ... . ";
     } elseif(strstr($REMOTE_ADDR,"141.39.217.")) {
         echo "in the new building of the University of ... . ";
    } elseif(strstr($REMOTE_ADDR,"141.39.209.") Or
             strstr($REMOTE_ADDR,"141.39.210.") Or
             strstr($REMOTE_ADDR,"141.39.211.") Or
             strstr($REMOTE_ADDR,"141.39.212.") Or
             strstr($REMOTE_ADDR,"195.37.24.") ) {
```

```
          echo "at the campus of the University of Lüneburg! ";
      }
      else {
      echo "an Internet-User! ";
      }
  ?>
Your computer "<?php echo gethostbyaddr($REMOTE_ADDR);?>"
uses the IP number: <?php echo $REMOTE_ADDR;?>.
  <?php
     if(strstr($HTTP_USER_AGENT,"MSIE")) {
       echo "Ok, ";
          echo $HTTP_USER_AGENT;
          echo " is your browser. ";
          } else {
          echo "You are using the browser: ";
          echo $HTTP_USER_AGENT;
          echo ". ";
          }
      ?>
Your browser has used the port: <?php echo $REMOTE_PORT;?> with
the method <?php echo $REQUEST_METHOD;?> and
the protocol <?php echo $SERVER_PROTOCOL ?>. My web server
"<?php echo $HTTP_HOST;?>"
runs the software:
<?php echo $SERVER_SOFTWARE;?>
and uses the port:
<?php echo $SERVER_PORT;?>.
My web server computer is a IBM AIX RS/6000 and has the IP number:
<?php echo $SERVER_ADDR;?>
</p>
<hr />
<h1><img src="as.gif" alt="explanation of as" /></h1>
<h1><a href="content.html">Main port: content.html</a></h1>
<h3><a href="http://nemo.fhnon.de">http://nemo/fhnon.de</a></h3>
<h1><a href="/politik/global.html">
  Globalisierung &amp; Machtk&auml;mpfe</a></h1>
<h3><a href="/content.html">
  Prof. Dr.rer.publ. Dipl.-Ing. Dipl.-Wirtsch-Ing. Hinrich E. G. Bonin</a>;
email: <a href="mailto:bonin@fhnon.de">bonin@fhnon.de</a></h3>
<h3><a href="http://www.fhnon.de">University of
  Applied Sciences in North-East-Lower-Saxony (Germany)</a>
  (<a href="http://www.fhnon.de">
    Fachhochschule Nordostniedersachsen</a> in
 <a href="http://www.lueneburg.de/index.html">
   L&uuml;neburg</a>)</h3>
<hr />
```

```
<h3><a href="content.html">
  <?php echo date("l dS of F Y G:i:s");?></a> ---
  <a href="http://mailman.fhnon.de/mailman/mailman.cgi">
    fhnon web mail</a></h3>
<h2>
<a href="/lebenslauf.html"><img
  src="bonin.jpg" alt="Foto von H. Bonin"
    height="50" width="50" /></a>
<a href="http://validator.w3.org/check/referer"><img
  src="/valid-xhtml10.gif" alt="Valid XHTML 1.0!"
    height="25" width="100" /></a>
<a href="http://jigsaw.w3.org/css-validator"><img
  src="vcss.gif" alt="Valid CSS!"
    height="25" width="100" /></a>
<a href="http://www.anybrowser.org/campaign/"><img
  src="starsbtn.gif" alt="Open to Any Browser"
    height="25" width="100" /></a>
<a href="http://www.gi-ev.de/"><img
  src="/myGI/praesidiumsmitglied.gif" alt="GI-Leitungsmitglied"
    height="25" width="100" /></a>
<a href="/content.html"><img
  src="/wanzendreh.gif" alt="[TKG -- Wanzenaktion]"
    height="50" width="50" /></a>
</h2>
</body>
<!-- Ende der Datei /u/bonin/mywww/index.php3 -->
</html>
```

9.2 Cascading Style Sheets (*CSS*)

Cascading Style Sheet ist ein zweigestufter Mechanismus (CSS1 ≡ Level 1; CSS2 ≡ Level 2), der jeweils sowohl dem Autor wie dem Leser ermöglicht Farben, Schriftart, Schriftgröße und Zwischenräume (≈ das gewünschte Layout) mit dem Dokument zu verknüpfen. CSS1 ist eine leicht lesbare und schreibbare Spezifikationssprache, die sich an der üblichen Desktop Publishing Terminology orientiert.

DTP

9.2.1 CSS-Konstrukte

Ein einfaches CSS-Konstrukt folgt folgender Form:

selektor { eigenschaft: wert }

Zum Beispiel wird mit dem folgendem Konstrukt die Hauptüberschrift als roter Text dargestellt:

```
h1 { color: red }
```

Ein Konstrukt besteht aus zwei Hauptteilen:

1. Selektor
   Im Beispiel: `h1`
2. Deklaration
   Im Beispiel: `color: red`
   Die Deklaration hat zwei Teile:
   (a) Eigenschaft (*property*)
   Im Beispiel: `color`
   (b) Wert (*value*)
   Im Beispiel: `red`

Der Selektor bildet die Verknüpfung zwischen dem HTML-Konstrukt und der Spezifikation des *Style Sheet*. Alle HTML-Elementtypen sind mögliche Selektoren. Die Eigenschaft `color` ist beispielsweise eine von ≈ 50 Eigenschaften, die die Präsentation festlegen. Der Autor eines Dokuments braucht nur seine speziellen Vorstellungen über die spätere Präsentation zu spezifizieren, weil der Browser (*User Agent*) ein *Default Style Sheet* besitzt. Der Autor überschreibt mit seiner Spezifikation dessen *Default*-Werte.

9.2.2 HTML-Dokument $\Leftrightarrow$ CSS

Das HTML-Dokument kann von dem *Style Sheet* auf verschiedene Weise verknüpft werden. Das folgende Beispiel zeigt vier Möglichkeiten:

1. im `<head>`-Konstrukt
   (a) mit dem `<link>`-Konstrukt wird ein Verweis auf eine externe CSS-Datei angeben und diese wird über den Web-Server geladen laden

<style>

(b) mit dem `<style>`-Konstrukt und der `@import`-Angabe wird ein Verweis auf eine externe CSS-Datei angegeben und über den Web-Server geladen

(c) direkt codiert im `<style>`-Konstrukt

2. im `<body>`-Bereich

   - mit dem `style`-Attribut eines HTML-Elementes

```
<html>
  <head>
    <link  href="myStyle.css"
      rel="stylesheet" type="text/css" />
    <title>Mein Dokument</title>
    <style type="text/css">
      @import url(http://as.fhnon.de/main.css);
      h1 { color: red }
    </style>
  </head>
  <body>
    <h1>Mein Dokument in Rot</h1>
    <p style="color: blue">Mein blauer Text</p>
  </body>
</html>
```

9.2.3 Gruppierung & Vererbung

Zur Verkürzung der CSS-Textlänge können Selektoren in Form einer Liste gruppiert werden. Zum Beispiel:

```
h1, h2, h3 { font-family: Times }
```

Oder auch in Kurzschreibweise notiert werden. Zum Beispiel:

```
h1 { font: bold 12pt/14pt Arial }
```

statt ausführlich:

```
h1 {
  font-weight:  bold;
  font-size:    12pt;
  line-height:  14pt;
  font-family:  Arial;
  font-variant: normal;
  font-style:   normal;
}
```

Bei geschachtelten Konstrukten erben die inneren Konstrukte Eigenschaften von den äußeren Konstrukten. Gilt zum Beispiel für den Selektor `<h1>`:

```
h1, h2 {
  font-size:   24pt;
  font-weight: bold;
  font-family: Arial, Helvetica;
  color:       yellow;
  background-color: blue;
  margin: 5px;
}
```

und den Selektor `<em>`:

```
em {
  font-style:italic
}
```

dann ist die folgende Überschrift ganz in Gelb auf blauem Hintergrund geschrieben.

```
<h1><em>CTP</em>-Dokumentation</h1>
```

Das `em`-Konstrukt erhält seine Farbe vom „Parent element“, hier: `<h1>`. Nicht jede Eigenschaft wird vererbt. So wird beispielsweise die Eigenschaft `background` nicht vererbet. Es empfiehlt sich daher bei einer Angabe von `color` stets auch eine Angabe für `background` zu machen.

```
body {
  background: url(http://as.fhnon.de/gelberBall.gif) black;
  color: white;
}
```

Im obigen Beispiel ist die Textfarbe weiß und der Hintergrund wird aus dem Bild „gelber Ball“ gebildet. Ist das Bild kleiner als die Hintergrundfläche wird das Bild wiederholt dargestellt. Die „Zwischenräume“ werden mit der zweiten Angabe von `background` aufgefüllt; hier also schwarz dargestellt. Die zweite Angabe wird auch benutzt, wenn das Bild nicht zugreifbar ist.

Bei der Spezifikation von einer Eigenschaft kann man sich auf andere Eigenschaften beziehen. Ein Beispiel ist die Prozentangabe bei der Eigenschaft `line-height`.

```
p {
  font-size:   14pt;
  line-height: 150%;
}
```

9.2.4 Selektor: `class` & `id`

class

Man kann Eigenschaften zu einer Klasse zusammenfassen. Die Klasse wird benannt und mit einem Punkt unmittelbar hinter dem Selektor notiert. Eine Klasse die für mehrere Selektoren genutzt werden soll wird ohne Selektorangabe mit einem Punkt beginnend spezifiziert. Es kann nur eine Klasse pro Selektor spezifiziert werden, wie das folgende Beispiel skizziert.

```
h1.myKopf {
  color:            yellow;
  background-color: black;
}
h1 {
  color:            red;
  background-color: black;
}
.myClass {
  color:            blue;
  background-color: maroon;
}
...
<h1 class="myKopf">Das Gelbe vom Ei</h1>
```

```
<h1>Der rote Kopf</h1>
<h1 class="myClass">Das Blaue vom Himmel</h1>
<p class="myClass">Blau, blau ... </p>
...
Geht nicht!
<h1 class="myKopf" class="myClass">Fehler</h1>
```

Mit dem `id`-Attribut wird üblicherweise einem einzelnen Element eine CSS-Spezifikation zugeordnet. Der `id`-Wert muß im Dokument eindeutig sein. Er wird beginnend mit einem Hashzeichen „#“ notiert, wie das folgende Beispiel zeigt. id

```
#ZZ981 { letter-spacing: 0.3em }
#ZZ982 { letter-spacing: 0.5em }
...
<p id="ZZ982">Buchstaben mit viel Zwischenraum</p>
```

9.2.5 Kontextabhängige Selektoren

Im folgenden CSS-Beispiel sind alle `<em>`-Konstrukte im Dokument von der Spezifikation (grüne Textfarbe) betroffen:

```
h1 {
  color: red;
}
em {
  color: green;
}
```

Soll sich die Spezifikation nur auf `<em>`-Konstrukte innerhalb eines `<h1>`-Konstruktes beziehen, dann kann ein kontextabhängiger Selektor wie folgt notiert werden

```
h1 {
  color: red;
}
h1 em {
  color: green;
}
```

Auch solche kontextabhängigen Selektoren sind grupperbar. Zum Beispiel entspricht

```
h1 em, h2 b {
  color: blue;
}
```

der Spezifikation

```
h1 em {
  color: blue;
}
h2 b {
  color: blue;
}
```

9.2.6 Kommentare im CSS

/*...*/ Ein Kommentar wird innerhalb eines CSS mit der Slash-Sternchen-Kombination gekennzeichnet, ähnlich wie in Java oder C.

```
/* Bild wird häufig nicht angezeigt */
ul {
  list-style-image:
    url(http://as.fhnon.de/gelberBall.gif) white;
  list-style-position: inside;
}
```

9.2.7 Pseudo-Konstrukte (`a:link`, `p:first-letter`, usw.)

Üblicherweise zeigt ein Web-Browser neue Links (Anker: `a`-Konstrukte) in anderem Layout an als die schon „besuchten“. Ihr Layout läßt sich mit Hilfe der sogenannten *Anchor Pseudo-Classes* spezifizieren. Pseudo-Konstrukte werden ähnlich wie Klassen angeben, allerdings mit Doppelpunkt und nicht mit Punkt getrennt.

```
a:link {         /* unbesuchte Link */
  color: red;
}
a:visited  {     /* aufgesuchter Link */
```

```
  color: blue;
}
a:active {       /* aktiver Link */
  color: green;
}
```

Die Pseudo-Konstrukte `first-line` und `first-letter` werden benutzt um einen Absatz zu gestalten, zum Beispiel mit einem großen Buchstaben am Anfang.

```
p:first-letter {
  font-size: 24pt;
  float: left;
  color: yellow;
  background-color: black;
}
```

Dabei können Pseudo-Konstrukte mit Klassen in den Selektoren verknüpft werden, wie das folgende Beispiel zeigt:

```
p.anfang:first-letter {
  color: yellow;
  background-color: black;
}
...
<p class="anfang">Erster Ansatz im Text</p>
...
```

9.2.8 Die Kascade & Konflikte

Ein HTML-Dokument kann von mehr als einer CSS-Spezifikation beeinflußt werden. Verantwortlich sind dafür zwei Aspekte:

- Modularität: mehr als eine CSS-Angabe in einem HTML-Dokument Zum Beispiel:

```
@import url(http://as.fhnon.de/mainStlye.css);
@import url(http://as.fhnon.de/myStlye.css);
h1 {
  color: blue /* ueberschreibt importierte Sheets */
}
```

- Autor⇔Leser-Balance
  Der Leser kann mit seinem Style Sheet die Autoren-Vorgaben beeinflussen (↪ `important`-Deklaration).

Die CSS-Angaben für ein HTML-Dokument können Konflikte aufweisen. Diese werden mit Hilfe von Gewichtungsfaktoren gelöst. So ist normalerweise das Gewicht der Leser-Spezifikation geringer als das Gewicht der Autoren-Spezifikation. Es sei denn, in der Leser-Spezifikation wird eine Eigenschaft-Wert-Angabe mit `! important` gekennzeichnet.

```
h1 {
  color: black ! important;
}
p {
  font-size: 12pt ! important;
  font-style: italic
}
```

Die Farbangabe eines Lesers für den obigen `h1`-Selektor überschreibt eine Farbangabe des Autors, weil diese mit `! important` markiert ist.

Um CSS-Konflikte zu lösen, werden CSS-Eigenschaft-Wert-Angaben nach folgender Vorgehensweise abgearbeitet (↪ [LieBos96] Chapter 3):

1. Für einen Selektor werden alle Eigenschaft-Wert-Angaben festgestellt.

2. Gibt es keine entsprechende Angabe wird die geerbte Angabe eingesetzt. Gibt es keine geerbte, dann wird der Initialwert verwendet.

3. Die Angaben werden nach Gewicht sortiert. Als wichtig gekennzeichnet Angaben (`! important`) haben dabei ein höheres Gewicht.

4. Die Angaben werden nach der Quelle sortiert. Dabei gilt: Autoren-Angeben haben mehr Gewicht als Leser-Angaben. Diese haben mehr Gewicht als Einstellungen des Web-Browsers (*User Agent*).

5. Die Angaben werden nach dem „Grad der Spezifizierung“ sortiert. Spezielle Angaben überschreiben generelle Angaben. Dieser Grad wird wie folgt ermittelt:

   α `id`-Feststellung

   β `class`-Feststellung

   γ Feststellung der Anzahl der HTML-Tags in der Deklaration

   Das folgende Beispiel skizziert die Gewichtsermittlung:

   ```
   li {...}              /* α = 0 β = 0 γ = 1 ⟶ Gewicht= 1 /*
   ul li {...}           /* α = 0 β = 0 γ = 2 ⟶ Gewicht= 2 /*
   ul ol li {...}        /* α = 0 β = 0 γ = 3 ⟶ Gewicht= 3 /*
   li.red {...}          /* α = 0 β = 1 γ = 1 ⟶ Gewicht= 11 /*
   ul ol li.red {...}    /* α = 0 β = 1 γ = 3 ⟶ Gewicht= 13 /*
   #x123 {...}           /* α = 1 β = 0 γ = 0 ⟶ Gewicht= 100 /*
   ```

   Dabei zählen die *Pseudo*-Selektoren wie zum Beispiel `a:link` als normale Elemente.

6. Bei Angaben mit gleichem Gewicht wird die zuletzt spezifizierte gewählt.

9.2.9 CSS-Beispiel

Das HTML-Dokument `exampleCSS.html` enthält in seinem `<link>`-Konstrukt einen Verweis auf die CSS-Datei `myStyle.css`. Die Abbildung 9.1 S. 386 zeigt die Darstellung eines Auszugs des Dokumentes mit dem Browser *Microsoft Internet Explorer*, Version 6.0.2600, auf einer Windows-XP-Plattform.

CSS-Datei `myStyle.css`

```
/* Basis-Layout fuer das Projekt: FOO  */
/*   Bonin 23-06-1998                 */
/*   Update                           */
/* Achtung: Kaum ein Browser zeigt     */
/*          dieses Layout korrekt!     */
/* Bild wird häufig nicht angezeigt    */
ul {
 color: white;
```

Legende:
Dargestellt mit *Microsoft Internet Explorer*, Version 6.0.2600, auf einer Windows-XP-Plattform.
Quellcode `exampleCss.html` ↪ S. 389, Quellcode `myStyle.css` ↪ S. 385

Abbildung 9.1: XHTML: `exampleCSS.html` & CSS: `myStyle.css`

```
  background-color: black;
  list-style-image:
    url(http://as.fhnon.de/gelberBall.gif);
  list-style-position: inside;
}
ol {
  color: white;
  background-color: black;
  list-style-type: lower-roman;
}

p:first-letter {
  font-size: 24pt;
  float: left;
  color: yellow;
  background-color: black;
}
p {
  font-family:    "Times New Roman", Times, serif;
  font-weight:    bold;
  font-size:      14pt;
  line-height:    150%;
  letter-spacing: 0.5em;
  color: green;
  background-color: yellow;
}
h3 {
  font-family: Arial, Helvetica, sans-serif;
  font-size:   20pt;
  font-weight: bold;
  text-decoration: underline;
  color: green;
  background-color: white;
  margin: 3em;
}
h1, h2 {
  font-size:   24pt;
  font-weight: bold;
  font-family: Arial, Helvetica;
  color:       yellow;
  background-color: blue ! important;
  margin: 5px;
```

```
}
body {
  background:
    url(http://as.fhnon.de/gelberBall.gif) black;
  color:            white;
}
a:link {
  font-weight:     bold;
  text-decoration: none;
  color:           red;
  background-color: black;
}
a:visited {
  font-weight:     bold;
  text-decoration: none;
  color:           blue;
  background-color: black;
}
a:active {
  font-weight:     bold;
  text-decoration: none;
  color:           yellow;
  background-color: black;
}
EM {
  font-style:italic
}
.hinweis {
  font-style:  italic;
  font-weight: bold;
  color:       yellow;
  background-color: black;
}
.hervorhebung {
  color:            white;
  background-color: maroon;
  margin:           10px;
  border:           none;
}
.dickeListe {
  color:           white;
  background-color: black;
```

```
  list-style-type: square;
  font-family:     "Times New Roman", Times, serif;
  font-size:       18pt;
  font-weight:     bold;
}
.anhang {
  color:       white;
  background-color: black;
  font-style:  italic;
  font-family: Helvetica, sans-serif;
  font-size:   12pt;
  text-indent: 25px;
}
```

HTML-Datei `exampleCSS.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- CSS-Beispiel                                    -->
<!-- Testbett fuer Applets                          -->
<!-- Bonin  25-Jun-1998                             -->
<!-- Update ... 23-Dec-2002                         -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
<link  href="http://as.fhnon.de/anwd/xhtml/myStyle.css"
  rel="stylesheet" type="text/css" />
<title>CTP-Dokumentation</title>
</head>
<body>
<h1><em>CTP</em>-Dokumentation</h1>
<p class="hinweis">
Graphische Darstellung der
<a href="Fachklassen.html">Fachklassen</a>
</p>
<h3>Vorgehensweise</h3>
<p>Es sind folgende Schritte zu vollziehen:</p>
<ul>
 <li>Bestimme die Jahrestrainingsstunden</li>
 <li>Bestimme Hauptwettkampkalenderwoche und Jahr</li>
 <li>Prüfe den Hauptwettkampftermin</li>
 <li>Berechne die Handlungsspanne</li>
 <li>Berechne die Wartezeit</li>
 <li>Berechne frühesten Beginntermin des Trainingsplanes</li>
 <li>Berechne spätesten Beginntermin des Trainingsplanes</li>
 <li>Lege den Trainingsbeginn fest</li>
```

```
</ul>
<p>Danach kann ein kompletter Trainingsplan
erstellt werden, dessen Hauptwettkampf erreichbar ist.
<div class="hervorhebung">
Dazu dienen folgende Methoden:
<ul class="dickeListe">
  <li>getFruehestesTrainingsPlanBeginnJahr()</li>
  <li>getFruehesteTrainingsPlanBeginnWoche()</li>
  <li>getSpaetestesTrainingsPlanBeginnJahr()</li>
  <li>getSpaetesteTrainingsPlanBeginnWoche()</li>
</ul>
<p>Diese Methoden sind mehr als die üblichen
"get-Methoden". Sie berechnen abhängig von der
Handlungsspanne und der aktuellen Woche,
den frühesten und spätesten Trainingsplanbeginn.
Dabei wird der Jahresumbruch berücksichtigt, das hei&szlig;t,
der Hauptwettkampf liegt im nächsten Kalenderjahr.</p>
</div>
<p class="rechtsKlein">
<a href="/copyright.html">Copyright</a>
  <a href="mailto:bonin@fhnon.de">Bonin</a>
  Apr-1995,..., Dec-2002 all rights reserved
<a href="http://validator.w3.org/check/referer">
  <img src="/valid-xhtml10.gif"
    alt="Valid XHTML 1.0!" height="15" width="44" /></a>
</p>
</body>
<!-- Ende der Datei /u/bonin/mywww/html/exampleCSS.html -->
</html>
```

Kapitel 10

Java^{TM} — OO-Anspruch und OO-Wirklichkeit

OO war ursprünglich ein Denkmodell der Programmierung im engeren Sinne primär in der Form der ersten OO-Sprachen wie `Simula 67` (↪ [Dahl+67]) oder `CLOS` (↪ [Gabriel91]). Heute ist Objektorientierung (OO) ein Paradigma der gesamten Softwareentwicklung, das das ganze Spektrum von der Spezifikation, der Konstruktion bis zur Implementierung inclusive Betrieb und Wartung unterstützt. Java^{TM} ist unstrittig im Bereich Internet- und Client/Server-Systeme die zweckmäßige OO-Sprache. Allerdings erfüllt Java^{TM} nur bedingt das OO-Paradigma strikt und vermeidet natürlich auch nicht Mängel dieses Paradigmas. Plakativ formuliert: Java^{TM} ist bewährte „OO-Hausmannskost“ und nicht auf dem heutigen Stand der wissenschaftlich verstandenen Programmiermethodik und Softwaretechnik.

Trainingsplan

Das Kapitel „Java^{TM} — OO-Anspruch und OO-Wirklichkeit“ erläutert:

- das OO-Paradigma, also OO-Grundlagen, OO-Prinzipien, OO-Konzepte im Hinblick auf ihre konkrete Umsetzung und

↪ Seite 392 . . .

- die strikte Objekt-Orientierung im Java^{TM}-Kontext.
  ↪ Seite 393 . . .

10.1 OO-Paradigma — unvollständige Umsetzung

Üblicherweise werden als wesentliche Elemente der Objekt-Orientierung folgende genannt (z. B. ↪ [Broy/Siedersleben02] S. 4):

- Klassen mit Attributen und Methoden als Granulat zur Beschreibung und Strukturierung von Programmen,
- Schnittstellen als Listen von Methoden,
- Erzeugung von Objekten als Instanzen von Klassen,
- Speicherung von Daten und Zuständen in Attributen von Klassen und Objekten,
- Objektidentität definiert durch die Speicheradresse,
- Vererbung und Polymorphie.

Aufbauend auf diesen Elementen beansprucht die Objekt-Orientierung für sich, die folgenden Prinzipien umzusetzen (↪ [Broy/Siedersleben02] S. 4):

1. Datenabstraktion,
2. Geheimnisprinzip,
3. wohldefinierte Schnittstellen,
4. Modularität durch Kapselung der Objektdaten,
5. Dynamik und Flexibilität durch die Instanziierung von Objekten,
6. Wiederverwendung von Code durch Vererbung und Aggregation.

In diesem Kontext behält die Objekt-Orientierung jedoch Mängel bei, zum Beispiel (↪ [Broy/Siedersleben02] S. 4):

- OO liefert keinen geeingenten Komponentenbegriff als Architekturbasis.
- OO kennt keine Komposition von Klassen.
- OO realisiert ein sequenzielles Ausführungsmodell (— wie bei prozeduralen Sprachen üblich).
- OO sagt uns nicht, wie wir das Verhalten (Funktionen und Interaktionen) von Schnittstellen definieren sollen.

Ein wesentliche OO-Kritik basiert auf der undefinierten Fernwirkung einer Methodenapplikation (↪ [Broy/Siedersleben02] S. 6). Dazu folgendes Beispiel: x, y seien Instanzen der Klassen X bzw. Y; $f()$ sei eine Methode von X und $g()$ eine Methode von Y. Die Notation

$$x.f() \approx> y.g()$$

bedeutet: $x.f()$ kann den Aufruf $y.g()$ verursachen, und zwar direkt (der Aufruf $y.g()$ steht im Code von $f()$) oder indirekt (eine Folge von Unteraufrufen führt vom Aufruf $x.f()$ zum Aufruf $y.g()$). Die in Java^{TM} vorhandene Importanweisung (z. B. in X: `import Y;`) sagt nur, welche anderen Klassen zur Kompilierung von X benötigt werden, aber nur wenig über den Wirkungsbereich $W(x, f())$ des Aufrufs $x.f()$:

$$W(x.f()) = \{(y.g())|x.f() \approx> y.g()\}$$

Der Wirkungsbereich W ist bestenfalls im Kommentar beschrieben. Diese Java^{TM}-OO unterstützt uns nicht bei der notwendigen W-Ermittlung.

10.2 Strikte Objekt-Orientierung

"Although it is based on C++,
Java is more of a 'pure' object-oriented language."
(↪ [Eckel02] p. 77)

In Java^{TM} entspricht die Basis nicht dem Konzept einer strikten Objekt--Orientierung weil die sogenannten primitiven Typen keine Instanzen einer Klasse sind (*PrimitiveType* ↪ Tabelle 6.5 S. 150). Beispielsweise sind die Symbole 1, 2, 3, ... Werte und keine Namen für Integer-Objekte. Ebenso sind a, b, c, ... Werte und keine Identifier für Elemente der Zeichenmenge. Auch `true` und `false` sind Werte und keine Identifier für die beiden Boolean-Objekte. Bei der strikten Objekt-Orientierung sind 1, 2, 3, ... Namen für die Zahl 1, die Zahl 2, die Zahl 3, und so weiter.

Die strikte OO-Welt besteht nur aus Objekten. Ein Objekt ist entweder zusammengesetzt aus anderen (*composite object*) oder einfach (*simple object*), also ohne eine Referenz auf eine andere Eigenschaft. Die Tabelle 10.1 S. 395 verdeutlicht diesen Java^{TM}-OO-Mangel.

Strikte Objekt-Orientierung		
Ausdruck	**OO-Code**	**OO-Feature**
-1	`1.negate()`	*Method invocation* Das Objekt `1` appliziert die Integer-Negationsmethode und eine Referenz zum neuen Integer-Objekt `-1` wird zurückgegeben.
$1 + 2$	`1.add(2)`	*Method invocation*; *Collaboration*
Integers: a, b $a + b$	`a.add(b)`	*Method invocation*; *Collaboration*
Integers: a, b, c $a + b * c$	`a.add(b.mult(c))`	*Method invocation*; *Collaboration*; *Cascaded method call*
Characters: a, b $a < b$	`a.lt(b)`	*Method invocation*; *Collaboration*; *Composition* (Eine Referenz zum Boolean-Objekt `true` oder `false` wird zurückgegeben.
Booleans: a, b $a \wedge b$	`a.and(b)`	*Method invocation*; *Collaboration*

Legende:

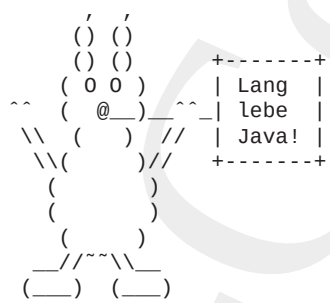
Tabellenidee $\hookrightarrow$ [Ourosoff02] p. 106.

In Java^{TM} sind 1, 2, 3, ... Werte und keine Bezeichner für Objekte, daher ist der OO-Code (Spalte 2) nur bedingt abbildbar. Statt 1 wäre zum Beispiel `eins` zu notieren, wobei dann vorab `int eins = 1;` zu erklären wäre.

Tabelle 10.1: Einfache Ausdrücke $\Rightarrow$ OO-Code

Kapitel 11

Java^{TM} N Plattform: Hoffnungen & Visionen



Jeder Text von derartiger Länge und „Tiefe“ verlangt ein abschließendes Wort für seinen getreuen Leser. Es wäre nicht fair, nach so vielen Seiten, die nächste aufzuschlagen und dann den Anhang zu finden. Daher zum Schluß ein kleiner Ausblick. Sicherlich wird die Objekt-Orientierung zu einer selbstverständlichen Basistechnik reifen. Die Innovationen dazu werden auf höheren Abstraktionsebenen erwartet. Wie aber wird sich in diesem Kontext Java^{TM} weiterentwickeln? Was sind unsere Hoffnungen und Visionen? Was kennzeichnet die Plattform $N > 2$? Gibt es im Jahr 2010 noch ein zeitgemäßes Java^{TM}?

Wenn die bisherigen Generationen schlagwortartig bezeichnet werden können als:

1. objekt-orientierte Sprache (Desktop-Generation),
2. Web-Tool (Server-Generation) und

3. Enterprise Information Plattform (Komponenten-Generation)

wie lautet dann das Schlagwort der nächsten Generation? Vielleicht . . .? Halt, bevor der JAVA^{TM}–COACH zum Science-Fiction-Roman mutiert, zurück zur Programmierung bzw. zur Softwarekonstruktion. Dafür lassen sich holzschnittartig folgende Vermutungen formulieren:

- **Objekt-Orientierung**++
  Der Visionär sieht beispielsweise ein Vererbungskonzept, das viel stärker der Biologie entspricht, also Objekten ermöglicht durch „Kopulieren“ ein Objekt („Kind“) zu erzeugen, das die Eigenschaften seiner Erzeuger aufweist.

- **Netzorientierung**++
  Der Visionär sieht die Unterstützung von autonomen, sich „selbständig“ im Netz bewegenden Programmen, die sich durch das biologische Vererbungskonzept weiterentwickeln. Der Virus von heute mutiert zum Arbeitspferd von morgen.

Programmieren bleibt schwierig!

Die ganz am Anfang stehende Aussage *Programmieren bleibt schwierig!* wird bestimmt auch in Zukunft gültig bleiben. Wenn sich Java^{TM} kontinuierlich weiterentwickelt, dann gibt es sicherlich andere Kernfragen bei den Schwierigkeiten. Es besteht aber die berechtigte Hoffnung, das Ihre Kenntnisse, die Sie beim Durcharbeiten gewonnen haben, für Sie nützlich bleiben. Beim Nutzen dieser Kenntisse wünsche ich Ihnen abschließend viel Erfolg.

Anhang A

Übungen

A.1 Modellierung einer Stückliste

Das Unternehmen *RadVertriebsExperten GmbH* (RVE) verkauft im Geschäftsjahr ≈ 5000 Fahrräder, die bei ≈ 7 Herstellern eingekauft werden. RVE beauftragt das Softwarehaus *InterSystems AG* (IAG) ein objektorientiertes Warenwirtschaftssystem grob zu planen. Als erstes Diskussionspapier soll die IAG zunächst nur ein Klassendiagramm für die Produkte aus der **Montagesicht** aufstellen.

Im Rahmen dieses Auftrages stellt die IAG bei ihren Recherchen folgende Punkte fest:

Requirements

1. Alle Produkte sind Fahrräder.
2. Ein Fahrrad ist entweder ein Einrad oder ein Zweirad. Diese beiden Radtypen unterscheiden sich durch die Anzahl ihrer Laufräder.
3. Ein Laufrad wird durch seinen Durchmesser beschrieben.
4. Jedes Fahrrad hat eine Rahmennummer.
5. Schutzbleche und Gepäckträger sind Anbauteile.
6. Jedes Anbauteil hat eine Teilenummer.
7. Ein Schutzblech wird durch die Materialart und die Breite beschrieben.

8. Ein Gepäckträger wird durch die Tragkraft beschrieben.
9. Nur an ein Zweirad können Anbauteile montiert werden.
10. Es werden stets zwei Schutzbleche montiert.
11. Es werden maximal zwei Gepäckträger montiert.
12. Zu jedem Anbauteil gibt es eine Montageanleitung.
13. Die Montageanleitung nennt die durchschnittliche Montagedauer und hat einen Text, der die Vorgehensschritte beschreibt.
14. Jedem Fahrrad ist anzusehen, ob es probegefahren wurde.

A.1.1 Klassendiagramm für die Montagesicht

Entwerfen Sie ein Klassendiagramm für die RVE. Notieren Sie Ihr Klassendiagramm in UML. Es sollte möglichst viele der obigen Punkte abbilden.

A.1.2 Diagrammerweiterung um den Montageplatz

In der ersten Diskussionsrunde mit der RVE möchte Herr Abteilungsleiter Dr. Moritz Krause unbedingt den Montageplatz noch aufgenommen haben. Ein Montageplatz ist ausgestattet nach der Vorgabe G ($\equiv$ Grundausstattung) oder S ($\equiv$ Sonderausstattung). Skizzieren Sie die notwendige Ergänzung in Ihrem Klassendiagramm.

A.2 Klassendiagramm für mehr Transparenz

Das Unternehmen *SportwaffenVertriebInternational GmbH* (SVI) setzt pro Geschäftsjahr $\approx$ 10000 Jagd- und Sportwaffen um. Es werden 11 Zweigstellen beliefert. Die umsatzstärkste Zweigstelle ist in Mannheim. Sie verkauft $\approx$ 1200 Waffen, die umsatzschwächste ist in Lüneburg und verkauft $\approx$ 240. Der SVI-Geschäftsführer beauftragt das Softwarehaus *Multimedia InformationsSysteme Tübingen* (MIST-AG) ein modernes Warenwirtschaftssystem grob zu planen. Der Projektleiter Herr Emil Jonnis arbeitet sich in die Materie ein und stellt dabei zunächst folgende Fakten fest:

1. Alle SVI-Produkte sind Sport- oder Jagdwaffen (kurz: Waffen).
2. Es werden Langwaffen von Kurzwaffen unterschieden. Langwaffen sind mindestens 60 cm lang.
3. Eine Waffe ist entweder ein Gewehr oder ein Revolver oder eine Pistole.
4. Gewehre sind Langwaffen. Pistolen und Revolver sind Kurzwaffen.
5. Ein Gewehr ist entweder eine Flinte oder eine Büchse oder eine Kombination davon, also eine kombinierte Waffe.
6. Flinten haben einen glatten Lauf.
7. Büchsen haben einen gezogenen Lauf.
8. Ein Lauf wird durch das Kaliber beschrieben. Die Kaliberangabe ist entstehungsgeschichtlich bedingt. Sie läßt sich als eine Zeichenkette, zum Beispiel für einen Flintenlauf „`12/70`“ oder einen Büchsenlauf „`.308Win`“ beschrieben.
9. Jede Waffe hat eine Herstellernummer. Diese wird vom Hersteller vergeben. Sie ist nur mit dem Herstellernamen eindeutig.
10. Alle Teile, die dem Gasdruck ausgesetzt sind tragen ein Beschußzeichen. Es werden aber nur die Beschußzeichen auf den Läufen im Warenwirtschaftssystem registiert.
11. Es werden derzeit folgende Gewehrtypen verkauft:
    (a) Querflinte ≡ zwei nebeneinanderliegende Flintenläufe
    (b) Bockflinte ≡ zwei übereinanderliegende Flintenläufe
    (c) Bockbüchse ≡ zwei übereinanderliegende Büchsenläufe
    (d) Bockbüchsflinte ≡ ein Flintenlauf liegt über einem Büchsenlauf
    (e) Drilling ≡ eine Querflinte mit zusätzlichem Büchsenlauf
12. Hat das Gewehr mindestens einen Büchsenlauf, dann kann es auch ein Zielfernrohr haben.

13. Ein Zielfernrohr wird durch seine Brennweite und Lichstärke beschrieben.
14. Jedes Zielfernrohr hat zum Zielen ein sogenanntes „Absehen“.
15. Bei den Absehen gibt es unterschiedliche Typen, zum Beispiel Absehen1, Absehen4, Absehen4A oder Absehen8.

A.2.1 Klassendiagramm notieren

Herr Emil Jonnis scheint bei dieser Faktenmenge den Überblick zu verlieren. Sie möchtem ihm helfen und entwerfen deshalb ein vorläufiges Klassendiagramm in UML-Notation. Ihr Diagramm sollte möglichst viele der obigen Fakten abbilden. [Hinweis: Da es sich um die fachlichen Klassen handeln sollte, sind „Getter“ und „Setter“ nicht aufzunehmen.]

A.2.2 Diagrammergänzung um zusätzlichen Aspekt

Herr Emil Jonnis möchte im Rahmen seiner Analyse über Fragen zur Waffenbesitzkarte (WBK) mit Fachleuten diskutieren. Bisher kennt er nur folgende Fakten:

1. Jeder Käufer einer Kurzwaffe muß in seiner gültigen Waffenbesitzkarte den Waffentyp und das Kaliber vorab eingetragen haben.
2. Eine Waffenbsitzkarte wird von der zuständigen Ordnungsbehörde ausgestellt.
3. Jede Waffenbesitzkarte hat bezogen auf die Ausstellungsbehörde eine eindeutige Nummer.
4. Beim Verkauf einer Kurzwaffe wird daher sofort die jeweilige WBK registiert.

Ergänzen Sie Ihr bisheriges Klassendiagramm um diese Fakten.

A.3 Shell-Kommando „`echo`“ programmieren

A.3.1 Abbildung als Applikation

Schreiben Sie eine Applikation, die das übliche `echo`-Kommando einer UNIX- und/oder MS-DOS-Shell abbildet. (Idee entnommen [Flanagan96])

A.3.2 Unterschiede zum Shell-Kommando

Nennen Sie Unterschiede Ihrer Lösung zum `echo`-Kommando einer üblichen Shell.

A.4 Applikation `Wert`

Der „Gleichheitsoperator“ `==` testet, ob seine beiden Operanden auf dasselbe Objekt verweisen. Obwohl zwei Objekte die gleiche Zeichenkette darstellen, kann das Testergebnis daher `false` sein. Für Literalkonstanten werden Objekte der Klasse `String` angelegt, wobei die Literalkonstante (Zeichenkette) dann die Referenz auf dieses Objekt repräsentiert. Längere Literalkonstanten können zerlegt und mit `+` wieder zusammengesetzt werden.

Klasse `Wert`

```
/**
 *  Beispiel: Gleichheit in Java
 *
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.gleichheit;

public class Wert
{

   public static void main(String[] args)
   {
      String wert = "Software";

      String part1 = "Soft";
      String part2 = "ware";

      String s1 = new String(wert);
      String s2 = new String(wert);

      System.out.println("Fall 1: " +
            ("Software" == s1));
      System.out.println("Fall 2: " +
            (s1 == s2));
```

```
      System.out.println("Fall 3: " +
            ("Software" == "Soft" + "ware"));
      System.out.println("Fall 4: " +
            (wert == "Soft" + "ware"));

      System.out.println("Fall 5: " +
            (wert == "Soft" + part2));
      System.out.println("Fall 6: " +
            (wert == part1 + part2));

      System.out.println("Fall 7: " +
            (wert.equals(part1 + part2)));
   }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnis an:

```
>java de.fhnon.gleichheit.Wert
```

A.5 Applikation `Scoping`

Die Klasse `Scoping` skizziert eine Möglichkeit zur Begrenzung der Reichweite mittels zusätzlicher Blockstrukturierung (↪ Abschnitt8.3.2 S. 362).

Klasse `Scoping`

```
/**
 *  Beispiel: Scoping in Java
 *
 *@author     Hinrich Bonin
 *@version    1.0  18-Mar-2004
 */

package de.fhnon.scope;

import java.util.Date;

public class Scoping
{

   public static void main(String[] args)
```

```
   {
      Date i = new Date();
      {
         Date j = new Date();
         System.out.println("j: " + j);
      }
      System.out.println("i: " + i);

      System.out.println("j: " + j);
   }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnis an:

```
>javac de/fhnon/scope/Scoping.java
```

A.6 Applikation Kontrolle

Klasse Kontrolle

```
/**
 *  Kleine Kostprobe fuer: data types and control structures
 *
 *@author     Hinrich Bonin
 *@created    7. Januar 2003
 *@version    1.1
 *@since      02-Apr-1998
 */
public class Kontrolle
{
   public static void main(String[] args)
   {
      int i = args.length;
      int j;
      int k;
      double m = 0.3141592653589791;
      int n = (int) m;

      String p = "java";

      boolean wichtig;
      boolean vielleicht = true;
      boolean klar;
```

```
      i += p.length();
      j = i++;
      i--;
      k = ++i;
      --i;

      wichtig = (i == j && vielleicht);
      wichtig = (wichtig != vielleicht);
      klar = (i <= k) || (wichtig == true);

      System.out.println(
            "Werte: " +
            "\ni = " + i +
            "\nj = " + j +
            "\nk = " + k +
            "\nm = " + m +
            "\nn = " + n +
            "\np = " + p +
            "\nvielleicht = " + !vielleicht +
            "\nwichtig = " + !wichtig +
            "\nklar = " + klar);
   }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnis an:

```
>java Kontrolle ist besser!
```

A.7 Applikation `Iteration`

Klasse `Iteration`

```
/**
 *  Kleine Kostprobe fuer: Iterationen
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.0
 *@since      02-Apr-1998
 */
public class Iteration
{
```

```
public static void main(String[] args)
{
   boolean in = true;
   int zaehler = 0;
   int index;
   String spruchTabelle[]
          = new String[args.length];

   String meinSpruch = "";
   String wortZumSuchen = "C++";
   String wortZumErsetzen = "Java";

   spruchTabelle[0] = "Maximum";
   spruchTabelle[1] = "UML";
   spruchTabelle[2] = "&";
   spruchTabelle[3] = "C++";
   spruchTabelle[4] = "in der";
   spruchTabelle[5] = "Anwendungsentwicklung";

   int anzahlPositionen = spruchTabelle.length;

   while (zaehler < anzahlPositionen)
   {
      if (spruchTabelle[zaehler].equals(
            wortZumSuchen))
      {
         spruchTabelle[zaehler] =
               wortZumErsetzen;
         break;
      }
      zaehler++;
   }

   zaehler = -1;
   do
   {
      zaehler++;
      meinSpruch += spruchTabelle[zaehler];
      meinSpruch += " ";
   } while (zaehler < (anzahlPositionen - 1));
   System.out.println(meinSpruch +
         "\nDies sind " + meinSpruch.length() +
         " Zeichen!");
}
}
```

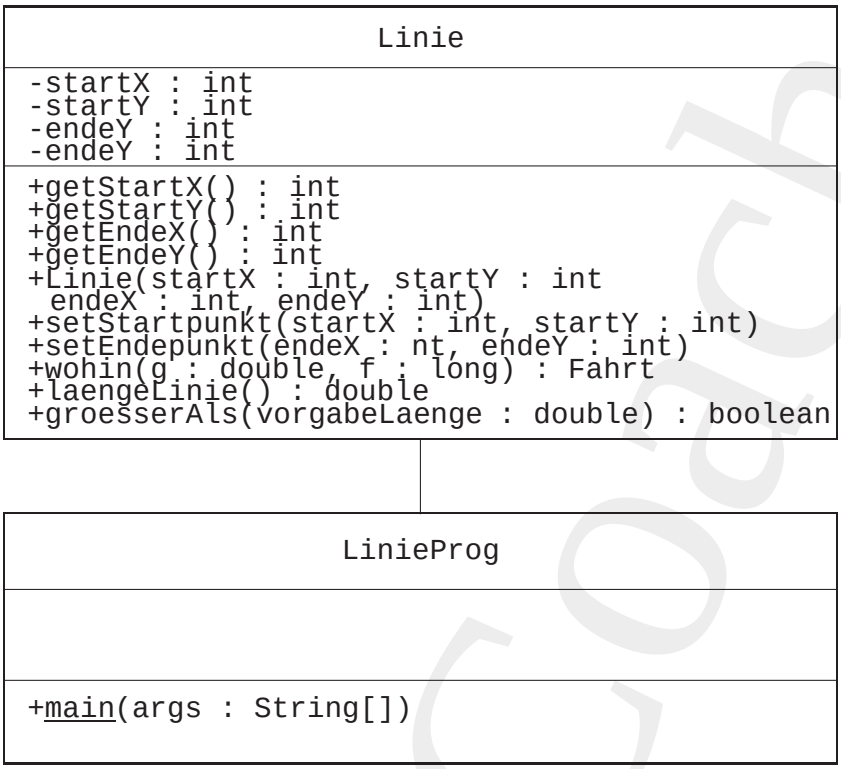


Abbildung A.1: Klassendiagramm für `LinieProg`

Geben Sie bei den folgenden Aufrufen das Ergebnisse an:

```
>java Iteration 1 2 3 4 5 6 7
>java Iteration 1 2
```

A.8 Applikation `LinieProg`

Die Abbildung A.1 S. 408 zeigt das Klassendiagramm der Applikation `LinieProg`.

Klasse Linie

```
/**
 *  Kleine Kostprobe fuer: this und super-Konstruktor
 *
 *@author     Hinrich Bonin
 *@version    1.1 23-Mar-2004
 *@since      06-Apr-1998, 15-Jul-1998, 26-Nov-2002
 */
public class Linie
{
   private int startX, startY, endeX, endeY;


   public int getStartX()
   {
      return startX;
   }


   public int getStartY()
   {
      return startY;
   }


   public int getEndeX()
   {
      return endeX;
   }


   public int getEndeY()
   {
      return endeY;
   }


   public Linie(int startX, int startY,
         int endeX, int endeY)
   {
      super();
      this.startX = startX;
      this.startY = startY;
      this.endeX = endeX;
      this.endeY = endeY;
   }
```

```
   public void setStartpunkt(int startX, int startY)
   {
      this.startX = startX;
      this.startY = startY;
   }


   public void setEndepunkt(int endeX, int endeY)
   {
      this.endeX = endeX;
      this.endeY = endeY;
   }


   public double laengeLinie()
   {
      return (Math.sqrt(
            Math.pow((double) endeX - startX, 2.0) +
            Math.pow((double) endeY - startY, 2.0)));
   }


   public boolean groesserAls(double vorgabeLaenge)
   {
      return (this.laengeLinie() > vorgabeLaenge);
   }
}
```

Klasse `LinieProg`

```
/**
 *  Kleine Kostprobe fuer: this und super-Konstruktor
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.0
 *@since      06-Apr-1998, 15-Jul-1998
 */

public class LinieProg
{

   public static void main(String[] args)
```

```
   {
      Linie l1 = new Linie(10, 10, 13, 14);

      Linie l2 = l1;
      l2.setStartpunkt(0, 0);
      l2.setEndepunkt(3, 4);

      double vorgabeLaenge = 0.25000e2;

      System.out.println("" +
            l1.getStartX() + l1.getStartY() +
            l1.getEndeX() + l1.getEndeY() + "\n" +
            l1.laengeLinie() + "\n" +
            l1.groesserAls(vorgabeLaenge)
            );
   }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java LinieProg
```

A.9 Applikation `Inheritance`

Die Abbildung A.2 S. 412 zeigt das Klassendiagramm der Applikation `Inheritance`.

Klasse `Inheritance`

```
/**
 *  Kleine Kostprobe fuer: Vererbung --- abstract class
 *
 *@since      05-Apr-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.inheritance;

public class Inheritance extends Bar
{
```

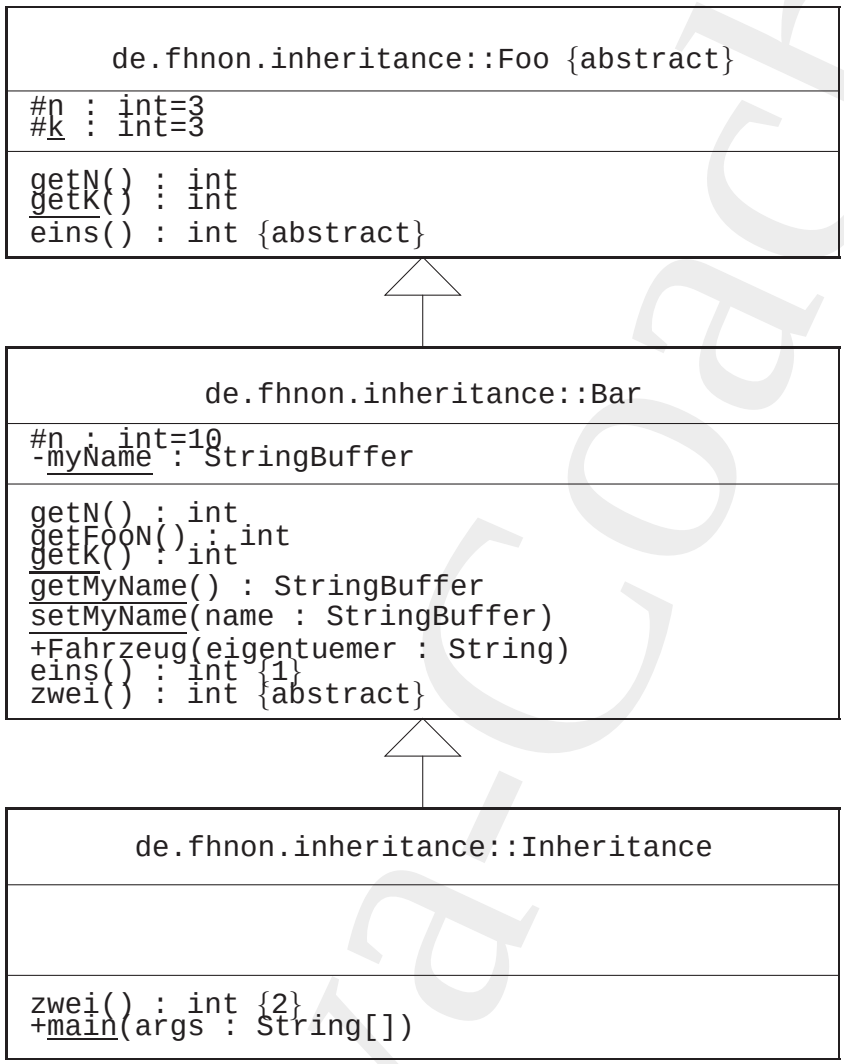


Abbildung A.2: Klassendiagramm für `Inheritance`

```
   int zwei()
   {
      return 2;
   }


   public static void main(String[] args)
   {
      Inheritance m = new Inheritance();

      Bar.setMyName(new StringBuffer("Otto AG"));
      Bar.getMyName().setCharAt(3, 'i');
      System.out.println(Bar.getMyName());

      System.out.println("Fall I  : " +
            (m.eins() + m.zwei() +
            m.getFooN() + Foo.getK()));
      System.out.println("Fall II : " +
            m.eins() + m.zwei() +
            m.getN() + Bar.getK());
   }
}
```

Klasse Foo

```
/**
 *  Kleine Kostprobe fuer: Vererbung --- abstract class
 *
 *@since      05-Apr-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.inheritance;

abstract class Foo
{
   protected int n = 3;
   protected static int k = 3;


   abstract int eins();
```

```
   int getN()
   {
      return n;
   }


   static int getK()
   {
      return k;
   }
}
```

Klasse `Bar`

```
/**
 *  Kleine Kostprobe fuer: Vererbung --- abstract class
 *
 *@since      05-Apr-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.inheritance;

abstract class Bar extends Foo
{

   protected int n = super.n + 7;
   private static StringBuffer myName;


   abstract int zwei();


   int getN()
   {
      return n;
   }


   int getFooN()
   {
      return super.getN();
```

```
   }

   static int getK()
   {
      return 2 * k;
   }

   static void setMyName(StringBuffer name)
   {
      myName = name;
   }

   static StringBuffer getMyName()
   {
      return myName;
   }

   int eins()
   {
      return 1;
   }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.fhnon.inheritance.Inheritance
```

A.10 Applikation TableProg

Die Abbildung A.3 S. 416 zeigt das Klassendiagramm der Applikation TableProg.

Klasse TableProg

```
/**
 *  Kleine Kostprobe fuer einen Fehler der nicht von javac
 *  erkannt wird Idee entnommen aus Adam Freeman / Darrel
 *  Ince; activeJava, 1996, p.105. (Quellcode stark
```

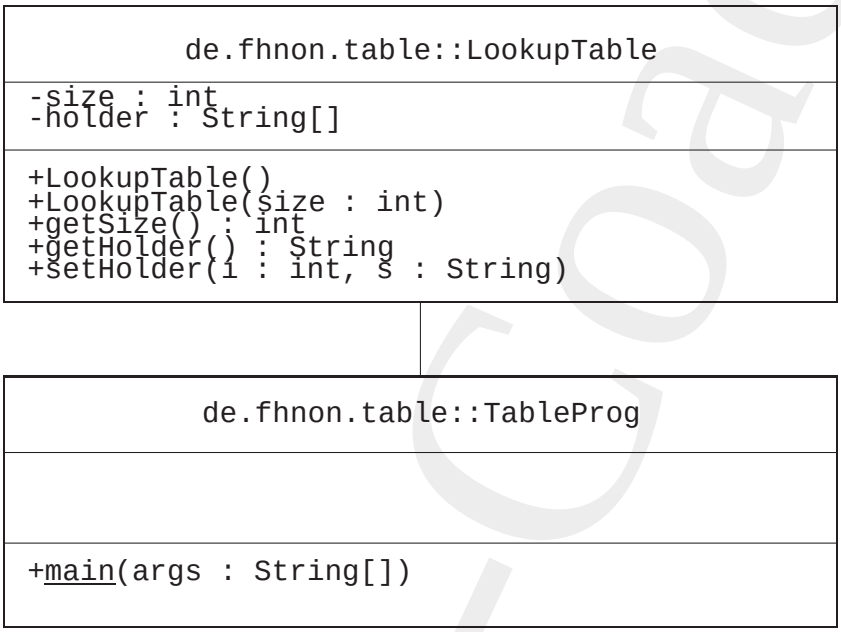


Abbildung A.3: Klassendiagramm für `TableProg`

```
 *  modifiziert.)
 *
 *@since      13-Apr-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.table;

public class TableProg
{

   public static void main(String args[])
   {
      LookupTable myTable = new LookupTable();
      myTable.setHolder(
            99, "Alles richtig, oder was?");
      System.out.println(
            "Tabelle mit " +
            myTable.getSize() +
            " erzeugt!");
      System.out.println(myTable.getHolder(99));
   }
}
```

Klasse `LookupTable`

```
/**
 *  Kleine Kostprobe fuer einen Fehler der nicht von javac
 *  erkannt wird Idee entnommen aus Adam Freeman / Darrel
 *  Ince; activeJava, 1996, p.105. (Quellcode stark
 *  modifiziert.)
 *
 *@since      13-Apr-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.table;

final class LookupTable
{
   private int size;
```

```
   private String holder[];


   LookupTable()
   {
      this(100);
   }


   LookupTable(int size)
   {
      this();
      this.size = size;
      holder = new String[size];
   }


   public int getSize()
   {
      return size;
   }


   public String getHolder(int i)
   {
      return holder[i];
   }


   public void setHolder(int i, String s)
   {
      holder[i] = s;
   }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.fhnon.table.TableProg
```

A.11 Applikation `Rekursion`

Klasse `Rekursion`

```
/**
```

```
 *  Kleine Kostprobe fuer eine Rekursion Beispiel Fakultaet:
 *  n! = n * (n - 1)!
 *
 *@since      10-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 *@created    28-Dec-2002
 */

package de.fhnon.rekursion;

import java.math.*;

public class Rekursion
{

   public static void main(String[] args)
   {

      Fakultaet foo = new Fakultaet();

      String in;
      if (args.length == 0)
      {
         in = "0";
      } else
      {
         in = args[0].replace('+', '0');
      }

      long k = Long.parseLong(in);
      long grenze = Long.MAX_VALUE;

      if (k <= grenze && k >= 0)
      {
         System.out.println(
               "Fakultaetsfunktion: fac(" + k +
               ") = " + foo.fac(k));
      } else
      {
         System.out.println("Wert = " + k +
               " kann nicht berechnet werden!");
      }

      System.out.println(
          "Anzahl der Aufrufe von fac(): " +
```

```
            Fakultaet.anzahlAufrufeFac);
   }
}
```

Klasse `Fakultaet`

```
/**
 *  Kleine Kostprobe fuer eine Rekursion Beispiel Fakultaet:
 *  n! = n * (n - 1)!
 *
 *@since      10-Apr-1998
 *@author     Hinrich Bonin
 *@created    28-Dec-2002
 *@version    1.1
 */

package de.fhnon.rekursion;

import java.math.*;

class Fakultaet
{
   /*
    *  long-Wertebereich: 64 Bit
    *  -9223372036854775808 ... 9223372036854775807
    *  BigInteger von beliebiger Groesse
    */
   BigInteger wert;
   BigInteger basisWert = new BigInteger("1");

   static long anzahlAufrufeFac = 0;


   BigInteger fac(long n)
   {
      anzahlAufrufeFac += 1;

      if (n == 0)
      {
         return basisWert;
      } else
      {
         System.out.println("Aufruf  n = " + n);

         wert = this.fac(n - 1).multiply(
```

```
                new BigInteger(Long.toString(n)));

            System.out.println("Rueckgabe wert = " +
                  wert.toString());
            return wert;
        }
    }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.fhnon.rekursion.Rekursion 4
```

A.12 Applikation `Durchschnitt` mit `HashMap`

Die `HashMap`-Klasse entpricht weitgehend der `Hashtable`-Klass bis auf die Ausnahme, dass sie `unsynchronized` ist und `nulls` erlaubt.

Klasse `Fach`

```
/**
 *  Fach mit Teilnehmeranzahl und Durchschnittsnote
 *
 *@author     Bonin
 *@created    18. Mai 2005
 */
package de.uni_lueneburg.as.durchschnitt;

import java.util.Collection;
import java.util.HashMap;

public class Fach {

        private static HashMap faecherListe = new HashMap();

        private String bezeichnung;
        private int teilnehmer;
        private double durchschnittsNote;


        /**
```

```
     *  Constructor for the Fach object
     *
     *@param  bezeichnung  Name des Faches
     */
    public Fach(String bezeichnung) {
            this.bezeichnung = bezeichnung;
            faecherListe.put(bezeichnung, this);
    }


    /**
     *  Gets the faecherListe attribute of the Fach class
     *
     *@return    The faecherListe value
     */
    public static Collection getFaecherListe() {
            return faecherListe.values();
    }


    /**
     *  Gets the fach attribute of the Fach class
     *
     *@param  bezeichnung  Description of the Parameter
     *@return              The fach value
     */
    public static Fach getFach(String bezeichnung) {
            return (Fach) faecherListe.get(bezeichnung);
    }


    /**
     *  Gets the bezeichnung attribute of the Fach object
     *
     *@return    The bezeichnung value
     */
    public String getBezeichnung() {
            return bezeichnung;
    }


    /**
     *  Gets the durchschnittsNote attribute of the Fach object
     *
     *@return    The durchschnittsNote value
     */
```

```
        public double getDurchschnittsNote() {
                return durchschnittsNote;
        }


        /**
         *  Berechnet die neue Durchschnittsnote
         *
         *@param  note  Einzelnote des Studenten
         */
        public void aktualisiereDurchschnittsNote(double note) {
                durchschnittsNote =
                        (durchschnittsNote * teilnehmer + note)
                         / ++teilnehmer;
        }

}
```

Klasse `Student`

```
/**
 *  Student mit Matrikelnummer und Noten
 *
 *@author     Bonin
 *@created    18. Mai 2005
 */
package de.uni_lueneburg.as.durchschnitt;

import java.util.Collection;
import java.util.HashMap;

public class Student {

        private String name;
        private String matrikel;
        private HashMap noten = new HashMap();


        /**
         *  Constructor for the Student object
         *
         *@param  name      Vor- und Zuname
         *@param  matrikel  Matrikelnummer
         */
        public Student(String name, String matrikel) {
```

```
        this.name = name;
        this.matrikel = matrikel;
}


/**
 *  Sets the note attribute of the Student object
 *
 *@param  fach  Objekt der Klasse Fach
 *@param  note  Note in diesem Fach
 */
public void setNote(Fach fach, double note) {
        noten.put(fach, new Double(note));
        fach.aktualisiereDurchschnittsNote(note);
}


/**
 *  Gets the matrikel attribute of the Student object
 *
 *@return    The matrikel value
 */
public String getMatrikel() {
        return matrikel;
}


/**
 *  Sets the matrikel attribute of the Student object
 *
 *@param  matrikel  The new matrikel value
 */
public void setMatrikel(String matrikel) {
        this.matrikel = matrikel;
}


/**
 *  Gets the name attribute of the Student object
 *
 *@return    The name value
 */
public String getName() {
        return name;
}
```

```
        /**
         *  Sets the name attribute of the Student object
         *
         *@param  name  The new name value
         */
        public void setName(String name) {
                this.name = name;
        }


        /**
         *  Gets the noten attribute of the Student object
         *
         *@return    The noten value
         */
        public Collection getNoten() {
                return noten.values();
        }


        /**
         *  Gets the note attribute of the Student object
         *
         *@param  fach  Description of the Parameter
         *@return       The note value
         */
        public double getNote(Fach fach) {
                return ((Double)
                        noten.get(fach)).doubleValue();
        }
}
```

Klasse `DurchschnittProg`

```
/**
 *  Applikation zur Durchschnittsnotenberechnung
 *
 *@author     Bonin
 *@created    18. Mai 2005
 */
package de.uni_lueneburg.as.durchschnitt;

public class DurchschnittProg {
```

```
        /**
         *  The main program for the DurchschnittProg class
         *
         *@param  args  The command line arguments
         */
        public static void main(String[] args) {

                Student student1 = new Student("Ewin Ente", "12345");
                Student student2 = new Student("Klara Witwe", "444444");
                Student student3 = new Student("Emma Schulze", "98765");

                Fach prog = new Fach("Programmierung");
                Fach theo = new Fach("Theoretische Informatik");

                student1.setNote(Fach.getFach("Programmierung"), 3.0);
                student2.setNote(Fach.getFach("Programmierung"), 2.0);
                student3.setNote(Fach.getFach("Programmierung"), 1.0);

                student1.setNote(theo, 1.0);
                student2.setNote(theo, 2.0);
                student3.setNote(theo, 2.0);

                System.out.println("Note von " + student1.getName() + ", "
                                + prog.getBezeichnung() + ": "
                                + student1.getNote(theo));
                System.out.println("\nDurchschnittsnoten:");
                System.out.println(prog.getBezeichnung() + " = "
                                + prog.getDurchschnittsNote());
                System.out.println(theo.getBezeichnung() + " = "
                                + (double) Math.round(
                                  theo.getDurchschnittsNote() * 10) / 10.0);

        }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.uni_lueneburg.as.durchschnitt.DurchschnittProg
```

A.13 Assoziation: Foo ↔ Bar

Klasse Foo

```
/**
 *  Assoziationsbeispiel Foo --> Bar --> Foo
 *
 *@author     Bonin 1-Dec-98
 *@created    3. Dezember 2002
 *@version    1.0
 */

package de.fhnon.assozi;

class Foo
{
   private Bar v;
   public static Foo c;


   public Bar getV()
   {
      return v;
   }


   public void setV(Bar v)
   {
      this.v = v;
   }


   public static void main(String[] args)
   {
      Foo b = new Foo();
      b.setV(new Bar());
      Foo.c = b;
      if (Foo.c.getV() instanceof Bar)
      {
         System.out.println("Alles durchdacht? Foo!");
      }
   }
}
```

Klasse `Bar`

```
/**
 *  Assoziationsbeispiel Bar --> Foo --> Bar
 *
 *@author     Bonin 1-Dec-98
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.assozi;

class Bar
{
   private Foo v;
   public static Bar c;


   public Foo getV()
   {
      return v;
   }


   public void setV(Foo v)
   {
      this.v = v;
   }


   public static void main(String[] args)
   {
      Bar b = new Bar();
      b.setV(new Foo());
      Bar.c = b;
      Bar.c.getV().setV(new Bar());
      if (Bar.c.getV().getV() instanceof Bar)
      {
         System.out.println(
               "Alles durchdacht? Bar!");
         Foo.main(new String[0]);
      } else
      {
         System.out.println("OK");
      }
   }
}
```

Geben Sie bei den folgenden Aufrufen das jeweilige Ergebnisse an:

```
>javac de/fhnon/assozi/Foo.java
>dir de\fhnon\assozi\*.class
>java de.fhnon.assozi.Bar
>java de.fhnon.assozi.Foo
```

A.14 Gleichnamige Attributen: SlotI

Klasse SlotI

```
/**
 *  Beispiel zur Frage der Vererbung gleichnamigen Slots
 *  (Attributen)
 *
 *@since      21-Dec-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.vererbung;

class SlotI
{

   public static void main(String[] args)
   {
      Foo f = new Foo();
      System.out.println(
            "f.getI() = " + f.getI());

      Bar b = new Bar();
      System.out.println(
            "b.getI() = " + b.getI());

      b.setI(3);
      System.out.println(
            "b.setI(3) dann b.getI() = " +
            b.getI());
      System.out.println(
```

```
            "b.setI(3) dann b.i = " +
            b.i);

      f.setI(4);
      System.out.println(
            "f.setI(4) dann f.getI() = " +
            f.getI());
      System.out.println(
            "f.setI(4) dann b.getI() = " +
            b.getI());
   }
}
```

Klasse Foo

```
/**
 *  Beispiel zur Frage der Vererbung gleichnamigen Slots
 *  (Attributen)
 *
 *@since      21-Dec-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.vererbung;

class Foo
{
   private int i = 1;


   public int getI()
   {
      return this.i;
   }


   public void setI(int i)
   {
      this.i = i;
   }
}
```

Klasse `Bar`

```
/**
 *  Beispiel zur Frage der Vererbung gleichnamigen Slots
 *  (Attributen)
 *
 *@since      21-Dec-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.vererbung;

class Bar extends Foo
{

   int i = 2;
}
```

Geben Sie bei den folgenden Aufrufen das jeweilige Ergebnisse an:

```
>javac de/fhnon/vererbung/SlotI.java
>java de.fhnon.vererbung.SlotI
```

A.15 Applikation `QueueProg` — Fall I

Hinweis: Aufgabenidee aus [Freeman/Ince96] entnommen. Quellcode stark modifiziert und ergänzt.

Klasse `Queue`

```
/**
 *  Kleine Kostprobe fuer eine ,,Zirkulaere Liste'' mit dem
 *  Prinzip ,,first-in-first-out''
 *
 *@since      10-Apr-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */
```

```
package de.fhnon.queue;

final class Queue
{
   private final int queueCapacity = 3;
   private String circleList[]
          = new String[queueCapacity];
   private int noOfItemsInQueue = 0;
   private int frontOfTheQueue = 0;
   static int noOfQueues = 0;


   Queue()
   {
      noOfQueues++;
   }


   public int getQueueCapacity()
   {
      return queueCapacity;
   }


   public int getNoOfItemsInQueue()
   {
      return noOfItemsInQueue;
   }


   public boolean isQueueEmpty()
   {
      return (noOfItemsInQueue == 0);
   }


   public boolean isQueueFull()
   {
      return (noOfItemsInQueue == queueCapacity);
   }


   private void addNthItem(
         int n, String itemToBeAdded)
   {
      int index;
```

```
      index = frontOfTheQueue + (n - 1);
      if (index >= queueCapacity)
      {
         index = index % queueCapacity;
      }
      circleList[index] = itemToBeAdded;
   }


   public boolean addItem(
         String itemToBeAdded)
   {
      if (this.isQueueFull())
      {
         System.out.println(
               "Item = " + itemToBeAdded +
               " nicht aufgenommen!");
         return false;
      } else
      {
         noOfItemsInQueue++;
         this.addNthItem(
               noOfItemsInQueue, itemToBeAdded);
         return true;
      }
   }


   public String getFirstItem()
   {
      if (this.isQueueEmpty())
      {
         return "";
      } else
      {
         return circleList[frontOfTheQueue];
      }
   }


   public boolean removeFirstItem()
   {
      if (this.isQueueEmpty())
      {
         System.out.println("Kein Item entfernbar");
         return false;
```

```
      } else
      {
         noOfItemsInQueue--;
         if (frontOfTheQueue == (queueCapacity - 1))
         {
            frontOfTheQueue = 0;
         } else
         {
            frontOfTheQueue++;
         }
         return true;
      }

   }


   public boolean isItemInQueue(String item)
   {
      int count = 0;
      while (count < noOfItemsInQueue)
      {
         count++;
         if (this.getNthItem(count) == item)
         {
            return true;
         }
      }
      return false;
   }


   private String getNthItem(int n)
   {
      int index;
      index = frontOfTheQueue + (n - 1);
      if (index >= queueCapacity)
      {
         index = index % queueCapacity;
      }
      return (circleList[index]);
   }
}
```

Klasse `QueueProg`

```
/**
 *  Kleine Kostprobe fuer eine ,,Zirkulaere Liste'' mit dem
 *  Prinzip ,,first-in-first-out''
 *
 *@since      10-Apr-1998
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.1
 */

package de.fhnon.queue;

public class QueueProg
{

   public static void main(String[] args)
   {

      Queue myQ = new Queue();
      Queue myL = new Queue();
      System.out.println(
            "Step 0: Queue.noOfQueues = " +
            Queue.noOfQueues);
      System.out.println(
            "Step 1: myQ.getQueueCapacity() = " +
            myQ.getQueueCapacity());

      myQ.addItem("Otto AG");
      myQ.addItem("Emma AG");
      myQ.addItem("Klara AG");
      myQ.removeFirstItem();

      System.out.println(
            "Step 2: myQ.getFirstItem() = " +
            myQ.getFirstItem());

      myQ.addItem("Willi AG");
      myQ.addItem("Ernst AG");
      myQ.removeFirstItem();
      myQ.removeFirstItem();
      myQ.addItem("Ernst AG");

      System.out.println(
            "Step 3: myQ.getFirstItem() = " +
            myQ.getFirstItem());
```

```
      System.out.println(
            "Step 4: myQ.getNoOfItemsInQueue = " +
            myQ.getNoOfItemsInQueue());

      boolean inCircleList = myQ.isItemInQueue("Ernst AG");
      System.out.println(
            "Step 5: myQ.isItemInQueue(Ernst AG) = "
             + inCircleList);
   }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.fhnon.queue.QueueProg
```

A.16 Applikation `QueueProg` — Fall II

Hinweis: Aufgabenidee aus Web-Quelle:
`http://www.uni-klu.ac.at/~thaichho/java/k100098.html`
(online 30-May-2005) entnommen. Quellcode stark modifiziert und ergänzt.

Interface `Queue`

```
package de.uni_lueneburg.as.queue;

import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 *  Description ofQueueProg the Interface
 *
 *@author     bonin
 *@created    30. Mai 2005
 */
public interface Queue {

        /**
         *  Fügt Element hinzu
         *
         *@param  o  Element
         *@return    True or False
```

```
         */
        public boolean add(Object o);


        /**
         *  Entnimmt erste Element
         *
         *@return   Element
         *@exception  NoSuchElementException
         */
        public Object retrieve()
          throws NoSuchElementException;


        /**
         *  Iteriert über alle Elemente
         *
         *@return Iterator
         */
        public Iterator iterator();
}
```

Klasse LinkedQueue

```
package de.uni_lueneburg.as.queue;

import java.io.Serializable;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 *  Implementiert Queue
 *
 *@author    bonin
 *@created   30. Mai 2005
 */
public class LinkedQueue implements Queue, Serializable {

        /**
         *  ElementWrappper ist interne Class
         */
        protected ElementWrapper first;
        protected ElementWrapper last;

        protected int count;
```

```
public LinkedQueue() {
        first = last = null;
        count = 0;
}


/**
 *  Fügt Element hinzu
 *
 *@param  o  Element
 *@return    True or False
 */
public boolean add(Object o) {
        if (count == 0) {
                first = new ElementWrapper();
                last = first;
                count = 1;
        } else {
                last.next = new ElementWrapper();
                last = last.next;
                ++count;
        }
        last.element = o;
        last.next = null;
        return true;
}


/**
 *  Entnimmt erstes Element
 *
 *@return     Element
 *@exception  NoSuchElementException
 */
public Object retrieve()
                throws NoSuchElementException {
        if (count <= 0) {
                throw new NoSuchElementException();
        }
        ElementWrapper ret = first;
        --count;
        first = first.next;
        if (first == null) {
                last = null;
                count = 0;
```

```
        }
        return ret.element;
}


/**
 *  Iterator
 *
 *@return Iterator
 */
public Iterator iterator() {
        return
                new Iterator() {
                        ElementWrapper tmp = first;


                        public boolean hasNext() {
                                return tmp != null;
                        }


                        public Object next() {
                                if (tmp == null) {
                                        throw new
                                NoSuchElementException();
                                }
                                Object ret = tmp.element;
                                tmp = tmp.next;
                                return ret;
                        }


                        public void remove() {
                                throw new
                          UnsupportedOperationException();
                        }
                };
}


class ElementWrapper implements Serializable {

        public Object element;

        public ElementWrapper next;
}
```

```
}
```

Klasse QueueProg

```
package de.uni_lueneburg.as.queue;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Iterator;

/**
 *  Description of the Class
 *
 *@author     bonin
 *@created    30. Mai 2005
 */
public class QueueProg {

        static LinkedQueue queue = new LinkedQueue();

        /**
         *  Zeigt die Elemente der Queue
         */
        private static void showQueue() {
                System.out.println("\nElemente in der Queue:");
                Iterator it = queue.iterator();
                while (it.hasNext()) {
                        System.out.println(it.next().toString());
                }
        }


        /**
         *  Erzeugt eine Test-Queue
         *
         *@param anzahl  Anzahl der Elemente
         */
        private static void testQueue(int anzahl) {
                for (int i = 1; i <= anzahl; ++i) {
                        queue.add(Integer.toString(i));
                }
        }
```

```
/**
 *  Schreibt die Queue in eine Datei
 *
 *@param  dateiname
 */
private static void writeQueueToFile(String dateiname) {
        try {
                FileOutputStream fileout =
                  new FileOutputStream(dateiname);
                ObjectOutputStream objout =
                  new ObjectOutputStream(fileout);
                objout.writeObject(queue);
                objout.close();
                System.out.println("\nQueue in Datei "
                                   + dateiname
                                   + " gespeichert.");
        } catch (Exception e) {
                e.printStackTrace(System.out);
        }

}


/**
 *  Liest die Queue aus der Datei
 *
 *@param  dateiname
 */
private static void readQueueFromFile(String dateiname) {
        try {
                FileInputStream filein =
                  new FileInputStream(dateiname);
                ObjectInputStream objectin =
                  new ObjectInputStream(filein);
                queue = (LinkedQueue)
                  objectin.readObject();
                System.out.println(
                  "\nQueue wurde aus Datei "
                  + dateiname
                  + " gelesen.");

        } catch (Exception e) {
                e.printStackTrace(System.out);
        }
}
```

```
/**
 *  The main program for the QueueProg class
 *
 *@param  args  Anzahl Elemente in Testqueue
 */
public static void main(String[] args) {
        int anzahl = 1;
        try {
                anzahl = Integer.parseInt(args[0]);
        } catch (ArrayIndexOutOfBoundsException e) {
                System.err.println(
                  "Keine Elementeanzahl angegeben!");
                System.exit(1);
        } catch (NumberFormatException e) {
                System.err.println(
                  "Elementeanzahl nicht interpretierbar!");
                System.exit(1);
        }
        if (anzahl < 1) {
                anzahl = 1;
                System.out.println(
          "Die Queue wurde auf Anfangsgroesse = 1 gesetzt!");
        }

        QueueProg.testQueue(anzahl);
        QueueProg.showQueue();

        Object head = queue.retrieve();
        if (head != null) {
                System.out.println(
                  "\n1. Element entnommen: "
                  + head);

        } else {
                System.out.println("\nQueue ist leer!");
        }

        QueueProg.showQueue();

        queue.add("Emma Musterfrau");
        System.out.println("Eingefügt: Emma Musterfrau");
        queue.add("Hans Otto");
        System.out.println("Eingefügt: Hans Otto");
        queue.add("Karl Stein");
        System.out.println("Eingefügt: Karl Stein");
```

```
                QueueProg.writeQueueToFile("queue.ser");
                QueueProg.readQueueFromFile("queue.ser");

                QueueProg.showQueue();
        }
}
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.uni_lueneburg.as.queue.QueueProg 3
```

A.17 Applet SimpleThread

HTML-Datei SimpleThread.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<!-- Primitives Testbett fuer Applet SimpleThread    -->
<!-- Bonin  07-May-1998                              -->
<!-- update 07-Jan-2003                              -->
<head>
<title>Mond am Himmel</title>
</head>
<body>
<h1>Mond am Himmel</h1>
<h1>
<applet name="Mond" code="SimpleThread.class"
  width="450" height="120" alt="Mond am Himmel">
  Hier soll SimpleThread-Applet laufen!
</applet>
</h1>
<p>Der Mond ist aufgegangen ...</p>
<p>Copyright Bonin 07-May-1998 all rights reserved</p>
<address>
<a href="mailto:bonin@fhnon.de">bonin@fhnon.de</a>
</address>
</body>
<!-- File C:\bonin\anwd\code\SimpleThread.html       -->
</html>
```

Klasse `SimpleThread`

```
/**
 *  Kleines Beispiel fuer eine ,,Animation mittels Thread'',
 *  Idee aus: Hubert Partl; Java-Einfuehrung, Version April
 *  1998, S. 82 http://www.boku.ac.at/javaeinf/ Quellcode
 *  leicht modifiziert
 *
 *@author     Hinrich Bonin
 *@version    1.0
 *@since      08-Mai-1998
 */
import java.applet.*;
import java.awt.*;

public class SimpleThread extends Applet
       implements Runnable
{

   int x, y, width, height;
   Graphics grafik;
   Image bild;
   Color nachtFarbe = new Color(0, 0, 102);
   Color mondFarbe = new Color(204, 204, 255);

   Thread myT = null;


   public void init()
   {
      Dimension d = getSize();
      width = d.width;
      height = d.height;
      bild = createImage(width, height);
      grafik = bild.getGraphics();
      x = width / 2;
      y = height / 2;
      System.out.println("x = " + x + " y = " + y);
   }


   public void start()
   {
      if (myT == null)
      {
         myT = new Thread(this);
         myT.start();
```

```
      }
      System.out.println("start() appliziert");
   }


   public void stop()
   {
      if (myT != null)
      {
         myT.stop();
         myT = null;
      }
      System.out.println("stop() appliziert");
   }


   public void run()
   {
      while (true)
      {
         grafik.setColor(nachtFarbe);
         grafik.fillRect(0, 0, width, height);
         grafik.setColor(mondFarbe);
         grafik.fillArc(x, y - 25, 50, 50, 270, 180);
         x += 2;
         if (x > width + 50)
         {
            x = -50;
         }
         repaint();
         try
         {
            System.out.println(
                  "In run() vor Thread.sleep(1000)");
            Thread.sleep(1000);
            System.out.println(
                  "In run() nach Thread.sleep(1000)");
         } catch (InterruptedException e)
         {
            System.out.println(
                  "In run() Fehler");
         }
      }
   }
```

```
   public void paint(Graphics g)
   {
      update(g);
   }


   public void update(Graphics g)
   {
      if (bild != null)
      {
         g.drawImage(bild, 0, 0, this);
      }
   }
}
```

Skizzieren Sie bei dem folgenden Aufruf das Ergebnisse:

```
>appletviewer SimpleThread.html
```

Oder nutzen Sie einen Browser mit einer Java 2 Plattform.

A.18 Applet DemoAWT

HTML-Datei ExampleAWT

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<!-- Testbett fuer Applet DemoAWT                    -->
<!-- Bonin  19-April 1998                            -->
<!-- Update 08-Jan-2003                              -->
<head>
<title>Willi liebt Sport</title>
</head>
<body>
<h1>Willi liebt Sport</h1>
<h1>
  <applet name="Willi"
    code="de.fhnon.awt.DemoAWT.class"
    width="350" height="120"
    alt="Willi will Ausdauersport">
    Willi will Ausdauersport!
  </applet>
```

```
</h1>
<p>Copyright Bonin 1998--2003 all rights reserved</p>
<address>
<a href="mailto:bonin@fhnon.de">bonin@fhnon.de</a>
</address>
</body>
<!-- File C:\bonin\anwd\code\ExampleAWT.html       -->
</html>
```

Klasse MaskeAufbau

```
/**
 *  Kleines Beispiel fuer die ,,Abstract Window Toolkit''
 *  Bibliothek
 *
 *@since      14-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.awt;

import java.applet.Applet;
import java.awt.*;

class MaskeAufbau extends Applet
{
   Panel topPanel,
         leftPanel,
         centerPanel,
         rightPanel,
         bottomPanel;


   public void doUserInterface(Frame frame)
   {
      frame.setLayout(new BorderLayout());
      topPanel = new Panel();
      leftPanel = new Panel();
      centerPanel = new Panel();
      rightPanel = new Panel();
      bottomPanel = new Panel();
      frame.add("North", topPanel);
      frame.add("West", leftPanel);
      frame.add("Center", centerPanel);
      frame.add("East", rightPanel);
```

```
      frame.add("South", bottomPanel);

      MenuBar myMbar = new MenuBar();

      Menu myMTria = new Menu("Triathlon");
      myMTria.add(new MenuItem("Schwimmen"));
      myMTria.add(new MenuItem("Radfahren"));
      myMTria.add(new MenuItem("Laufen"));
      myMbar.add(myMTria);

      Menu myMDua = new Menu("Duathlon");
      myMDua.add(new MenuItem("1. Laufen"));
      myMDua.add(new MenuItem("Radfahren"));
      myMDua.add(new MenuItem("2. Laufen"));
      myMbar.add(myMDua);

      frame.setMenuBar(myMbar);
   }
}
```

Klasse `MyCanvas`

```
/**
 *  Kleines Beispiel fuer die ,,Abstract Window Toolkit''
 *  Bibliothek
 *
 *@since      14-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.awt;

import java.awt.*;

class MyCanvas extends Canvas
{

   public final int width = 80;
   public final int height = 120;


   public void paint(Graphics g)
   {
      /*
```

```
         *  x-Achse:  waagerecht von links nach rechts
         *  y-Achse:  senkrecht von oben nach unten
         *  rgbWert:  jeweils 0...255
         */
        int x;
        /*
         *  x-Achse:  waagerecht von links nach rechts
         *  y-Achse:  senkrecht von oben nach unten
         *  rgbWert:  jeweils 0...255
         */
        int y;
        /*
         *  x-Achse:  waagerecht von links nach rechts
         *  y-Achse:  senkrecht von oben nach unten
         *  rgbWert:  jeweils 0...255
         */
        int rgbWert;
        for (x = 0, y = 0, rgbWert = 0;
              (x < (width / 2)) &&
              (y < (height / 2) && (rgbWert < 256));
              x += 2, y += 2, rgbWert += 6)
        {
           g.setColor(new Color(255 - rgbWert,
                 rgbWert, 0));
           g.fillRect(x, y, width - (2 * x),
                 height - (2 * y));
        }
        g.setColor(Color.blue);
        g.drawString("D T U",
              (width - g.getFontMetrics().stringWidth(
              "D T U")) / 2,
              height / 2);
    }


    /*
     *  minimumSize() wird vom Layout-Manger aufgerufen,
     *  um zu erfahren, wie gross der minimale Platz ist,
     *  der benoetigt wird.
     */
    public Dimension minimumSize()
    {
        return new Dimension(width + 20, height + 20);
    }
```

```
   /*
    *  preferredSize() wird vom Layout-Manager aufgerufen,
    *  um zu erfahren, wie gross man es gern haette.
    */
   public Dimension preferredSize()
   {
      return this.minimumSize();
   }
}
```

Klasse `SimpleListener`

```
/**
 *  Kleines Beispiel fuer die ,,Abstract Window Toolkit''
 *  Bibliothek
 *
 *@since      14-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.awt;

import java.awt.*;
import java.awt.event.ActionListener;

class SimpleListener implements ActionListener
{

   private Frame fr;


   public SimpleListener(Frame f)
   {
      fr = f;
   }


   public void actionPerformed(
         java.awt.event.ActionEvent e)
   {
      String name = e.getActionCommand();
      System.out.println(
            "actionPerformed() appliziert: " + name);
      if (name.equals("Anmelden!"))
```

```
      {
         fr.setTitle("Danke Willi!");
      }
      if (name.equals("Absagen!"))
      {
         fr.setTitle("Schade Willi!");
      }
   }
}
```

Klasse DemoAWT

```
/**
 *  Kleines Beispiel fuer die ,,Abstract Window Toolkit''
 *  Bibliothek
 *
 *@since      14-Apr-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.awt;

import java.awt.*;

public class DemoAWT extends MaskeAufbau
{

   Frame myFrame
          = new Frame(
         "Willi will Ausdauersport!");


   public void init()
   {
      DemoAWT myDemo = new DemoAWT();
      myDemo.doUserInterface(myFrame);
      myFrame.pack();
      myFrame.show();
   }


   public void stop()
   {
      System.out.println("stop() appliziert!");
```

```
   }

   public void doUserInterface(Frame frame)
   {
      super.doUserInterface(frame);

      topPanel.setLayout(new GridLayout(1, 2));
      topPanel.add(new Checkbox("DTU-Lizenz"));
      Choice myC = new Choice();
      myC.addItem("Kurzdistanz");
      myC.addItem("Mitteldistanz");
      myC.addItem("Langdistanz");
      topPanel.add(myC);

      Button anmelden = new Button("Anmelden!");
      anmelden.addActionListener(
            new SimpleListener(frame));
      leftPanel.add(anmelden);

      centerPanel.add(new MyCanvas());

      Button absagen = new Button("Absagen!");
      absagen.addActionListener(
            new SimpleListener(frame));
      rightPanel.add(absagen);

      int widthDTUinPixel = new MyCanvas().width;
      bottomPanel.add(new TextArea(
            "Beschreiben Sie genau Ihren Leistungsstand!",
            3, widthDTUinPixel / 2));
   }
}
```

Skizzieren Sie bei dem folgenden Aufruf das Ergebnis:

```
>appletviewer ExampleAWT.html
```

A.19 Innere Klasse

Die selten zu Späßen aufgelegte Programmierin Emma Klug hat die folgende Applikation `Regal` geschrieben. Sie zeigt Ihnen folgenden Protokollauszug der Arbeit.

Protokollauszug

```
>java -fullversion
java full version "JDK 1.1.6 IBM build a116-19980529" (JIT: jitc
>javac Regal.java
>java Regal\$Test
...
```

Klasse `Regal`

```
/**
 *  Inner-Classes-Beispiel
 *
 *@author     Hinrich Bonin
 *@created    26. November 2002
 *@version    1.0
 *@since      22-Jan-1999
 */
public final class Regal
{
   private static int anzahl = 0;
   private int anzahlSchubladen = 0;
   private int benutztAnzahl = 0;


   public Regal()
   {
      Regal.anzahl++;
   }


   public class Schublade
   {
      private boolean belegt = false;
      private int gezogenAnzahl = 0;
      private String inhalt = "Leer";
```

```
    public Schublade()
    {
       anzahlSchubladen++;
    }


    public String getInhalt()
    {
       benutztAnzahl++;
       gezogenAnzahl++;
       return inhalt;
    }


    public void setInhalt(String inhalt)
    {
       this.inhalt = inhalt;
       benutztAnzahl++;
       gezogenAnzahl++;
       belegt = true;

    }
}


public static class Test
{
    public static void main(String argv[])
    {
       Regal baz = new Regal();
       Regal bar = new Regal();
       Regal foo = baz;

       Regal.Schublade fooS1
              = foo.new Schublade();
       Regal.Schublade fooS2
              = foo.new Schublade();
       fooS1.setInhalt("Java-Disketten");
       fooS2.setInhalt("Java-Artikel");

       Regal.Schublade barS1
              = bar.new Schublade();
       barS1.setInhalt("Java-CD-ROM");

       System.out.println("Das Regalsystem hat " +
            Regal.anzahl + " Regal(e).");
```

```
            System.out.println("Das Regal foo hat " +
                  baz.anzahlSchubladen + " Schubladen.");

            if (fooS1.belegt || fooS2.belegt)
            {
               System.out.println(
                     "Schubladeninhalte: " +
                     "\n  " + fooS1.getInhalt() +
                     "\n  " + fooS2.getInhalt());
            }
            System.out.println("Das Regal foo wurde " +
                  baz.benutztAnzahl + "x benutzt.");
         }
      }
   }
```

A.19.1 Erzeugte Klassen feststellen

Stellen Sie fest, ob die Datei `Regal` fehlerfrei compiliert werden konnte und geben Sie an, welche Dateien nach dem Compilieren entstanden sind. Ist der Aufruf zum „Laufenlassen“ dieser Applikation korrekt?

A.19.2 Vervollständigen des Protokollauszuges

Ersetzen Sie die „drei Punkte“ des obigen Protokollauszuges durch das Ergebnis der Programmausführung.

A.20 FastObjects-Beispielprogramm `Buch`

In diesem FastObjects-Beispiel läuft der DBMS-Server auf dem Rechner (IP: 193.174.33.20) mit dem Namen `oodbserver`. Das Erzeugen des Schema `BuchScholzDict` und der Datenbank `BuchScholzDB` erfolgt auf einem anderen Rechner (IP: 193.174.33.143). Damit ein Buchobjekt mit Namen `PKS01` eingespeichert werden kann, müssen entsprechende Zugriffsrechte bei den persistenten Klassen `Buch`, `Person` und `Autor` vorliegen. Da FastObjects in seiner sogenannten Tool-Klasse `PtName` den Namen `PKS01` speichert, muß man auch für die Klasse `PtName` Schreibzugriffsrechte haben.

Poet-Parameterdatei `ptj.opt`

```
/**
  *
  * Konfigurationsdatei für das Buch-Beispiel
  *
  */

[schemata\dict]
name=BuchScholzDict
oneFile = false

[databases\base]
name=BuchScholzDB
schema=dict
oneFile = false

[classes\Buch]
persistent = true
schema=dict

[classes\Autor]
persistent = true
schema=dict

[classes\Person]
persistent = true
schema=dict
```

Klasse `Buch`

```
import com.poet.odmg.*;
import java.util.*;

public class Buch implements Constraints
{
   private String titel;
   private String isbn;
   private int erscheinungsJahr;
   private Autor hauptAutor;
   private String schlagWoerter;
   private transient int alter;
   private String verlagsKurzName;


   public String getTitel()
```

```
{
   return titel;
}


public String getIsbn()
{
   return isbn;
}


public int getErscheinungsJahr()
{
   return erscheinungsJahr;
}


public Autor getHauptAutor()
{
   return hauptAutor;
}


public String getSchlagWoerter()
{
   return schlagWoerter;
}


public int getAlter()
{
   return alter;
}


public String getVerlagsKurzName()
{
   return verlagsKurzName;
}


public void setTitel(String titel)
{
   this.titel = titel;
}
```

```
public void setIsbn(String isbn)
{
   this.isbn = isbn;
}


public void setErscheinungsJahr(
      int erscheinungsJahr)
{
   this.erscheinungsJahr = erscheinungsJahr;
}


public void setHauptAutor(Autor hauptAutor)
{
   this.hauptAutor = hauptAutor;
}


public void setSchlagWoerter(
      String schlagWoerter)
{
   this.schlagWoerter = schlagWoerter;
}


public void setAlter(int alter)
{
   this.alter = alter;
}


public void setVerlagsKurzName(
      String verlagsKurzName)
{
   this.verlagsKurzName = verlagsKurzName;
}


// Methode zur Rekonstruktion

public void postRead()
{
   Calendar cal = Calendar.getInstance();
   int heute = cal.get(cal.YEAR) - 1900;
```

```
      this.setAlter(heute -
            this.getErscheinungsJahr());
   }


   // Methoden zur Interfaceerfüllung

   public void preWrite()
   {
      System.out.println(
            "preWrite-Methode appliziert!");
   }


   public void preDelete()
   {
      System.out.println(
            "preDelete-Methode appliziert!");
   }
}
```

Klasse `Autor`

```
import com.poet.odmg.*;
import java.util.*;

public class Autor extends Person
{
   private String themen;
   private String orgKurzName;


   public Autor() { }


   public Autor(String name)
   {
      this();
      this.setZuName(name);
   }


   public String getThemen()
   {
      return themen;
```

```
   }


   public String getOrgKurzName()
   {
      return orgKurzName;
   }


   public void setThemen(String themen)
   {
      this.themen = themen;
   }


   public void setOrgKurzName(
         String orgKurzName)
   {
      this.orgKurzName = orgKurzName;
   }
}
```

Klasse `Person`

```
import com.poet.odmg.*;
import java.util.*;

public class Person
{
   private String zuName;
   private String vorNamen;


   public String getZuName()
   {
      return zuName;
   }


   public String getVorNamen()
   {
      return vorNamen;
   }
```

```
   public void setZuName(String zuName)
   {
      this.zuName = zuName;
   }


   public void setVorNamen(String vorNamen)
   {
      this.vorNamen = vorNamen;
   }
}
```

Klasse BuchBind

```
import com.poet.odmg.*;
import org.odmg.ODMGException;
import org.odmg.ObjectNameNotUniqueException;
import org.odmg.ODMGRuntimeException;

public class BuchBind
{

   public static void main(String[] argv)
          throws ODMGException
   {
      Database myDB = new Database();
      myDB.open("poet://oodbserver/BuchScholzDB",
            Database.OPEN_READ_WRITE);
      Transaction myT = new Transaction(myDB);
      myT.begin();
      try
      {
         Buch myBuch = new Buch();
         myBuch.setTitel(
               "Softwarekonstruktion mit LISP");
         myBuch.setIsbn("3-11-011786-X");
         myBuch.setErscheinungsJahr(91);
         myBuch.setHauptAutor(new Autor("Bonin"));
         myBuch.setSchlagWoerter(
               "Arbeitstechniken, Qualität");
         myBuch.postRead();
         myBuch.setVerlagsKurzName(
               "WalterDeGruyter");

         System.out.println(
```

```
                "Zuname des Autors: " +
                myBuch.getHauptAutor().getZuName() +
                "\nAlter des Buches : " +
                myBuch.getAlter());
            myDB.bind(myBuch, "PKS01");
         } catch (ObjectNameNotUniqueException exc)
         {
            System.out.println("PKS01 gibt es schon!");
         } catch (ODMGRuntimeException exc)
         {
            myT.abort();
            throw exc;
         }
         myT.commit();
         myDB.close();
      }
}
```

Klasse `BuchLookUp`

```
import com.poet.odmg.*;
import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;

public class BuchLookup
{

   public static void main(String[] argv)
          throws ODMGException
   {
      Database myDB = new Database();
      myDB.open(
            "poet://oodbserver/BuchScholzDB",
            Database.OPEN_READ_WRITE);
      Transaction myT = new Transaction(myDB);
      myT.begin();
      try
      {
         Buch myBuch =
               (Buch) myDB.lookup("PKS01");
         System.out.println(
               "Zuname des Autors: " +
               myBuch.getHauptAutor().getZuName() +
               "\nAlter des Buches: " +
               myBuch.getAlter());
```

```
      } catch (ODMGRuntimeException exc)
      {
         myT.abort();
         throw exc;
      }
      myT.commit();
      myDB.close();
   }
}
```

Klasse `ListeLookUp`

```
/**
 *  Selektieren und Rekonstruieren von mehreren FastObjects
 *  (Buechern)
 *
 *@author     Hinrich Bonin
 *@version    1.0
 */
import com.poet.odmg.util.*;
import com.poet.odmg.*;
import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;
import java.util.*;

public class ListeLookup
{
   public static void main(String[] argv)
          throws ODMGException
   {
      Database myDB = new Database();
      myDB.open(
           "poet://oodbserver/BuchScholzDB",
           Database.OPEN_READ_WRITE);
      Transaction myT = new Transaction(myDB);
      myT.begin();
      try
      {
         String query =
               "define extent alleBuecher for Buch;" +
               "select buch from buch in alleBuecher";
         OQLQuery abfrage = new OQLQuery(query);
         Object result = abfrage.execute();
         Iterator e =
               ((CollectionOfObject) result).iterator();
```

```
         while (e.hasNext())
         {
            Buch buch = (Buch) e.next();
            System.out.println(buch.getTitel());
         }
      } catch (ODMGRuntimeException ore)
      {
         myT.abort();
         ore.printStackTrace();
      }
      myT.commit();
      myDB.close();
   }
}
```

Skizzieren Sie bei dem folgenden Aufruf das Ergebnis:

```
>java BuchLookup
```

A.21 Vererbung

Die folgende Java-Quellcodedatei `Foo.java` wurde fehlerfrei compiliert.

```
>java -fullversion
java full version "1.4.0_01-b03"
>javac de/fhnon/innerclass/Foo.java
>
```

Klasse `Foo`

```
/**
 *  Vererbungsbeispiel
 *
 *@since      30-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.innerclass;
```

```
public class Foo
{
   public static class KlasseA
   {
      public static int updateAnzahlSlot = 0;
      private String slot = "KlasseA";


      public String getSlot()
      {
         return slot;
      }


      public void setSlot(String slot)
      {
         updateAnzahlSlot = updateAnzahlSlot + 1;
         this.slot = slot;
      }
   }


   public static class KlasseB extends KlasseA
   {
      public static int updateAnzahlSlot = 0;
      private String slot = "KlasseB";


      public String getSlot()
      {
         return slot;
      }


      public void setSlot(String slot)
      {
         updateAnzahlSlot = updateAnzahlSlot + 1;
         this.slot = slot;
      }
   }


   public static class KlasseC extends KlasseB
   {
      public static int updateAnzahlSlot = 0;
```

```
      private String slot = "KlasseC";


      public String getSlot()
      {
         return slot;
      }


      public void setSlot(String slot)
      {
         updateAnzahlSlot = updateAnzahlSlot + 1;
         this.slot = slot;
      }
   }


   public static class Bar
   {
      public static void main(String[] args)
      {
         KlasseA a = new KlasseA();
         KlasseB b = new KlasseB();
         KlasseC c = new KlasseC();
         b.setSlot(a.getSlot());
         c.setSlot(b.getSlot());
         a.setSlot(c.getSlot());
         System.out.println(
               "Slot-Wert in Instanz c: " +
               c.getSlot() +
               "\nAnzahl der Änderungen in KlasseA: " +
               KlasseA.updateAnzahlSlot);
      }
   }
}
```

A.21.1 Erzeugte Dateien angeben

Geben Sie an, welche Dateien nach dem Compilieren von `Foo.java` entstanden sind.

A.21.2 Java-Aufruf angeben

Ersetzen Sie im folgenden Aufruf die drei Punkte.

```
>java de.fhnon.innerclass.Foo...
>
```

Geben Sie an, für welche Plattform (AIX oder NT) Ihr Ersatz gilt.

A.21.3 Ergebnis des `java`-Aufrufes angeben

Geben Sie das Ergebnis Ihres Aufrufes an.

A.22 Read-Write-File-Programm schreiben

Die Datei `TelefonBuchProg.java` enthält den Java-Quellcode für ein sehr einfaches Telefonbuch. Notiert ist dabei primär nur der Teil, der für die Persistenz der Eintragen in das Telefonbuch sorgt.

Klasse `TelefonBuchProg`

```
/**
 *  Einfaches permanentes Telefonbuch mit Schreibtest
 *
 *@since      29-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.telefon;

import java.io.*;
import java.util.*;

public class TelefonBuchProg
{

   public static void main(String argv[])
   {
      TelefonBuch t = new TelefonBuch();
      t.addEintrag("Key1",
            new TelefonEintrag(
            "Otto", "+49/4131/677175"));
      t.addEintrag("Key2",
            new TelefonEintrag(
            "Emma", "+49/4131/677144"));
      try
```

```
      {
         FileOutputStream fout =
               new FileOutputStream("tbuch.ser");
         ObjectOutputStream out =
               new ObjectOutputStream(fout);
         out.writeObject(t);
         out.close();
         /*
          *  Zur Kontrolle: Wiedereinlesen und Vergleichen
          */
         FileInputStream fin =
               new FileInputStream("tbuch.ser");
         ObjectInputStream in =
               new ObjectInputStream(fin);
         TelefonBuch copy =
               (TelefonBuch) in.readObject();
         in.close();
         if (t.gleichheit(copy))
         {
            System.out.println(
                  "OK --- Objekte sind gleich!");
         } else
         {
            System.out.println(
                  "Fehler --- Objekte sind ungleich!");
         }
      } catch (Exception e)
      {
         e.printStackTrace(System.out);
      }
   }
}
```

Klasse `TelefonBuch`

```
/**
 *  Einfaches permanentes Telefonbuch mit Schreibtest
 *
 *@since      29-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.telefon;
```

```
import java.io.*;
import java.util.*;

public class TelefonBuch implements Serializable
{
   Hashtable tabelle;


   public TelefonBuch()
   {
      tabelle = new Hashtable();
   }


   public TelefonEintrag getEintrag(String key)
   {
      return (TelefonEintrag) tabelle.get(key);
   }


   public TelefonEintrag addEintrag(
         String key, TelefonEintrag te)
   {
      return (TelefonEintrag) tabelle.put(key, te);
   }


   public int size()
   {
      return tabelle.size();
   }


   public boolean gleichheit(TelefonBuch t)
   {
      if ((t == null) || (size() != t.size()))
      {
         return false;
      }
      Enumeration keys = tabelle.keys();
      while (keys.hasMoreElements())
      {
         String key = (String) keys.nextElement();
         TelefonEintrag myTe = getEintrag(key);
         TelefonEintrag otherTe = t.getEintrag(key);
         if (!myTe.gleichheit(otherTe))
```

```
         {
            return false;
         }
      }
      return true;
   }
}
```

Klasse `TelefonEintrag`

```
/**
 *  Einfaches permanentes Telefonbuch mit Schreibtest
 *
 *@since      29-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.telefon;

import java.io.*;

public class TelefonEintrag implements Serializable
{

   private String kurzname;
   private String telefon;


   public String getKurzname()
   {
      return kurzname;
   }


   public String getTelefon()
   {
      return telefon;
   }


   public TelefonEintrag(
         String kurzname, String telefon)
   {
      if ((kurzname == null) || (telefon == null))
```

```
        {
           throw new IllegalArgumentException();
        }
        this.kurzname = kurzname;
        this.telefon = telefon;
        System.out.println("TelefonEintrag: " +
              kurzname + " " + telefon);
    }


    public boolean gleichheit(
          TelefonEintrag te)
    {
        return
              (getKurzname().equalsIgnoreCase(
              te.getKurzname())) &&
              (getTelefon().equalsIgnoreCase(
              te.getTelefon()));
    }
}
```

A.22.1 Ergebnis von `java TelefonBuchProg` angeben

Geben Sie das Ergebnis des folgenden Aufrufs an:
`>java de.fhnon.telefon.TelefonBuchProg`

A.22.2 Programmieren von `TelefonLookupProg`

Die Applikation `TelefonLookupProg` erfüllt folgende Anforderungen:

1. `TelefonLookupProg` nutzt das permanente Telefonbuch von `TelefonBuchProg`
2. `TelefonLookupProg` nutzt die Klassen `TelefonEintrag` und `TelefonBuch`
3. `TelefonLookupProg` sucht für einen vorgegebenen Kurznamen die Telefonbucheintragung und gibt den Wert von `kurzname` und von `telefon` aus.
4. Der vorgegebene Kurzname (Wert von `kurzname`) wird beim Aufruf als Argument genannt, zum Beispiel:

```
>java de.fhnon.telefon.TelefonLookupProg Emma
```

5. Findet `TelefonLookupProg` keine Eintragung, dann gibt es keine Ausgabe und die Applikation wird beendet.

Notieren Sie einen Quellcode für diese Applikation `TelefonLookupProg`.

A.23 Fachsprache verstehen

In einer Diskussionsrunde zwischen den Verantwortlichen für die Softwareentwicklung werden die folgenden Aussagen festgestellt:

1. Aller Programmcode befindet sich in einer Datei.
2. Alle Klassen und Interfaces gehören zum Paket `de.fhnon.foo`
3. `K1` ist eine Java-Applikation
4. Klasse `K1` implementiert das Interface `I0`
5. `I0` umfaßt die Methoden `m1()` und `m2()`
6. `K1` hat die Instanzvariablen `v1`, `v2`, `v3` vom Typ `K4`
7. Klasse `K2` enthält eine Instanz `s` der Klasse `K1`
8. Klasse `K3` ist Subklasse von `K2`
9. `K3` hat die Klassenvariable `c1` vom Typ `K4`
10. Klasse `K4` hat die Methode `m3()`

A.23.1 Objektbeziehungen in Java^{TM} abbilden

Bilden Sie die obigen Aussagen in Java-Quellcode ab.

A.23.2 *Getter*- und *Setter*-Methoden ergänzen

Herr Franz Otto ist Anhänger des Java-Beans-Modell. Er möchte unbedingt die Zugriffsmethoden mit dargestellt sehen. Ergänzen Sie daher Ihren Java-Quellcode um die sogenannten *Getter*- und *Setter*-Methoden.

A.24 Paket mit Klassen- & Interface-Dateien notieren

Nach einer eingehenden Systemanalyse ergeben sich folgenden Aussagen:

1. Alle Klassen und Interfaces gehören zum Paket `de.fhnon.bar`.
2. Das Interface `I0` umfaßt die allgemein zugreifbaren Methoden `m1()` und `m2()`. Beide Methoden haben keinen Rückgabewert.
3. Das Interface `I1` hat die allgemein zugreifbare Methode `m3()`. Die Methode hat keinen Rückgabewert.
4. Die Klasse `K1` implementiert das Interface `I0` und das Interface `I1`.
5. `K1` ist eine Java-Applikation.
6. `K1` hat die privaten Instanzvariablen `v1` und `v2` vom Typ `String`.
7. `K1` hat die private Instanzvariable `v3` vom Typ `K2`.
8. Die Klasse `K2` enthält eine Instanz vom Typ `K4`. Die zugehörige Referenz `v` ist privat, nicht allgemein zugreifbar.
9. `K2` hat die allgemein zugreifbare Klassenvariable `c2` vom Typ `Vector`
10. Die Klasse `K3` ist eine abstrakte Klasse.
11. `K3` hat die allgemein zugreifbare Klassenkonstante `c3` vom Typ `int` mit dem festen Wert 100.
12. Die Klasse `K4` ist Subklasse von `K3`
13. `K4` hat die geschützte, bedingt zugreifbare Methode `m4()` mit dem Parameter `a`. Der Parameter ist vom Typ `String`. Die Methode gibt nur den Wert von `a` zurück.

A.24.1 Aussagen als Klassendiagramm in UML-Notation abbilden

Bilden Sie die obigen Aussagen als ein Klassendaigramm in UML-Notation ab.

A.24.2 Aussagen in Java-Quellcode abbilden

Bilden Sie die obigen Aussagen in Java-Quellcode ab.

A.24.3 Aufruf der Datei `K1.java`

Nehmen Sie an, Ihr obiger Java-Quellcode ist in der Datei `K1.java` gespeichert. Skizzieren Sie kurz die Wirkung der `package`-Angabe für den Aufruf zum Compilieren (`javac` ...) und zum Ausführen (`java` ...).

A.25 `HTML`-Dokument mit CSS

Das folgende HTML-Dokument `myPage.html` nutzt die CSS-Datei `myPageStyle.css`. Außerdem weist es eine Layout-Spezifikation im `<style>`-Konstrukt auf.

XHTML-Datei `myPage.html`

```
 1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 2    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 3  <!-- CSS-Beispiel                                      -->
 4  <!-- Bonin  1-Jun-1998                                 -->
 5  <!-- Update ... 25-Dec-2002                           -->
 6  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
 7  <head>
 8  <title>Cascading Style Sheet</title>
 9  <link href="myPageStyle.css" rel="stylesheet"
10    type="text/css" />
11  <style>
12  h1 {
13    color:      white;
14    background: black;
15  }
16  </style>
17  </head>
```

```
18  <body>
19  <h1><em>CSS</em> (Cascading Style Sheet)</h1>
20  <ul>
21  <li>Frage:
22    <p>Wer mag denn nur <em>CSS?</em></p></li>
23  <li>Antwort:
24    <p>Jeder der HTML-Dokumente schreibt!</p></li>
25  </ul>
26  </body>
27  </html>
28
```

CSS-Datei `myPageStyle.css`

```
 1  /* Cascading Style Sheet: myStyle.css  */
 2  /* Bonin 30-Jun-1998 ... 24-Dec-2002   */
 3  p {
 4    font-size:  12pt;
 5    color:      red;
 6    background: white;
 7  }
 8  h1 em {
 9    font-size:  28pt;
10  }
11  h1 {
12    font-size:  14pt;
13    color:      white;
14    background: blue;
15  }
16  em {
17    color:      green;
18    background: white;
19    font-style: italic;
20  }
21
```

[Hinweis: Die Zeilennummern sind nicht Bestandteil der HTML-Datei und auch nicht der CSS-Datei.]

A.25.1 Header-Konstrukt interpretieren

Beschreiben Sie die Hauptüberschrift (`<h1>`-Konstrukt), wenn diese von einem Web-Browser angezeigt wird.

A.25.2 Hervorhebungsspezifikation

Erläutern Sie, warum in der CSS-Datei einerseits `h1 em {...}` und andererseits `em {...}` angegeben sind.

A.26 CSS-Datei und <style>-Konstrukt

Das folgende HTML-Dokument `myFINAL.html` nutzt die CSS-Datei `myFStyle.css`. Außerdem weist es eine Layout-Spezifikation im `<style>`-Konstrukt auf.

HTML-Datei `myFINAL.html`

```
 1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 2    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
 4  <!--  Bonin: 21-Jan-1995 ... 24-Dec-2002             -->
 5  <!--  Update:                                        -->
 6  <head>
 7  <title>FINAL, 5(8), 1998</title>
 8  <link href="myFStyle.css"
 9    rel="stylesheet" type="text/css" />
10  <style type="text/css">
11  h1 {
12    color:      black;
13    background: yellow;
14  }
15  </style>
16  </head>
17  <body>
18  <h1>FINAL, 5(8), 1998</h1>
19  <ul>
20  <li><em>F</em>achhochschule Nordostniedersachsen</li>
21  <li><em>In</em>formatik</li>
22  <li><em>A</em>rbeitsberichte</li>
23  <li><em>L</em>&uuml;neburg</li>
24  </ul>
25  <h>Persistente Objekte ---
26    Der <em>Elchtest</em> f&uuml;r ein Java-Programm</h1>
27  <p>FINAL, 8.Jahrgang, Heft 5, Dezember 1998,
28    ISSN 0939-8821<br />
29    Beziehbar: FHNON FBW, Volgershall 1,
30    D-21339 L&uuml;neburg</p>
```

```
31  </body>
32  </html>
33
```

CSS-Datei `myFStyle.css`

```
/* Cascading Style Sheet: myStyle.css  */
/* Bonin 21-Jan-1999 ... 25-Dec-2002   */
p {
  font-size:  12pt;
  color:      red;
  background: white;
}
h1 em {
  font-size:  28pt;
}
h1 {
  font-size:  14pt;
  color:      white;
  background: blue;
}
em {
  color:      green;
  background: white;
  font-style: italic;
}
```

[Hinweis: Die Zeilennummern sind nicht Bestandteil der HTML-Datei und auch nicht der CSS-Datei.]

A.26.1 Fehler finden und korrigieren

Die Datei `myFINAL.html` enthält einen „Schreibfehler“ (— wenn diese von einem Web-Browser angezeigt werden soll, der die W3C-Empfehlungen in Bezug auf XHTML und CSS erfüllt). Finden Sie diesen Fehler und korrigieren Sie das betreffende Konstrukt.

A.26.2 Cascading Style Sheet auswerten

Notieren Sie das Resultat der „Kaskade“ beim Anzeigen der Datei `myFINAL.html` in folgender Form:

```
p {...} h1 em {...} h1 {...} em {...}
```

A.26.3 Beschreibung einer angezeigten Überschrift

Beschreiben Sie, wie die Überschrift „Persistente Objekte — Der Elchtest für ein Java-Programm“ von einem Browser angezeigt wird. Geben Sie den Namen und die Version des Browsers an, den Sie für dieses Anzeigen nutzen würden.

A.27 Standardgerechtes Programmieren in Java

Der Programmierer Hansi Schlaumeier ist sich nicht sicher ob seine Applikation `Hund` ordnungsgemäß programmiert ist oder mehr einer Denksportaufgabe gleicht. Vorsichtshalber läßt er das Programm mit Hilfe eines Programms auf *Reflection*-Basis analysieren. Das Analyseprotokoll zeigt die Datei `Analyse.log`

Protokolldatei `Analyse.log`

```
>java -fullversion
java full version "JDK 1.1.6 IBM build a116-19980529" (JIT: jitc)
>javac Hund.java
>javac Analyse.java
>java Analyse Hund
synchronized class Hund extends java.lang.Object {
  // Feld(er)
  private java.lang.String name;
  public boolean weiblich;
  private Hund mutter;
  private Hund vater;
  // Konstruktor(en)
  public Hund(java.lang.String, boolean);
  // Methode(n)
  public java.lang.String getName();
  public Hund getMutter();
  public Hund setMutter(Hund);
  public Hund getVater();
  public Hund setVater(Hund);
  public static void main(java.lang.String[]);
}
>
```

Klasse `Hund`

```
/**
 *  Beispiel einer Rekursion innerhalb der Klasse: Vater und
 *  Mutter sind wieder vom Typ Hund
 *
 *@author     Hinrich Bonin
 *@version    1.0
 *@since      22-Jan-1999
 */
import java.util.*;

class Hund
{
   private String name = "";
   public boolean weiblich = true;
   private Hund mutter;
   private Hund vater;


   public Hund(String name, boolean weiblich)
   {
      this.name = name;
      this.weiblich = weiblich;
      System.out.println(name + " lebt!");
   }


   public String getName()
   {
      return name;
   }


   public Hund getMutter()
   {
      return mutter;
   }


   public Hund setMutter(Hund mutter)
   {
      this.mutter = mutter;
      return this;
   }


   public Hund getVater()
```

```
   {
      return vater;
   }


   public Hund setVater(Hund vater)
   {
      this.vater = vater;
      return this;
   }


   public static void main(String[] argv)
   {
      System.out.println(
            (new Hund("Bello von der Eulenburg", false))
            .setMutter(new Hund("Berta vom Lechgraben", true))
            .setVater(new Hund("Alex vom Hirschgarten", false))
            .getMutter().name);
   }
}
```

A.27.1 Beurteilung von `Hund`

Ist die Applikation `Hund` entsprechend dem üblichen Java-Standard programmiert? Wenn nicht, beschreiben Sie kurz die Abweichungen.

A.27.2 Ergebnis von `Hund`

Geben Sie das Ergebnis von `java Hund` an.

A.27.3 Reengineering von `Hund`

Ändern Sie den Quellcode von `Hund.java` im Sinne des üblichen Java-Standards ohne das Ergebnis von `java Hund` zu verändern.

A.28 *Side Effect* bei Objekt-Orientierung

Stellen Sie fest, an welchem Punkt gegen das Paradigma der Objekt-Orientierung „verstoßen“? Nennen Sie einen Korrekturvorschlag.

Protokolldatei `CProg.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/sideeffect/*.java

C:\bonin\anwd\code>java de.fhnon.sideeffect.CProg
Object o1 slot value: C2
Object o2 slot value: C2

C:\bonin\anwd\code>
```

Klasse `C1`

```
 1  /**
 2   *  Example "Side Effect"
 3   *
 4   *@since      1-Feb-2002
 5   *@author     Hinrich Bonin
 6   *@version    1.0
 7   */
 8
 9  package de.fhnon.sideeffect;
10
11  public class C1
12  {
13     private String slot = "C1";
14
15
16     public String getSlot()
17     {
18        return slot;
19     }
20
21
22     public void setSlot(String slot)
23     {
24        this.slot = slot;
25     }
26
27
```

```
28     public void m1(C2 o)
29     {
30        this.slot = o.getSlot();
31
32     }
33
34
35     public void m2(C2 o)
36     {
37        o.setSlot(this.getSlot());
38
39     }
40
41  }
42
43
```

Klasse C2

```
 1  /**
 2   *  Example "Side Effect"
 3   *
 4   *@since      1-Feb-2002
 5   *@author     Hinrich Bonin
 6   *@version    1.0
 7   */
 8
 9  package de.fhnon.sideeffect;
10
11  public class C2
12  {
13     private String slot = "C2";
14
15
16     public String getSlot()
17     {
18        return slot;
19     }
20
21
22     public void setSlot(String slot)
23     {
24        this.slot = slot;
25     }
26
27
28     public void m1(C1 o)
```

```
29     {
30        this.slot = o.getSlot();
31
32     }
33
34
35     public void m2(C1 o)
36     {
37        o.setSlot(this.getSlot());
38
39     }
40
41  }
42
43
```

Klasse `CProg`

```
 1  /**
 2   *  Example "Side Effect"
 3   *
 4   *@since      1-Feb-2002
 5   *@author     Hinrich Bonin
 6   *@version    1.0
 7   */
 8
 9  package de.fhnon.sideeffect;
10
11  public class CProg
12  {
13
14     public static void main(String[] args)
15     {
16        C1 o1 = new C1();
17        C2 o2 = new C2();
18        o1.m1(o2);
19        o1.setSlot("OK?");
20        o2.m2(o1);
21
22        System.out.println(
23              "Object o1 slot value: " +
24              o1.getSlot());
25        System.out.println(
26              "Object o2 slot value: " +
27              o2.getSlot());
28     }
29  }
```

```
30
31
```

A.29 XML-Daten bearbeiten

In dem Traditionsunternehmen *Lagerhaus Moritz GmbH, Hamburg* sind Daten der Geschäftspartner noch auf klassische Art gespeichert. Im Rahmen der IT-Modernisierung sind diese Daten auf XML-Format umzustellen.

Für das klassische Datenformat findet man in der alten Dokumentation folgende Beschreibung:

1. Die Datei enthält Firmen. Jede Firma beginnt mit einer Zeile, die nur die Eintragung `!F` enthält.
2. In der nächsten Zeile folgt der eindeutige Namen der Firma.
3. In der nächsten Zeile steht die Rechtsform.
4. In der nächsten Zeile steht der Gerichtsstand.
5. Eine Firma kann mehrere Adressen und auch mehrere Kontakte haben. Der Beginn einer Adresse wird durch eine Zeile mit der alleinigen Eintragung `!A` und der eines Kontaktes mit der alleinigen Eintragung `!K` gekennzeichnet.
6. Eine Adresse besteht immer aus drei Zeilen, erst die Postleitzahl, dann die Ortsangabe und dann die Straße (incl. Hausnummer).
7. Ein Kontakt besteht immer aus drei Zeilen, erst die Telefonnummer, dann die Faxnummer und dann die Email-Adresse.

Die Datei `Partner.txt` zeigt einen beispielhaften Auszug aus diesem Datenbestand.

Datei `Partner.txt`

```
1  !F
2  Otto
3  GmbH
4  Berlin
5  !A
6  D-21391
```

```
 7  Reppenstedt
 8  Eulenburg 6
 9  !A
10  D-21339
11  Lüneburg
12  Volgershall 1
13  !K
14  04131-677175
15  04131-677140
16  info@otto-lueneburg.com
17  !F
18  Meyer
19  AG
20  Hamburg
21  !A
22  D-21000
23  Hamburg
24  Alsterweg 18
25  !A
26  D-21000
27  Hamburg
28  Vogelsburg 2
29  !K
30  040-11111
31  040-11112
32  meyer@marktplatz-hamburg.de
33
```

A.29.1 DTD aufstellen

Stellen Sie in einer Datei `Partner.dtd` eine wohl überlegte Document Type Definition auf damit die Daten nach Überführung in eine XML-Datei mittels XML-Parser validierbar sind. Bitte skizzieren Sie Ihre Überlegungen.

A.29.2 XML-Datei erzeugen

Erstellen Sie ein Programm, das die Daten in das von Ihnen definierte XML-Format konvertiert. Nennen Sie die Ergebnisdatei `Partner.xml` und Ihre Programmdatei(en) `Partnern.java`. Dabei ist `n` eine laufende Nummer beginnend mit Null.

A.29.3 XML-Datei visualisieren

Erstellen Sie ein Programm, das Ihre XML-Daten in Form einer Tabelle anzeigt.

Anhang B

Lösungen zu den Übungen

Lösung Aufgabe A.1 S. 399:
A.1.1:
Die Abbildung B.1 S. 488 zeigt das Klassendiagramm für die RVE.

Eine Abbilung dieser Klassen und ihrer Verknüpfungen ist im folgenden angegeben. Um die Hierarchie der Konstruktoren zu zeigen wurde eine zusätzliche Klassenvariable `anzahl` eingeführt.

Klasse `Fahrrad`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>Fahrrad:
 *  abstrakte Klasse für Einrad und Zweirad</h2>
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;

public abstract class Fahrrad
{
   /*
    *  rahmenNummer identifiziert ein Fahrrad
    */
   private String rahmenNummer;
   /*
```

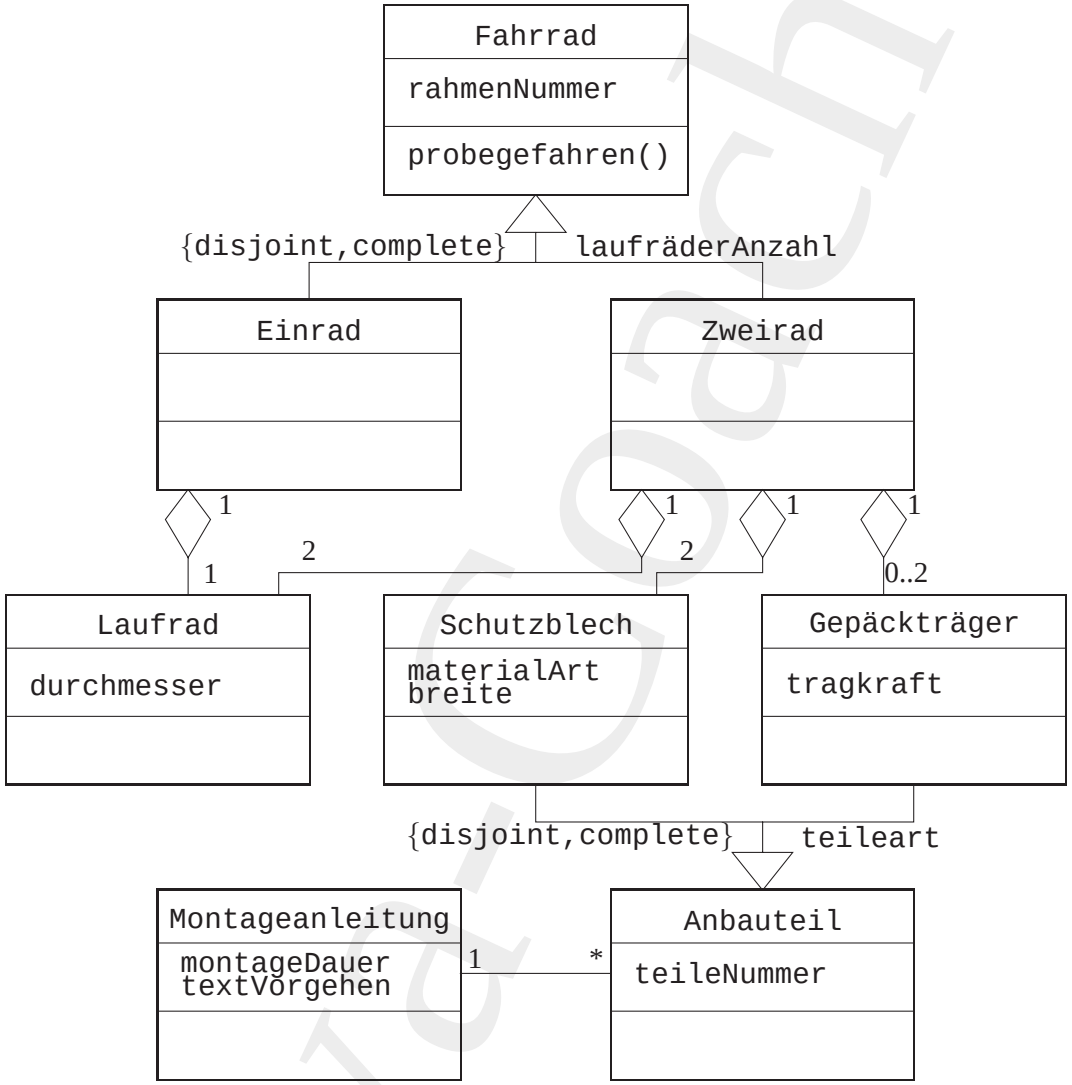


Abbildung B.1: Aufgabe A.1.1 S. 400: Klassendiagramm für die Montagesicht

```
     *  Setzt probegefahren()
     */
    private boolean probegefahren = false;


    public String getRahmenNummer()
    {
       return rahmenNummer;
    }


    public void setRahmenNummer(String rahmenNummer)
    {
       this.rahmenNummer = rahmenNummer;
    }


    public boolean getProbegefahren()
    {
       return probegefahren;
    }


    /*
     *  pobegefahren() applizieren nach der Probefahrt
     */
    public void probegefahren()
    {
       probegefahren = true;
    }
}
```

Klasse `Einrad`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>Einrad: 1
 *  Laufrad</h2>
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;

public class Einrad extends Fahrrad
```

```
{
   private Laufrad laufrad;


   Einrad(String rahmenNummer,
         Laufrad laufrad)
   {
      this.setRahmenNummer(rahmenNummer);
      this.laufrad = laufrad;
   }
}
```

Klasse `Zweirad`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>Zweirad ist
 *  Fahrrad mit 2 Laufrädern, gegebenenfalls mit 2
 *  Schutzblechen und bis zu 2 Gepäckträgern</h2>
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;

public class Zweirad extends Fahrrad
{
   /**
    *  Laufräder, Schutzbleche und Gepäckträger als Arrays
    *  abgebildet
    */
   private Laufrad laufrad[] =
         new Laufrad[2];
   private Schutzblech schutzblech[] =
         new Schutzblech[2];
   private Gepaecktraeger gepaecktraeger[] =
         new Gepaecktraeger[2];

   public static int anzahl = 0;


   public Laufrad getVorderrad()
   {
      return laufrad[0];
   }
```

```
public Laufrad getHinterrad()
{
   return laufrad[1];
}


public Schutzblech getVorderradSchutzblech()
{
   return schutzblech[0];
}


public Schutzblech getHinterradSchutzblech()
{
   return schutzblech[1];
}


public Gepaecktraeger getVorderradGepaecktraeger()
{
   return gepaecktraeger[0];
}


public Gepaecktraeger getHinterradGepaecktraeger()
{
   return gepaecktraeger[1];
}


public void setVorderradGepaecktraeger(
      Gepaecktraeger gepaecktraeger)
{
   this.gepaecktraeger[0] = gepaecktraeger;
}


public void setHinterradGepaecktraeger(
      Gepaecktraeger gepaecktraeger)
{
   this.gepaecktraeger[1] = gepaecktraeger;
}
```

```
   public Zweirad()
   {
      anzahl++;
   }


   /**
    *  Zweirad mit 2 Laufrädern ohne Anbauteile
    *
    *@param  rahmenNummer  ist der Zweirrad-Identifier
    *@param  vorderrad     ist ein Laufrad
    *@param  hinterrad     ist ein Laufrad
    */
   public Zweirad(String rahmenNummer,
         Laufrad vorderrad,
         Laufrad hinterrad)
   {
      this();
      this.setRahmenNummer(rahmenNummer);
      laufrad[0] = vorderrad;
      laufrad[1] = hinterrad;
   }


   /**
    *  Zweirad mit 2 Laufrädern und Anbauteilen
    *
    *@param  rahmenNummer         ist der
    *      Zweirrad-Identifier
    *@param  vorderrad            ist ein Laufrad
    *@param  hinterrad            ist ein Laufrad
    *@param  vorderradSchutzblech  ist ein Schutzblech
    *@param  hinterradSchutzblech  ist ein Schutzblech
    */
   public Zweirad(String rahmenNummer,
         Laufrad vorderrad,
         Laufrad hinterrad,
         Schutzblech vorderradSchutzblech,
         Schutzblech hinterradSchutzblech)
   {
      // Konstruktor mit den Laufrädern
      this(rahmenNummer, vorderrad, hinterrad);
      schutzblech[0] = vorderradSchutzblech;
      schutzblech[1] = hinterradSchutzblech;
   }
}
```

Klasse `Laufrad`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>Laufrad hat
 *  ID und Durchmesser </h2>
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;

public class Laufrad
{
   /*
    *  laufradID identifiziert ein Laufrad
    */
   private String laufradID;
   private int durchmesser;


   public int getDurchmesser()
   {
      return durchmesser;
   }


   Laufrad(String laufradID,
         int durchmesser)
   {
      this.laufradID = laufradID;
      this.durchmesser = durchmesser;
   }
}
```

Klasse `Schutzblech`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>Schutzblech
 *  ist Anbauteil und hat MaterialArt und Breite</h2>
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
```

```
 *@version    1.0
 */
package de.fhnon.rvegmbh;

public class Schutzblech extends Anbauteil
{
   private String materialArt;
   private String breite;


   public String getMaterialArt()
   {
      return materialArt;
   }


   public String getBreite()
   {
      return breite;
   }


   /**
    *@param  teileNummer       hat jedes Anbauteil
    *@param  materialArt       Description of the Parameter
    *@param  breite            Description of the Parameter
    *@param  montageanleitung  Description of the Parameter
    */
   Schutzblech(String teileNummer,
         String materialArt,
         String breite,
         Montageanleitung montageanleitung)
   {
      this.setTeileNummer(teileNummer);
      this.setMontageanleitung(montageanleitung);
      this.materialArt = materialArt;
      this.breite = breite;
   }
}
```

Klasse `Gepaecktraeger`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>
 *  Gepaecktraeger ist Anbauteil und hat Tragkraft-Slot</h2>
```

```
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;

public class Gepaecktraeger extends Anbauteil
{
   private int tragkraft;


   public int getTragkraft()
   {
      return tragkraft;
   }


   /**
    *@param  teileNummer       hat jedes Anbauteil
    *@param  tragkraft         Description of the Parameter
    *@param  montageanleitung  Description of the Parameter
    */
   Gepaecktraeger(String teileNummer,
         int tragkraft,
         Montageanleitung montageanleitung)
   {
      this.setTeileNummer(teileNummer);
      this.tragkraft = tragkraft;
      this.setMontageanleitung(montageanleitung);
   }
}
```

Klasse `Anbauteil`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>Anbauteil
 *  hat Teilenummer und Assoziation zu Montageanleitung</h2>
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;
```

```
public abstract class Anbauteil
{
   /*
    *  teileNummer identifiziert ein Anbauteil
    */
   private String teileNummer;
   private Montageanleitung montageanleitung;


   public String getTeileNummer()
   {
      return teileNummer;
   }


   public void setTeileNummer(String teilenummer)
   {
      this.teileNummer = teilenummer;
   }


   public Montageanleitung getMontageanleitung()
   {
      return montageanleitung;
   }


   public void setMontageanleitung(
         Montageanleitung montageanleitung)
   {
      this.montageanleitung = montageanleitung;
   }
}
```

Klasse `Montageanleitung`

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>
 *  Montageanleitung hat MontageDauer und TextVorgehen</h2>
 *
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;
```

```
public class Montageanleitung
{
   /*
    *  Achtung kein Identifier für Montageanleitung
    */
   private int montageDauer;
   private String textVorgehen;


   public int getMontageDauer()
   {
      return montageDauer;
   }


   public String getTextVorgehen()
   {
      return textVorgehen;
   }


   public Montageanleitung(int montageDauer,
         String textVorgehen)
   {
      this.montageDauer = montageDauer;
      this.textVorgehen = textVorgehen;
   }
}
```

Klasse RVEGmbH

```
/**
 *  <h1>RadVertriebsExperten GmbH (RVE)</h1> <h2>RVEGmbH
 *  stellt Beispiele bereit.</h2>
 *
 *@since      30-Mar-2001
 *@author     Bonin, Hinrich E.G.
 *@version    1.0
 */
package de.fhnon.rvegmbh;

public class RVEGmbH
{
   /**
    *  Beispiele für Einrad und Zweirad
```

```
     *
     *@param  args  The command line arguments
     */
    public static void main(String[] args)
    {
       Laufrad laufradI = new Laufrad("I", 26);
       Einrad einradA = new Einrad("A", laufradI);
       einradA.probegefahren();

       Laufrad laufradII = new Laufrad("II", 28);
       Einrad einradB = new Einrad("B", laufradII);

       Laufrad laufradIII = new Laufrad("III", 26);
       Laufrad laufradIV = new Laufrad("IV", 26);
       Zweirad zweirad1 = new Zweirad(
             "1", laufradIII, laufradIV);

       Laufrad laufradV = new Laufrad("V", 28);
       Laufrad laufradVI = new Laufrad("VI", 28);
       Montageanleitung anweisungAllgemein =
             new Montageanleitung(
             120, "Schraube von unten ...");
       Schutzblech schutzblech1 =
             new Schutzblech(
             "S1", "Kunststoff", "4 cm",
             anweisungAllgemein);
       Schutzblech schutzblech2 =
             new Schutzblech(
             "S2", "Blech", "5 cm",
             anweisungAllgemein);
       Gepaecktraeger gepaecktraegerX =
             new Gepaecktraeger(
             "GX", 100, anweisungAllgemein);

       Zweirad zweirad2 = new Zweirad(
             "2", laufradV, laufradVI,
             schutzblech1, schutzblech2);
       zweirad2.setHinterradGepaecktraeger(
             gepaecktraegerX);

       System.out.println("RVEGmbH:"
              + "\nEinrad: " +
             einradA.getRahmenNummer()
              + " probegefahren: " +
             einradA.getProbegefahren()
              + "\nEinrad: " +
```

```
                einradB.getRahmenNummer()
                 + " probegefahren: " +
                einradB.getProbegefahren()
                 + "\nZweirad: " +
                zweirad1.getRahmenNummer()
                 + " Vorderraddurchmesser: "
                 + zweirad1.getVorderrad().getDurchmesser()
                 + "\nZweirad: " +
                zweirad2.getRahmenNummer()
                 + " Schutzblechmontage: " +
                zweirad2.getHinterradSchutzblech()
                .getMontageanleitung().getTextVorgehen()
                 + "\nSumme Zweirad = " + Zweirad.anzahl + "."
                );
    }
}
```

Protokolldatei `RVEGmbH.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/rvegmbh/*.java

C:\bonin\anwd\code>java de.fhnon.rvegmbh.RVEGmbH
RVEGmbH:
Einrad: A probegefahren: true
Einrad: B probegefahren: false
Zweirad: 1 Vorderraddurchmesser: 26
Zweirad: 2 Schutzblechmontage: Schraube von unten ...
Summe Zweirad = 2.

C:\bonin\anwd\code>
```

A.1.2:
Die Abbildung B.2 S. 500 zeigt die Diagrammerweiterung um den Montageplatz.

Lösung Aufgabe A.2 S. 400:
A.2.1:
Die Abbildung B.3 S. 501 zeigt den Hauptteil des Klassendiagrammes

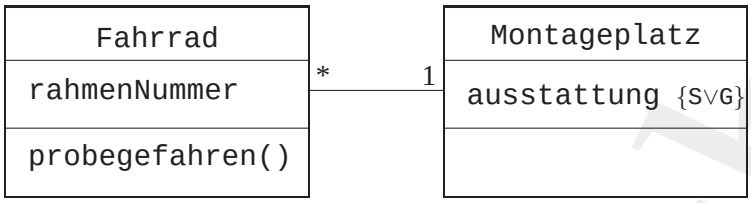


Abbildung B.2: Aufgabe A.1.2 S. 400: Diagrammerweiterung um den Montageplatz

für die SVI. In Abbildung B.5 S. 525 ist der Aspekt „Zielfernrohr“ abgebildet.

javadoc Eine Abbilung dieser Klassen und ihrer Verknüpfungen ist im folgenden angegeben. Dabei sind auch die Restriktionen abgebildet. Zusätzlich wurden Erläuterungen im Quellecode mit HTML-Konstrukten aufbereitet. Das *Application programming interface* (API) wurde mit `javadoc` erzeugt (↪ Abbildung B.4 S. 524).

Klasse `SVIProdukt`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>SVI-Produkt: abstracte Klasse für Waffe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class SVIProdukt
{
}
```

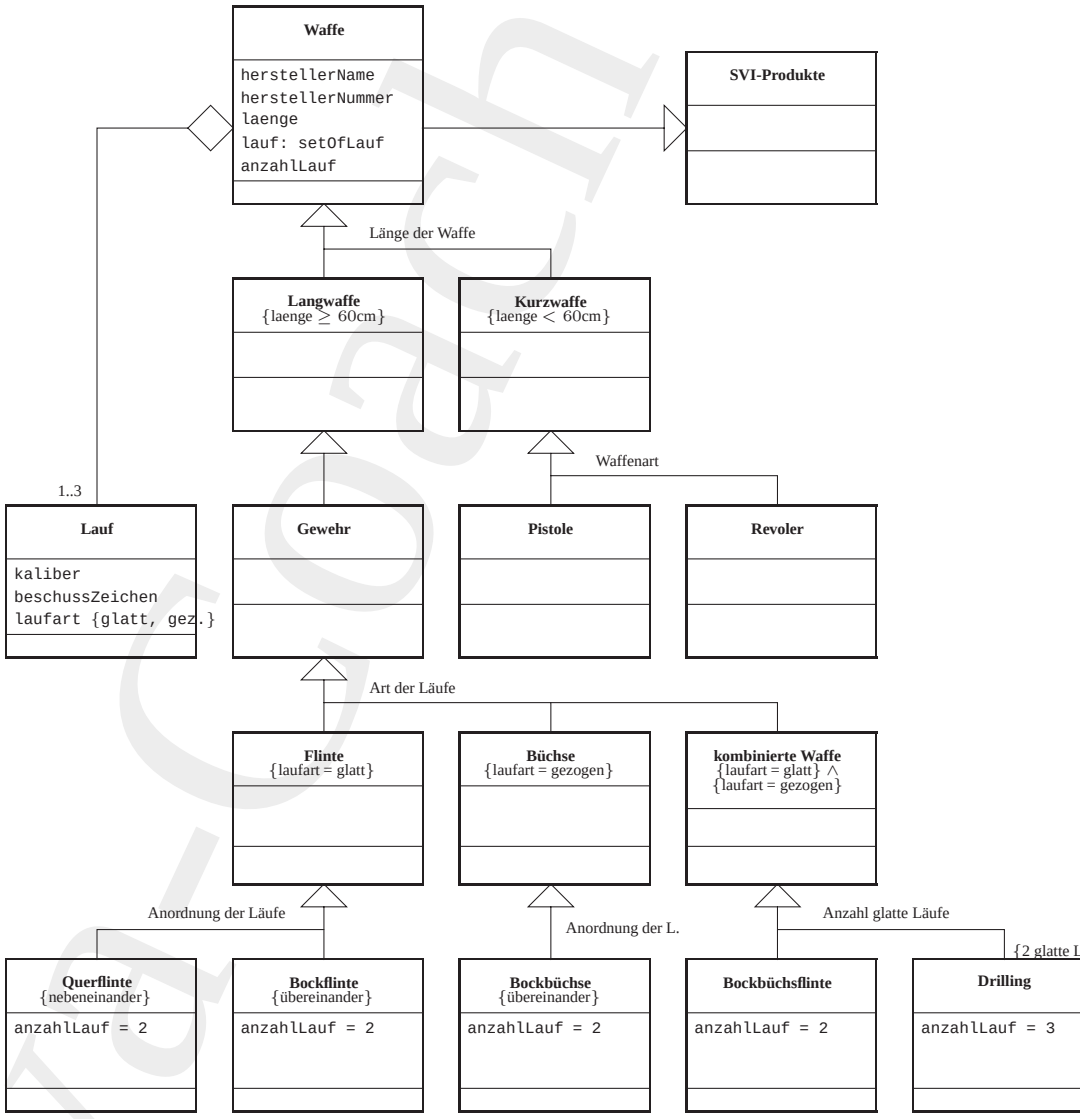


Abbildung B.3: Aufgabe A.2.1 S. 402: SVI-Klassendiagramm „Hauptteil“

Klasse `Waffe`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Waffe hat Herstellerangaben und bis zu 3 Laeufen
 *  </h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class Waffe extends SVIProdukt
{
   private String herstellerName;
   private String herstellerNummer;
   private int laenge;
   private Lauf lauf[] = new Lauf[3];
   private int anzahlLauf;


   public String getHerstellerName()
   {
      return herstellerName;
   }


   public void setHerstellerName(
         String herstellerName)
   {
      this.herstellerName = herstellerName;
   }


   public String getHerstellerNummer()
   {
      return herstellerNummer;
   }


   public void setHerstellerNummer(
         String herstellerNummer)
   {
      this.herstellerNummer = herstellerNummer;
```

```
    }

    public int getLaenge()
    {
       return laenge;
    }

    public void setLaenge(int laenge)
    {
       this.laenge = laenge;
    }

    public Lauf getLauf(int key)
    {
       return lauf[key];
    }

    public void setLauf(int key, Lauf lauf)
    {
       this.lauf[key] = lauf;
    }

    public int getAnzahlLauf()
    {
       return anzahlLauf;
    }

    public void setAnzahlLauf(int anzahlLauf)
    {
       this.anzahlLauf = anzahlLauf;
    }
}
```

Klasse `Lauf`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Lauf hat Kaliber, Beschusszeichen und Laufart</h2>
```

```
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class Lauf
{
   private String kaliber;
   private String beschussZeichen;
   private String laufart;


   public String getKaliber()
   {
      return kaliber;
   }


   public void setKaliber(String kaliber)
   {
      this.kaliber = kaliber;
   }


   public String getBeschussZeichen()
   {
      return beschussZeichen;
   }


   public void setBeschussZeichen(String beschussZeichen)
   {
      this.beschussZeichen = beschussZeichen;
   }


   public String getLaufart()
   {
      return laufart;
   }
```

```
   public void setLaufart(String laufart)
   {
      this.laufart = laufart;
   }


   public Lauf(
         String kaliber,
         String beschussZeichen,
         String laufart)
   {
      this.kaliber = kaliber;
      this.beschussZeichen = beschussZeichen;
      if (!((laufart == "glatt") ||
            (laufart == "gezogen")))
      {
         laufart = "Nicht OK!";
      }
      this.laufart = laufart;
   }
}
```

Klasse `Langwaffe`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Langwaffe ist laenger als 60 cm </h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class Langwaffe extends Waffe
{
   public static int laengeMinimum()
   {
      return 60;
   }
}
```

Klasse `Kurzwaffe`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Kurzwaffe ist unter 60 cm lang und hat nur einen
 *  Lauf</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class Kurzwaffe extends Waffe
{
   /*
    *  Kurzwaffe hat nur einen Lauf, daher ueberschreiben der
    *  Methoden aus Waffe
    */
   public Lauf getLauf()
   {
      return this.getLauf(0);
   }


   public void setLauf(Lauf lauf)
   {
      this.setLauf(0, lauf);
   }


   public static int laengeMaximum()
   {
      return 60;
   }
}
```

Klasse `Gewehr`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Gewehr ist eine Langwaffe</h2>
 *
```

```
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class Gewehr extends Langwaffe
{
}
```

Klasse `Pistole`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Pistole ist eine Kurzwaffe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class Pistole extends Kurzwaffe
{
   public static String waffenart()
   {
      return
            "Lauf- und Patronenlager bilden eine Einheit. "
             + "\nLadevorgang erfolgt durch Schlitten.";
   }


   Pistole(Lauf lauf)
   {
      this.setLauf(0, lauf);
   }


   public Pistole(
         String herstellerName,
         String herstellerNummer,
```

```
         int laenge, Lauf lauf)
   {
      this(lauf);
      this.setHerstellerName(herstellerName);
      this.setHerstellerNummer(herstellerNummer);
      if (laenge > Kurzwaffe.laengeMaximum())
      {
         this.setLaenge(Kurzwaffe.laengeMaximum());
      } else
      {
         this.setLaenge(laenge);
      }

   }
}
```

Klasse `Revolver`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Revolver ist eine Kurzwaffe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class Revolver extends Kurzwaffe
{
   public static String waffenart()
   {
      return
            "Lauf- und Patronenlager sind getrennt."
             + "\n Ladevorgang erfolgt"
             + " durch Drehung der Trommel.";
   }


   Revolver(Lauf lauf)
   {
      this.setLauf(0, lauf);
   }
```

```
   public Revolver(
         String herstellerName, String herstellerNummer,
         int laenge, Lauf lauf)
   {
      this(lauf);
      this.setHerstellerName(herstellerName);
      this.setHerstellerNummer(herstellerNummer);
      if (laenge > Kurzwaffe.laengeMaximum())
      {
         this.setLaenge(Kurzwaffe.laengeMaximum());
      } else
      {
         this.setLaenge(laenge);
      }

   }
}
```

Klasse `Flinte`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Flinte hat Anordnung glatte Lauefe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class Flinte extends Gewehr
{
   public static String laufart()
   {
      return "glatt";
   }
}
```

Klasse `Buechse`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Buechse hat gezogene Laeufe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class Buechse extends Gewehr
{
   public static String laufart()
   {
      return "gezogen";
   }
}
```

Klasse `KombinierteWaffe`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>KombinierteWaffe hat unterschiedliche Laufarten</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public abstract class KombinierteWaffe extends Gewehr
{
   public static String laufartGlatt()
   {
      return "glatt";
   }


   public static String laufartGezogen()
   {
      return "gezogen";
```

```
   }
}
```

Klasse `Querflinet`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Querflinte hat nebeneinanderliegende Laeufe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class Querflinte extends Flinte
{
   public static String anordnungLaeufe()
   {
      return "nebeneinander";
   }


   Querflinte(Lauf laufLinks, Lauf laufRechts)
   {
      if (laufLinks.getLaufart() != Flinte.laufart())
      {
         laufLinks.setLaufart(
               "Falscher Flintenlauftyp links!");
      }
      this.setLauf(0, laufLinks);

      if (laufRechts.getLaufart() != Flinte.laufart())
      {
         laufRechts.setLaufart(
              "Falscher Flintenlauftyp rechts!");
      }
      this.setLauf(1, laufRechts);

      this.setAnzahlLauf(2);
   }
```

```
   public Querflinte(
         String herstellerName,
         String herstellerNummer,
         int laenge, Lauf laufLinks,
         Lauf laufRechts)
   {
      this(laufLinks, laufRechts);
      this.setHerstellerName(herstellerName);
      this.setHerstellerNummer(herstellerNummer);
      if (laenge < Langwaffe.laengeMinimum())
      {
         this.setLaenge(Langwaffe.laengeMinimum());
      } else
      {
         this.setLaenge(laenge);
      }

   }
}
```

Klasse `Bockflinte`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Bockflinte hat Anordnung der Lauefe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class Bockflinte extends Flinte
{
   public static String anordnungLaeufe()
   {
      return "uebereinander";
   }


   Bockflinte(Lauf laufOben, Lauf laufUnten)
   {
      if (laufOben.getLaufart() != Flinte.laufart())
```

```
        {
           laufOben.setLaufart(
                 "Falscher Flintenlauftyp oben!");
        }
        this.setLauf(0, laufOben);

        if (laufUnten.getLaufart() != Flinte.laufart())
        {
           laufUnten.setLaufart(
                 "Falscher Flintenlauftyp unten!");
        }
        this.setLauf(1, laufUnten);

        this.setAnzahlLauf(2);
     }


     public Bockflinte(
           String herstellerName,
           String herstellerNummer,
           int laenge,
           Lauf laufOben,
           Lauf laufUnten)
     {
        this(laufOben, laufUnten);
        this.setHerstellerName(herstellerName);
        this.setHerstellerNummer(herstellerNummer);
        if (laenge < Langwaffe.laengeMinimum())
        {
           this.setLaenge(Langwaffe.laengeMinimum());
        } else
        {
           this.setLaenge(laenge);
        }

     }
}
```

Klasse Bockbuechse

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Bockbuechse hat Anordnung der Lauefe</h2>
 *
 *@since      8-Apr-2001
```

```
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class Bockbuechse extends Buechse
{
   public static String anordnungLaeufe()
   {
      return "uebereinander";
   }


   Bockbuechse(Lauf laufOben, Lauf laufUnten)
   {
      if (laufOben.getLaufart() != Buechse.laufart())
      {
         laufOben.setLaufart(
               "Falscher Buechsenlauftyp oben!");
      }
      this.setLauf(0, laufOben);

      if (laufUnten.getLaufart() != Buechse.laufart())
      {
         laufUnten.setLaufart(
               "Falscher Buechsenlauftyp unten!");
      }
      this.setLauf(1, laufUnten);

      this.setAnzahlLauf(2);
   }


   public Bockbuechse(
         String herstellerName,
         String herstellerNummer,
         int laenge,
         Lauf laufOben,
         Lauf laufUnten)
   {
      this(laufOben, laufUnten);
      this.setHerstellerName(herstellerName);
      this.setHerstellerNummer(herstellerNummer);
      if (laenge < Langwaffe.laengeMinimum())
```

```
      {
         this.setLaenge(Langwaffe.laengeMinimum());
      } else
      {
         this.setLaenge(laenge);
      }

   }
}
```

Klasse `Bockbuechsflinte`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Bockbuechse hat Anzahll der glatten Lauefe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class Bockbuechsflinte extends KombinierteWaffe
{
   public static int anzahlGlatteLaeufe()
   {
      return 1;
   }


   Bockbuechsflinte(Lauf laufI, Lauf laufII)
   {
      if (!
            (((laufI.getLaufart() ==
            KombinierteWaffe.laufartGlatt())
             && (laufII.getLaufart() ==
            KombinierteWaffe.laufartGezogen()))
             ||
            ((laufI.getLaufart() ==
            KombinierteWaffe.laufartGezogen())
             && (laufII.getLaufart() ==
            KombinierteWaffe.laufartGlatt()))
            )
```

```
            )
      {
         laufI.setLaufart(
               "Falsche Laufkombination!");
         laufII.setLaufart(
               "Falsche Laufkombination!");
      }
      this.setLauf(0, laufI);
      this.setLauf(1, laufII);

      this.setAnzahlLauf(2);
   }


   public Bockbuechsflinte(
         String herstellerName,
         String herstellerNummer,
         int laenge,
         Lauf laufI,
         Lauf laufII)
   {
      this(laufI, laufII);
      this.setHerstellerName(herstellerName);
      this.setHerstellerNummer(herstellerNummer);
      if (laenge < Langwaffe.laengeMinimum())
      {
         this.setLaenge(Langwaffe.laengeMinimum());
      } else
      {
         this.setLaenge(laenge);
      }

   }
}
```

Klasse `Drilling`

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>Drilling hat zwei glatte Laeufe</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
```

```
 */

package de.fhnon.waffenvertrieb;

public class Drilling extends KombinierteWaffe
{
   public static int anzahlGlatteLaeufe()
   {
      return 2;
   }


   Drilling(Lauf laufI, Lauf laufII, Lauf laufIII)
   {
      int iLaufartGlatt = 0;
      int iLaufartGezogen = 0;

      if (laufI.getLaufart() ==
            KombinierteWaffe.laufartGlatt())
      {
         iLaufartGlatt++;
      } else
      {
         if (laufI.getLaufart() ==
               KombinierteWaffe.laufartGezogen())
         {
            iLaufartGezogen++;
         } else
         {
            laufI.setLaufart("Falsche Laufart!");
         }
      }
      ;
      if (laufII.getLaufart() ==
            KombinierteWaffe.laufartGlatt())
      {
         iLaufartGlatt++;
      } else
      {
         if (laufII.getLaufart() ==
               KombinierteWaffe.laufartGezogen())
         {
            iLaufartGezogen++;
         } else
         {
            laufII.setLaufart("Falsche Laufart!");
```

```
         }
      }
      ;
      if (laufIII.getLaufart() ==
            KombinierteWaffe.laufartGlatt())
      {
         iLaufartGlatt++;
      } else
      {
         if (laufIII.getLaufart() ==
               KombinierteWaffe.laufartGezogen())
         {
            iLaufartGezogen++;
         } else
         {
            laufIII.setLaufart("Falsche Laufart!");
         }
      }
      ;
      if (!((iLaufartGlatt ==
            Drilling.anzahlGlatteLaeufe())
             && (iLaufartGezogen == 1)))
      {
         laufI.setLaufart(
               "Falsche Laufartkombination beim Drilling!");
         laufII.setLaufart(
               "Falsche Laufartkombination beim Drilling!");
         laufIII.setLaufart(
               "Falsche Laufartkombination beim Drilling!");
      }
      this.setLauf(0, laufI);
      this.setLauf(1, laufII);
      this.setLauf(2, laufIII);

      this.setAnzahlLauf(3);
   }


   public Drilling(
         String herstellerName,
         String herstellerNummer,
         int laenge,
         Lauf laufI,
         Lauf laufII,
         Lauf laufIII)
   {
```

```
      this(laufI, laufII, laufIII);
      this.setHerstellerName(herstellerName);
      this.setHerstellerNummer(herstellerNummer);
      if (laenge < Langwaffe.laengeMinimum())
      {
         this.setLaenge(Langwaffe.laengeMinimum());
      } else
      {
         this.setLaenge(laenge);
      }

   }
}
```

Klasse SVIGmbH

```
/**
 *  <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
 *  <h2>SVIGmbH stellt Beispiele bereit.</h2>
 *
 *@since      8-Apr-2001
 *@author     Bonin, Hinrich E.G.
 *@created    26. November 2002
 *@version    1.0
 */

package de.fhnon.waffenvertrieb;

public class SVIGmbH
{
   public static void main(String argv[])
   {
      System.out.println("SVIGmbH: ");

      /*
       *  Beispiel Revolver
       */
      Lauf lauf = new Lauf(".45LC", "Berlin", "gezogen");

      Revolver coltSAA =
            new Revolver("Colt", "23163", 14, lauf);

      System.out.println("\n"
            + "\nRevolver:   " +
            coltSAA.getHerstellerNummer()
```

```
        + "\n Kaliber:   " +
      coltSAA.getLauf().getKaliber()
        + "\n Waffenart: " +
      Revolver.waffenart());

/*
 *  Beispiel Bockflinte
 */
Lauf laufA = new Lauf(
      "12/70", "Braunschweig", Flinte.laufart());
Lauf laufB = new Lauf(
      "12/70", "Braunschweig", "dubios");

Bockflinte browning425 =
      new Bockflinte(
      "Browning", "978545", 57, laufA, laufB);

System.out.println("\n"
       + "\nBockflinte:   " +
      browning425.getHerstellerNummer()
       + "\n Laenge:      " +
      browning425.getLaenge()
       + "\n Laeufe:      " +
      Bockflinte.anordnungLaeufe()
       + "\n  Lauf oben : " +
      browning425.getLauf(0).getLaufart()
       + "\n  Lauf unten: " +
      browning425.getLauf(1).getLaufart());

/*
 *  Beispiel Drilling
 */
Lauf laufI = new Lauf(
      "30-06", "Nuernberg", Buechse.laufart());
Lauf laufII = new Lauf(
      "12/70", "Nuernberg", Flinte.laufart());
Lauf laufIII = new Lauf(
      "12/70", "Nuernberg", Buechse.laufart());

Drilling merkel =
      new Drilling(
      "Merkel", "662345", 63, laufI, laufII, laufIII);
System.out.println("\n"
       + "\nDrilling:    " +
      merkel.getHerstellerNummer()
       + "\n Laenge:     " +
```

```
                merkel.getLaenge()
                 + "\n Laufanzahl: " +
                merkel.getAnzahlLauf()
                 + "\n  LaufI :    " +
                merkel.getLauf(0).getLaufart()
                 + "\n  LaufII :   " +
                merkel.getLauf(1).getLaufart()
                 + "\n  LaufIII:   " +
                merkel.getLauf(2).getLaufart());
    }
}
```

Protokolldatei `SVIGmbH.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/waffenvertrieb/*.java

C:\bonin\anwd\code>java de.fhnon.waffenvertrieb.SVIGmbH
SVIGmbH:


Revolver:   23163
 Kaliber:   .45LC
 Waffenart: Lauf- und Patronenlager sind getrennt.
 Ladevorgang erfolgt durch Drehung der Trommel.


Bockflinte:   978545
 Laenge:      60
 Laeufe:      uebereinander
  Lauf oben : glatt
  Lauf unten: Falscher Flintenlauftyp unten!


Drilling:    662345
 Laenge:     63
 Laufanzahl: 3
  LaufI :    Falsche Laufartkombination beim Drilling!
  LaufII :   Falsche Laufartkombination beim Drilling!
```

```
  LaufIII:   Falsche Laufartkombination beim Drilling!

C:\bonin\anwd\code>javadoc de/fhnon/waffenvertrieb/*.java
Loading source file de/fhnon/waffenvertrieb/Bockbuechse.java...
Loading source file de/fhnon/waffenvertrieb/Bockbuechsflinte.java...
Loading source file de/fhnon/waffenvertrieb/Bockflinte.java...
Loading source file de/fhnon/waffenvertrieb/Buechse.java...
Loading source file de/fhnon/waffenvertrieb/Drilling.java...
Loading source file de/fhnon/waffenvertrieb/Flinte.java...
Loading source file de/fhnon/waffenvertrieb/Gewehr.java...
Loading source file de/fhnon/waffenvertrieb/KombinierteWaffe.java...
Loading source file de/fhnon/waffenvertrieb/Kurzwaffe.java...
Loading source file de/fhnon/waffenvertrieb/Langwaffe.java...
Loading source file de/fhnon/waffenvertrieb/Lauf.java...
Loading source file de/fhnon/waffenvertrieb/Pistole.java...
Loading source file de/fhnon/waffenvertrieb/Querflinte.java...
Loading source file de/fhnon/waffenvertrieb/Revolver.java...
Loading source file de/fhnon/waffenvertrieb/SVIGmbH.java...
Loading source file de/fhnon/waffenvertrieb/SVIProdukt.java...
Loading source file de/fhnon/waffenvertrieb/Waffe.java...
Constructing Javadoc information...
Standard Doclet version 1.4.0

Generating constant-values.html...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating packages.html...
Generating de\fhnon\waffenvertrieb\package-frame.html...
Generating de\fhnon\waffenvertrieb\package-summary.html...
Generating de\fhnon\waffenvertrieb\package-tree.html...
Generating de\fhnon\waffenvertrieb\Bockbuechse.html...
Generating de\fhnon\waffenvertrieb\Bockbuechsflinte.html...
Generating de\fhnon\waffenvertrieb\Bockflinte.html...
Generating de\fhnon\waffenvertrieb\Buechse.html...
Generating de\fhnon\waffenvertrieb\Drilling.html...
Generating de\fhnon\waffenvertrieb\Flinte.html...
Generating de\fhnon\waffenvertrieb\Gewehr.html...
Generating de\fhnon\waffenvertrieb\KombinierteWaffe.html...
Generating de\fhnon\waffenvertrieb\Kurzwaffe.html...
```

```
Generating de\fhnon\waffenvertrieb\Langwaffe.html...
Generating de\fhnon\waffenvertrieb\Lauf.html...
Generating de\fhnon\waffenvertrieb\Pistole.html...
Generating de\fhnon\waffenvertrieb\Querflinte.html...
Generating de\fhnon\waffenvertrieb\Revolver.html...
Generating de\fhnon\waffenvertrieb\SVIGmbH.html...
Generating de\fhnon\waffenvertrieb\SVIProdukt.html...
Generating de\fhnon\waffenvertrieb\Waffe.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...

C:\bonin\anwd\code>
```

A.2.2:
Die Abbildung B.6 S. 525 zeigt die Diagrammergänzung um den Aspekt „Waffenbesitzkarte“.

Lösung Aufgabe A.3 S. 402:
A.3.1:

Klasse `echo`

```
/**
 *  Klassenname mit kleinem Anfangsbuchstaben, also echo
 *  statt Echo, da in Shells echo üblicherweise in kleinen
 *  Buchstaben geschrieben wird.
 *
 *@author     bonin
 *@created    26. November 2002
 */
public class echo
{
   public static void main(String argv[])
   {
      for (int i = 0; i < argv.length; i++)
      {
         System.out.print(argv[i] + " ");
      }
      System.out.print("\n");
   }
}
```

Legende:
Quellecode der Klasse `Flinte` ↪ S. 509

Abbildung B.4: API mit `javadoc` der Klasse `Flinte`

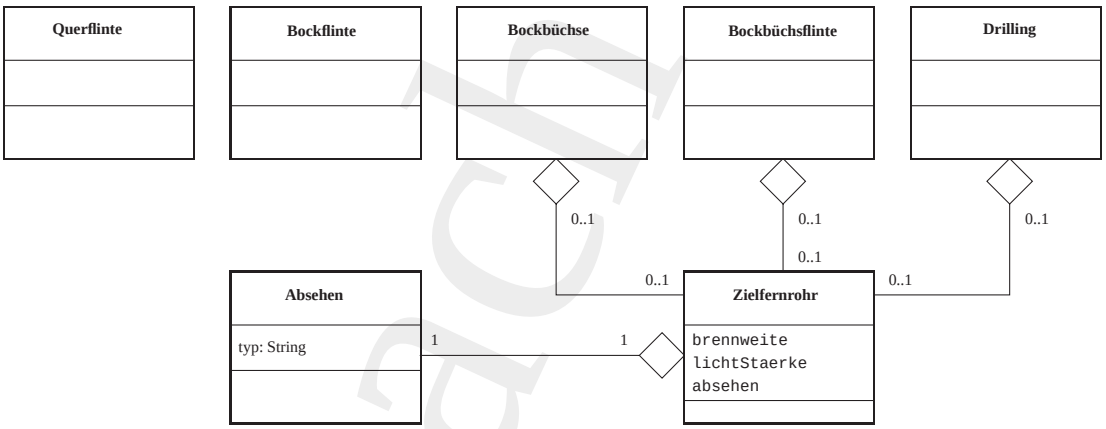


Abbildung B.5: Aufgabe A.2.1 S. 402: SVI-Klassendiagramm „Zielfernrohr“

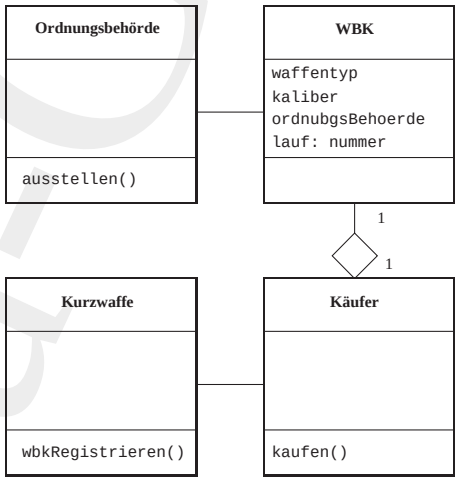


Abbildung B.6: Aufgabe A.2.2 S. 402: SVI-Klassendiagramm „Waffenbesitzkarte“

A.3.2:

- Leerzeichen zwischen den einzelnen Argumenten werden auf ein Leerzeichen reduziert.
```
>java echo Alles   klar?
Alles klar?
```

- Kein Rückgabewert — Abhilfe durch Ergänzung von `System.exit(0)`

Lösung Aufgabe A.4 S. 403:

Protokolldatei `Wert.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/gleichheit/Wert.java

D:\bonin\anwd\code>java de.fhnon.gleichheit.Wert
Fall 1: false
Fall 2: false
Fall 3: true
Fall 4: true
Fall 5: false
Fall 6: false
Fall 7: true

D:\bonin\anwd\code>
```

Lösung Aufgabe A.5 S. 404:

Protokolldatei `Scoping.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
```

```
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/scope/Scoping.java
de/fhnon/scope/Scoping.java:24: cannot resolve symbol
symbol  : variable j
location: class de.fhnon.scope.Scoping
      System.out.println("j: " + j);
                                 ^
1 error

D:\bonin\anwd\code>
```

Lösung Aufgabe A.6 S. 405:

Protokolldatei `Kontrolle.log`

```
cl3:/home/bonin/myjava:>java -fullversion
java full version "JDK1.1.4 IBM build a114-19971209" (JIT on)
cl3:/home/bonin/myjava:>javac Kontrolle.java
cl3:/home/bonin/myjava:>java Kontrolle ist besser!
Werte:
i = 6
j = 6
k = 7
m = 3.14159265358979
n = 3
p = java
vielleicht = false
wichtig = true
klar = true
cl3:/home/bonin/myjava:>
```

Lösung Aufgabe A.7 S. 406:

Protokolldatei `Iteration.log`

```
cl3:/home/bonin/myjava:>java -fullversion
java full version "JDK1.1.4 IBM build a114-19971209" (JIT on)
cl3:/home/bonin/myjava:>javac Iteration.java
cl3:/home/bonin/myjava:>java Iteration 1 2 3 4 5 6 7
```

```
Maximum UML & Java in der Anwendungsentwicklung null
Dies sind 53 Zeichen!
cl3:/home/bonin/myjava:>java Iteration 1 2
java.lang.ArrayIndexOutOfBoundsException: 2
at Iteration.main(Compiled Code)
cl3:/home/bonin/myjava:>
```

Lösung Aufgabe A.8 S. 408:

Protokolldatei `LinieProg.log`

```
d:\bonin\anwd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

d:\bonin\anwd\code>javac Linie.java LinieProg.java

d:\bonin\anwd\code>java LinieProg
0034
5.0
false

d:\bonin\anwd\code>
```

Lösung Aufgabe A.9 S. 411:

Protokolldatei `Inheritance.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/inheritance/*.java

C:\bonin\anwd\code>java de.fhnon.inheritance.Inheritance
Otti AG
```

```
Fall I  : 9
Fall II : 12106

C:\bonin\anwd\code>
```

Lösung Aufgabe A.10 S. 415:

Protokolldatei `TableProg.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/table/*.java

C:\bonin\anwd\code>java de.fhnon.table.TableProg
java de.fhnon.table.TableProg
java.lang.StackOverflowError
at de.fhnon.table.LookupTable.<init>(LookupTable.java:29)
at de.fhnon.table.LookupTable.<init>(LookupTable.java:23)
at de.fhnon.table.LookupTable.<init>(LookupTable.java:29)
...
```

Lösung Aufgabe A.12 S. 421:

Protokolldatei `Durchschnitt.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\anwd\code>javac
  de/uni_lueneburg/as/durchschnitt/Student.java

D:\bonin\anwd\code>javac
  de/uni_lueneburg/as/durchschnitt/Fach.java
```

```
D:\bonin\anwd\code>javac
  de/uni_lueneburg/as/durchschnitt/DurchschnittProg.java

D:\bonin\anwd\code>java
  de.uni_lueneburg/as/durchschnitt/DurchschnittProg
Note von Ewin Ente, Programmierung: 1.0

Durchschnittsnoten:
Programmierung = 2.0
Theoretische Informatik = 1.7

D:\bonin\anwd\code>
```

Lösung Aufgabe A.11 S. 418:

Protokolldatei `Rekursion.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/rekursion/*.java

C:\bonin\anwd\code>java de.fhnon.rekursion.Rekursion
Fakultaetsfunktion: fac(0) = 1
Anzahl der Aufrufe von fac(): 1

C:\bonin\anwd\code>java de.fhnon.rekursion.Rekursion 4
Aufruf  n = 4
Aufruf  n = 3
Aufruf  n = 2
Aufruf  n = 1
Rueckgabe wert = 1
Rueckgabe wert = 2
Rueckgabe wert = 6
Rueckgabe wert = 24
Fakultaetsfunktion: fac(4) = 24
Anzahl der Aufrufe von fac(): 5

C:\bonin\anwd\code>
```

Lösung Aufgabe A.13 S. 427:

Protokolldatei `FooBar.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/assozi/Foo.java

C:\bonin\anwd\code>dir de\fhnon\assozi\*.class
   956 Bar.class
   794 Foo.class

C:\bonin\anwd\code>java de.fhnon.assozi.Foo
Alles durchdacht? Foo!

C:\bonin\anwd\code>java de.fhnon.assozi.Bar
Alles durchdacht? Bar!
Alles durchdacht? Foo!

C:\bonin\anwd\code>
```

Lösung Aufgabe A.14 S. 429:

Protokolldatei `SlotI.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/vererbung/*.java

C:\bonin\anwd\code>java de.fhnon.vererbung.SlotI
f.getI() = 1
b.getI() = 1
```

```
b.setI(3) dann b.getI() = 3
b.setI(3) dann b.i = 2
f.setI(4) dann f.getI() = 4
f.setI(4) dann b.getI() = 3

C:\bonin\anwd\code>
```

Hinweis: Das `i` von `Foo` ist im Objekt `b` vorhanden und mit den *Gettern* und *Settern* von `Foo` verknüpft (beides vererbt!). Das `i` von `Bar` (egal ob private oder nicht) ist in `b` mit dem Initialwert 2 auch vorhanden — nur die *Gettern* und *Settern* aus den Definitionskontext greift nicht darauf zu, sondern auf des `i` von `Foo`.

Lösung Aufgabe A.15 S. 431:

Protokolldatei `QueueProg.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/queue/*.java

C:\bonin\anwd\code>java de.fhnon.queue.QueueProg
java de.fhnon.queue.QueueProg
Step 0: Queue.noOfQueues = 2
Step 1: myQ.getQueueCapacity() = 3
Step 2: myQ.getFirstItem() = Emma AG
Item = Ernst AG nicht aufgenommen!
Step 3: myQ.getFirstItem() = Willi AG
Step 4: myQ.getNoOfItemsInQueue = 2
Step 5: myQ.isItemInQueue(Ernst AG) = true

C:\bonin\anwd\code>
```

Lösung Aufgabe A.16 S. 436:

Protokolldatei `QueueProgII.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\anwd\code>javac
  de/uni_lueneburg/as/queue/*.java

D:\bonin\anwd\code>java
  de.uni_lueneburg.as.queue.QueueProg 3

Elemente in der Queue:
1
2
3

1. Element entnommen: 1

Elemente in der Queue:
2
3
Eingefügt: Emma Musterfrau
Eingefügt: Hans Otto
Eingefügt: Karl Stein

Queue in Datei queue.ser gespeichert.

Queue wurde aus Datei queue.ser gelesen.

Elemente in der Queue:
2
3
Emma Musterfrau
Hans Otto
Karl Stein

D:\bonin\anwd\code>
```

Lösung Aufgabe A.17 S. 443:

Protokolldatei `SimpleThread.log`

```
C:\bonin\anwd\code>javac -deprecation SimpleThread.java
```

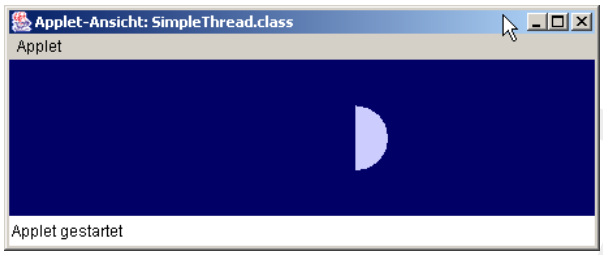


Abbildung B.7: Aufgabe A.17 S. 443: „Animierter Mond“— `appletviewer`

```
SimpleThread.java:55:
  warning: stop() in java.lang.Thread has been deprecated
         myT.stop();
            ^
1 warning

C:\bonin\anwd\code>appletviewer SimpleThread.html
x = 225 y = 60
In run() vor Thread.sleep(1000)
start() appliziert
In run() nach Thread.sleep(1000)
In run() vor Thread.sleep(1000)
In run() nach Thread.sleep(1000)
In run() vor Thread.sleep(1000)
.
.
.
In run() vor Thread.sleep(1000)
stop() appliziert

C:\bonin\anwd\code>
```

Lösung Aufgabe A.18 S. 446:

Protokolldatei `DemoAWT.log`

```
C:\bonin\anwd\code>java -version
```

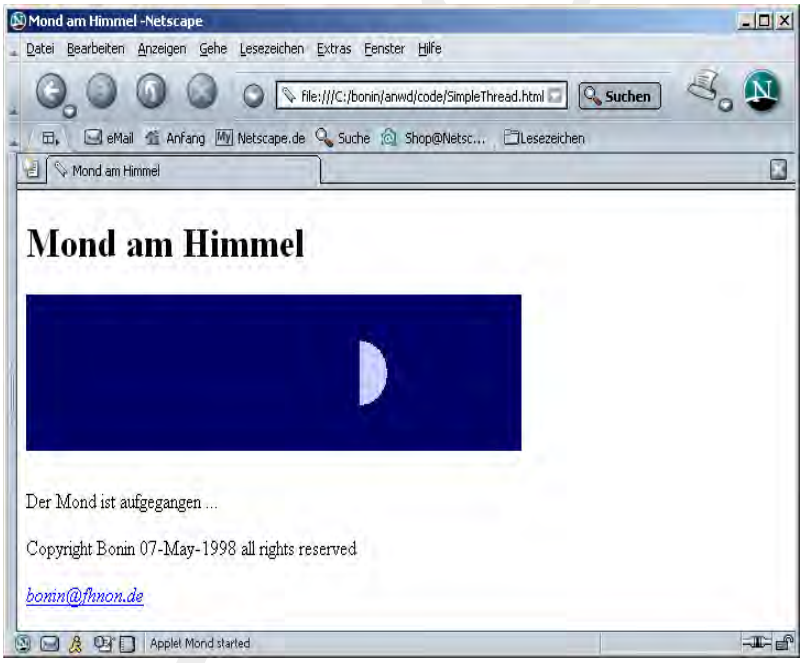


Abbildung B.8: Aufgabe A.17 S. 443: „Animierter Mond“— *Netscape 7.0*

Abbildung B.9: Aufgabe A.18 S. 446: Ausgangsfenster

```
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac -deprecation de/fhnon/awt/*.java
de/fhnon/awt/MyCanvas.java:64: warning:
  minimumSize() in java.awt.Component has been deprecated
   public Dimension minimumSize()
                    ^
de/fhnon/awt/MyCanvas.java:74: warning:
  preferredSize() in java.awt.Component has been deprecated
   public Dimension preferredSize()
                    ^
2 warnings

C:\bonin\anwd\code>appletviewer ExampleAWT.html
actionPerformed() appliziert: Anmelden!
actionPerformed() appliziert: Absagen!
stop() appliziert!

C:\bonin\anwd\code>
```

Lösung Aufgabe A.20 S. 455:
Das folgende Session-Protokoll wurde in der Emacs-Shell auf einer NT-

Abbildung B.10: Aufgabe A.18 S. 446: Hauptfenster

Workstation erstellt. Die Abbildung B.11 S. 539 zeigt die FastObjects-Klassen im Werkzeug „POET-Developer“.

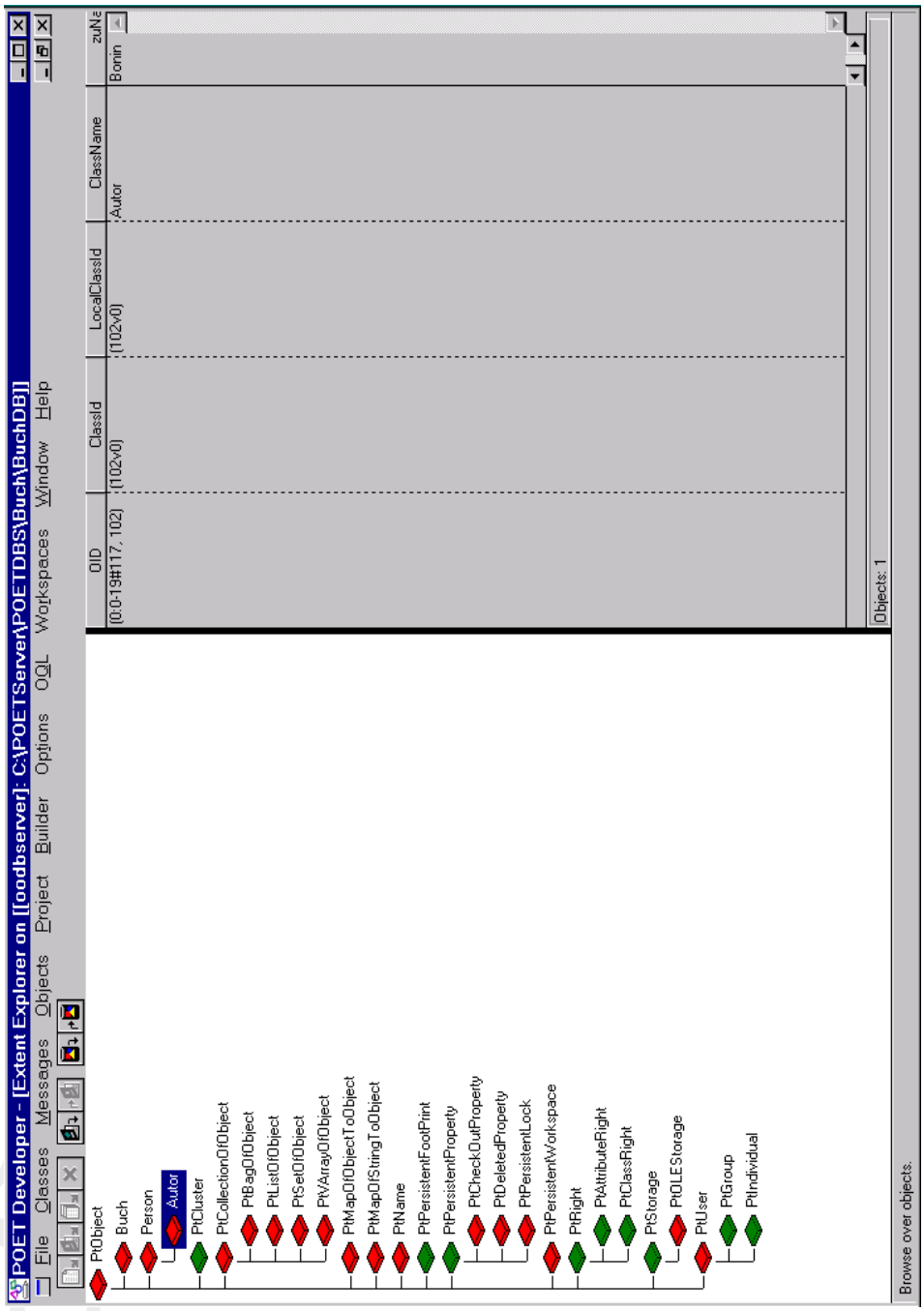


Abbildung B.11: POET Developer: Beispiel Buch

Protokolldatei `Buch.log`

```
In der Emacs-Shell auf NT-Plattform IP: 193.174.33.100


$ export CLASSPATH=C:/myjava\;C:/jdk1.1.5/lib/classes.zip\;
   C:/jdk1.1.5/lib\;C:/Programme/POET50/Lib/POETClasses.zip\;.
$ java -fullversion
java full version "JDK1.1.5K"
$ ptjavac Buch.java Person.java Autor.java
POET Java! Preprocessor Version 1.05.11
Copyright (C) 1996-97 POET Software Corporation
POET Java! Schema Creation Version 1.05.11
Copyright (C) 1997 POET Software Corporation
Registered: Person (already registered)
Registered: Buch (already registered)
Registered: Autor (already registered)
Update database: BuchDB => Done
$ java BuchBind
---> Hier kommt das POET-Login-Window
       Eingabe von Name und Password
Zuname des Autors: Bonin
Alter des Buches : 7
preWrite-Methode appliziert!
$ java BuchLookup
---> Hier kommt das POET-Login-Window
       Eingabe von Name und Password
Zuname des Autors: Bonin
Alter des Buches : 7
$ java BuchBind
---> Hier kommt das POET-Login-Window
       Eingabe von Name und Password
Zuname des Autors: Bonin
Alter des Buches : 7
PKS01 gibt es schon!
preWrite-Methode appliziert!
$
```

Protokolldatei `BuchNeu.log`
Mit (neuem) POET Java! SDK Toolkit, Version 6.1.8.76:

```
D:\bonin\anwd\code\POET>java -version
java version "1.3.1"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.1-b24)
Java HotSpot(TM) Client VM (build 1.3.1-b24, mixed mode)
D:\bonin\anwd\code\POET>javac Buch.java Autor.java Person.java
```

```
BuchBind.java BuchLookup.java
D:\bonin\anwd\code\POET>ptj -enhance -inplace
POET Java! SDK Toolkit, Version 6.1.8.76.
Copyright (C) 1996-2001 POET Software Corporation
D:\bonin\anwd\code\POET>dir _pt_*.class
18.12.2001  16:03               1.073 _pt_metaAutor.class
18.12.2001  16:03               1.180 _pt_metaBuch.class
18.12.2001  16:03               1.053 _pt_metaPerson.class
               3 Datei(en)          3.306 Bytes
               0 Verzeichnis(se),   1.733.425.152 Bytes frei
D:\bonin\anwd\code\POET>java BuchBind
Zuname des Autors: Bonin
Alter des Buches : 10
PKS01 gibt es schon!
D:\bonin\anwd\code\POET>java BuchLookup
java BuchLookup
Zuname des Autors: Bonin
Alter des Buches: 10
D:\bonin\anwd\code\POET>
d:\bonin\anwd\code\POET>java ListeLookup
Softwarekonstruktion mit LISP
Der Herr der Ringe
Die Bibel
Wo die Shareholder ihren Value bekommen
Theoretische Informatik
d:\bonin\anwd\code\POET>
```

Lösung Aufgabe A.19 S. 453:
A.19.1:
Der Java-Quellcode läßt sich fehlerfrei compilieren und ausführen. Nach dem Compilieren sind folgende Dateien entstanden:

`Regal.class`
`Regal$Schublade.class`
`Regal$Test.class`

Der Aufruf ist auf einer Unix-Plattform korrekt.

A.19.2:

Protokolldatei `Regal.log`

```
>java -fullversion
java full version "JDK 1.1.6 IBM build a116-19980529" (JIT: jitc)
>javac Regal.java
>ls R*.class
Regal$Schublade.class   Regal$Test.class        Regal.class
>java Regal\$Test
Das Regalsystem hat 2 Regal(e).
Das Regal foo hat 2 Schubladen.
Schubladeninhalte:
  Java-Disketten
  Java-Artikel
Das Regal foo wurde 4x benutzt.
>
```

Lösung Aufgabe A.21 S. 464:
A.21.1:
Es sind folgende Dateien nach dem Compilieren entstanden:

```
Foo.class
Foo$KlasseA.class
Foo$KlasseB.class
Foo$KlasseC.class
Foo$Bar.class
```

A.21.2:
Windows NT-Plattform:

```
>java de.fhnon.innerclass.Foo$Bar
```

Unix-Plattform:

```
>java de.fhnon.innerclass.Foo\$Bar
```

A.21.3:

Protokolldatei `Foo.log`

```
C:\bonin\anwd\code>java -version
java version "1.4.0_01"
```

```
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/innerclass/Foo.java

C:\bonin\anwd\code>java de.fhnon.innerclass.Foo$Bar
Slot-Wert in Instanz c: KlasseA
Anzahl der Änderungen in KlasseA: 1

C:\bonin\anwd\code>
```

Lösung Aufgabe A.22 S. 467:
A.22.1:

Protokolldatei `TelefonBuchProg.log`

```
C:\bonin\anwd\code>java -version
java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment, Standard Edition
  (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\anwd\code>javac de/fhnon/telefon/*.java

C:\bonin\anwd\code>java de.fhnon.telefon.TelefonBuchProg
TelefonEintrag: Otto +49/4131/677175
TelefonEintrag: Emma +49/4131/677144
OK --- Objekte sind gleich!

C:\bonin\anwd\code>java de.fhnon.telefon.TelefonLookupProg Emma
Emma +49/4131/677144

C:\bonin\anwd\code>
```

Außerdem wird eine Datei `tbuch.ser` erzeugt.

A.22.2:

Protokolldatei `TelefonLookupProg.log`

```
/**
 *  Primitives TelefonLookupProg.java
 *
 *@since      29-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.telefon;

import java.io.*;
import java.util.*;

public class TelefonLookupProg
{
   public static void main(String argv[])
   {
      if (argv.length != 1)
      {
         System.out.println(
               "Usage: java TelefonLookupProg name");
         System.exit(1);
      }
      String kurzname = argv[0];
      try
      {
         FileInputStream fin =
               new FileInputStream("tbuch.ser");
         ObjectInputStream in =
               new ObjectInputStream(fin);
         TelefonBuch t =
               (TelefonBuch) in.readObject();
         in.close();
         Enumeration keys =
               t.tabelle.keys();

         while (keys.hasMoreElements())
         {
            String key =
                  (String) keys.nextElement();
            TelefonEintrag te = t.getEintrag(key);
            if (te.getKurzname().equalsIgnoreCase(
                  kurzname))
            {
               System.out.println(
```

```
                    te.getKurzname() +
                    " " +
                    te.getTelefon());
                break;
             }
          }
       } catch (Exception e)
       {
          e.printStackTrace(System.out);
       }
    }
}
```

Lösung Aufgabe A.23 S. 472:
A.23.1:

Klasse K1ohne

```
/**
 *  Ergebnis der Diskussionsrunde
 *
 *@since      01-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.foo;

interface I0
{
   public void m1();


   public void m2();
}

class K1 implements I0
{


   private K4 v1;
   private K4 v2;
```

```
   private K4 v3;

   public void m1() { }

   public void m2() { }

   public static void main(String[] args)
   {
   }
}

class K2
{

   K1 s = new K1();
}

class K3 extends K2
{
   public static K4 c1;
}

class K4
{

   public void m3() { }
}
```

A.23.2:

Klasse `K1mit`

```
/**
 *  Ergebnis der Diskussionsrunde mit Getter und Setter
 *
 *@since      01-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */
```

```
package de.fhnon.foo;

interface I0
{
   public void m1();


   public void m2();
}

class K1 implements I0
{


   private K4 v1;
   private K4 v2;
   private K4 v3;


   public void m1() { }


   public void m2() { }


   public K4 getV1()
   {
      return v1;
   }


   public K4 getV2()
   {
      return v2;
   }


   public K4 getV3()
   {
      return v3;
   }


   public void setV1(K4 v1)
   {
      this.v1 = v1;
```

```
   }

   public void setV2(K4 v2)
   {
      this.v2 = v2;
   }

   public void setV3(K4 v3)
   {
      this.v3 = v3;
   }

   public static void main(String[] args)
   {
   }
}

class K2
{

   K1 s = new K1();

   public K1 getS()
   {
      return s;
   }

   public void setS(K1 s)
   {
      this.s = s;
   }
}
class K3 extends K2
{

   public static K4 c1;

   /*
```

```
     *  Klassenvariablen werden in der Regel
     *  direkt angesprochen, daher hier nur
     *  als Beispiel:
     */
    public static K4 getC1()
    {
       return c1;
    }


    public static void setC1(K4 c1)
    {
       K3.c1 = c1;
    }
}

class K4
{


    public void m3() { }
}
```

Lösung Aufgabe A.24 S. 473:
A.24.1:
Die Abbildung B.12 S. 550 zeigt das Klassendiagramm.

A.24.2:

Interface `I0`

```
/**
 *  Ergebnis der Systemanalyse
 *
 *@since      1-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.bar;
```

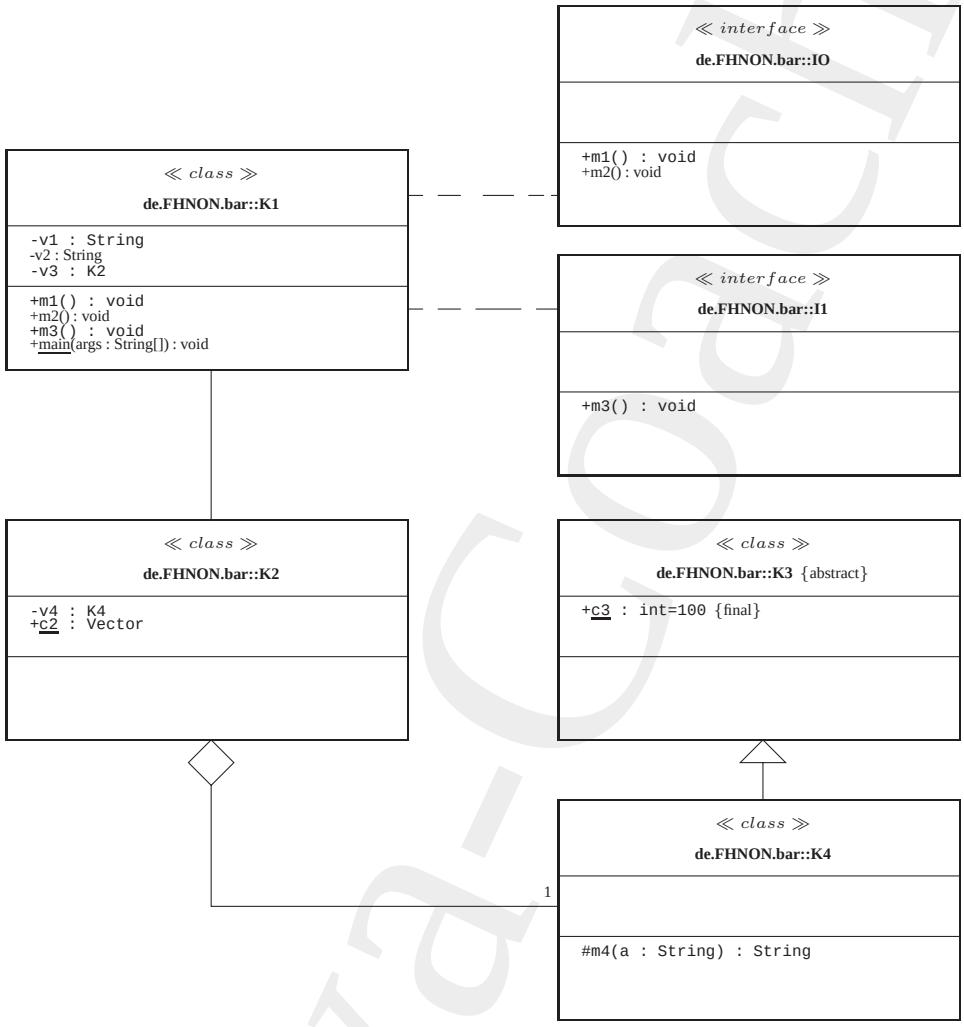


Abbildung B.12: Aufgabe A.24.1 S. 474: Klassendiagramm für `de.fhnon.as`

```
interface I0
{
   public void m1();


   public void m2();
}
```

Interface `I1`

```
/**
 *  Ergebnis der Systemanalyse
 *
 *@since      1-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.bar;

class K1 implements I0, I1
{
   private String v1;
   private String v2;
   private K2 v3;


   public void m1() { }


   public void m2() { }


   public void m3() { }


   public static void main(String[] args)
   {
   }
}
```

Klasse `K1`

```
/**
```

```
 *  Ergebnis der Systemanalyse
 *
 *@since      1-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.bar;

class K1 implements I0, I1
{
   private String v1;
   private String v2;
   private K2 v3;


   public void m1() { }


   public void m2() { }


   public void m3() { }


   public static void main(String[] args)
   {
   }
}
```

Klasse K2

```
/**
 *  Ergebnis der Systemanalyse
 *
 *@since      1-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.bar;

import java.util.Vector;

class K2
```

```
{
   private K4 v = new K4();
   public static Vector c2;
}
```

Klasse K3

```
/**
 *  Ergebnis der Systemanalyse
 *
 *@since      1-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.bar;

abstract class K3
{
   public final static int c3 = 100;
}
```

Klasse K4

```
/**
 *  Ergebnis der Systemanalyse
 *
 *@since      1-Jun-1998
 *@author     Hinrich Bonin
 *@version    1.1
 */

package de.fhnon.bar;

class K4 extends K3
{
   protected String m4(String a)
   {
      return a;
   }
}
```

A.24.3:
Windos NT-Plattform:

```
>javac .de\fhnon\bar\K1.java
>java de.fhnon.bar.K1
```

Unix-Plattform:

```
>javac .de/fhnon/bar/K1.java
>java de.fhnon.bar.K1
```

Lösung Aufgabe A.25 S. 474:
A.25.1:
Die Abbildung B.13 S. 555 zeigt das Ergebnis des `<H1>`-Konstruktes. CSS ist in größeren, grünen Buchstaben auf weißem Hintergrund in *Italic* geschrieben; gefolgt von weißen Buchstaben auf schwarzem Hintergrund.

CSS	color:	green
	background:	white
	font-style:	italic
	font-size:	28pt
(Cascading Style Sheet)	color:	white
	background:	black
	font-size:	14pt

A.25.2:
Die Unterscheidung ist erforderlich, damit das `<EM>`-Konstrukt nur in der Überschrift zu einem Font in der Größe von 28pt führt und nicht auch in der Aufzählung.

Lösung Aufgabe A.26 S. 476:
A.26.1:
Schreibfehler in Zeile 25 — korrekt: `<h1>`

A.26.2:

```
p {
  font-size: 12pt;
  color: red;
  background: white;
```

Abbildung B.13: Aufgabe A.25 S. 474: Mehrere Stylespezifikationen

```
}
h1 em {
  font-size: 28 pt;
}
h1 {
  font-size: 14pt;
  color: black;
  bachground: white;
}
em {
  font-style: italic;
  color: green;
  background: white;
}
```

A.26.3:
Das Wort „Elchtest“ in der Überschrift wird in grüner Schrägschrift auf weißem Hintergrund in einer Größe von 28pt dargestellt. Sonst wird die Überschrift in schwarzer Schrift auf geleben Hintergrund in einer Größe von 14pt dargestellt. Da ein Browser, zum Beispiel Netscape Communicator 4.05, für Überschriften häufig die Darstellung in Dickschrift (bold) voreingestellt hat, ist die gesamte Überschrift „dick“ dargestellt.

Lösung Aufgabe A.27 S. 478:
A.27.1:
Zeile 11:
Instanzvariable `weiblich` ist `public`. Das ist eine Verletzung der Datenkapselung mittels `get`- und `set`-Methoden.
Zeilen 26 und 33:
Die `set`-Methoden haben einen Rückgabewert. Standardgemäß haben `set`-Methoden keinen Rückgabewert (siehe zum Beispiel *Beans*). Zeile 41:
Direkter Zugriff auf die Variable, statt mit der Methode `getName()`.

A.27.2:

Protokolldatei `Hund.log`

```
>java Hund
Bello von der Eulenburg lebt!
Berta vom Lechgraben lebt!
Alex vom Hirschgarten lebt!
Berta vom Lechgraben
>
```

A.27.3:

Klasse `HundNorm`

```
/**
 *  Beispiel einer Rekursion innerhalb der Klasse: Vater und
 *  Mutter sind wieder vom Typ HundNorm
 *
 *@author     Hinrich Bonin
 *@version    1.0
 *@since      22-Jan-1999
 */
import java.util.*;

class HundNorm
{
   private String name = "";
   private boolean weiblich = true;
   private HundNorm mutter;
   private HundNorm vater;


   public HundNorm(String name, boolean weiblich)
   {
      setName(name);
      setWeiblich(weiblich);
      System.out.println(name + " lebt!");
   }


   public String getName()
   {
      return name;
   }
```

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


public boolean getWeiblich()
{
   return weiblich;
}


public void setWeiblich(boolean weiblich)
{
   this.weiblich = weiblich;
}


public HundNorm getMutter()
{
   return mutter;
}


public void setMutter(HundNorm mutter)
{
   this.mutter = mutter;
}


public HundNorm getVater()
{
   return vater;
}


public void setVater(HundNorm vater)
{
   this.vater = vater;
}


public static void main(String[] argv)
{
   HundNorm bello =
         new HundNorm("Bello von der Eulenburg", false);
```

```
        bello.setMutter(
              new HundNorm("Berta vom Lechgraben", true));
        bello.setVater(
              new HundNorm("Alex vom Hirschgarten", false));
        System.out.println(
              bello.getMutter().getName());
    }
}
```

Lösung Aufgabe A.28 S. 480:
Die Methoden `m2()` modifizieren ihr Parameterobjekt. Das strikte Paradigma der Objekt-Orientierung geht von Nachrichten aus, die mit Werten für Parameter an ein Objekt gesendet werden, um den Objektzustand zu modifizieren oder seine Werte zu nutzen. Eine Änderung von mitgeschickten Parameterobjekten erschwert wesentlich die Durchschaubarkeit und ist daher (möglichst) zu vermeiden. Als Korrektur bietet sich daher an:

1. Streichen der Methoden `m2()` in den Classen `C1` und `C2`.
2. In Klasse `CProg` wird in der Methode `main()` das Statement `o2.m2(o1);` durch das Statement `o1.m1(o2);` ersetzt.

Lösung Aufgabe A.29 S. 484:
A.29.1:

DTD-Datei `Partner.dtd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Bonin 30-Jun-2004            -->
<!ELEMENT partner (firma*)>
<!ELEMENT firma (adresse+, kontakt+)>
<!ATTLIST firma
       name CDATA #REQUIRED
       rechtsform CDATA #REQUIRED
       gerichtsstand CDATA #REQUIRED
>
<!ELEMENT adresse EMPTY>
```

```
<!ATTLIST adresse
        plz CDATA #REQUIRED
        ort CDATA #REQUIRED
        strasse CDATA #REQUIRED
>
<!ELEMENT kontakt EMPTY>
<!ATTLIST kontakt
        telefon CDATA #REQUIRED
        fax CDATA #REQUIRED
        email CDATA #REQUIRED
>
```

A.29.2:

Klasse `Partner0`

```
/**
 *  Erzeugung der XML-Datei Partner.xml --- Beispiellösung
 *  für die Klausur vom 30-Jun-2004 Aufgabe III.2
 *
 *@since      30-Jun-2004
 *@author     Hinrich E. G. Bonin
 *@version    1.0
 */

package de.fhnon.as.moritz;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import org.jdom.DocType;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;

public class Partner0
{
```

```
public void toXML(String sourceFile, String xmlFile,
      String dtdFile)
{
   Document document = new Document();
   document.setDocType(new DocType("partner", dtdFile));

   Element root = new Element("partner");
   document.setRootElement(root);

   try
   {
      BufferedReader reader =
            new BufferedReader(
            new FileReader(sourceFile));

      String line = null;
      Element aktuelleElement = null;

      while ((line = reader.readLine()) != null)
      {
         if (line.equals("!F"))
         {
            Element fElement = new Element("firma");
            fElement.setAttribute(
                  "name", reader.readLine());
            fElement.setAttribute(
                  "rechtsform", reader.readLine());
            fElement.setAttribute(
                  "gerichtsstand", reader.readLine());

            root.addContent(fElement);
            aktuelleElement = fElement;
         }
         if (line.equals("!A"))
         {
            Element aElement = new Element("adresse");
            aElement.setAttribute(
                  "plz", reader.readLine());
            aElement.setAttribute(
                  "ort", reader.readLine());
            aElement.setAttribute(
                  "strasse", reader.readLine());

            aktuelleElement.addContent(aElement);
         }
```

```
            if (line.equals("!K"))
            {
               Element kElement = new Element("kontakt");
               kElement.setAttribute(
                     "telefon", reader.readLine());
               kElement.setAttribute(
                     "fax", reader.readLine());
               kElement.setAttribute(
                     "email", reader.readLine());

               aktuelleElement.addContent(kElement);
            }
         }

         XMLOutputter outputter = new XMLOutputter();

         Format format = Format.getPrettyFormat();
         format.setEncoding("ISO-8859-1");
         outputter.setFormat(format);

         BufferedWriter writer = new BufferedWriter(
               new FileWriter(xmlFile));

         outputter.output(document, writer);

         writer.close();
      } catch (FileNotFoundException e)
      {
         e.printStackTrace();
      } catch (IOException e)
      {
         e.printStackTrace();
      }

   }


   public static void main(String[] args)
   {
      Partner0 partner = new Partner0();
      partner.toXML(
            "de/fhnon/as/moritz/Partner.txt",
            "de/fhnon/as/moritz/Partner.xml",
            "Partner.dtd");

   }
```

```
}
```

XML-Datei `Partner.xml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE partner SYSTEM "Partner.dtd">

<partner>
  <firma name="Otto" rechtsform="GmbH" gerichtsstand="Berlin">
    <adresse plz="D-21391" ort="Reppenstedt" strasse="Eulenburg 6" />
    <adresse plz="D-21339" ort="Lüneburg" strasse="Volgershall 1" />
    <kontakt telefon="04131-677175" fax="04131-677140"
      email="info@otto-lueneburg.com" />
  </firma>
  <firma name="Meyer" rechtsform="AG" gerichtsstand="Hamburg">
    <adresse plz="D-21000" ort="Hamburg" strasse="Alsterweg 18" />
    <adresse plz="D-21000" ort="Hamburg" strasse="Vogelsburg 2" />
    <kontakt telefon="040-11111" fax="040-11112"
      email="meyer@marktplatz-hamburg.de" />
  </firma>
</partner>
```

A.29.3:

Klasse `Partner1`

```
/**
 *  Ausgabe der XML-Datei Partner.xml --- Beispiellösung für
 *  die Klausur vom 30-Jun-2004 Aufgabe III.3
 *
 *@since      30-Jun-2004
 *@author     Hinrich E. G. Bonin
 *@version    1.0
 */
package de.fhnon.as.moritz;
```

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom.Attribute;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

public class Partner1
{

   public void showXML(String sourceFile)
   {

      try
      {
         Document document = new SAXBuilder().build(
               new File(sourceFile));
         Element root = document.getRootElement();

         List firmen = root.getChildren();

         for (int i = 0; i < firmen.size(); i++)
         {
            Element firma = (Element) firmen.get(i);
            System.out.println(
                  "Firma: " +
                  firma.getAttributeValue("name"));
            System.out.println(
                  "\tRechtsform: " +
                  firma.getAttributeValue(
                  "rechtsform"));
            System.out.println(
                  "\tGerichtsstand : " +
                  firma.getAttributeValue(
                  "gerichtsstand"));

            List infos = firma.getChildren();

            for (int j = 0; j < infos.size(); j++)
            {
               Element element = (Element) infos.get(j);

               List attribute = element.getAttributes();
```

```
                    Attribute attr1 = (Attribute) attribute.get(0);
                    Attribute attr2 = (Attribute) attribute.get(1);
                    Attribute attr3 = (Attribute) attribute.get(2);

                    System.out.println(
                          "\t" +
                          element.getName() + ":");
                    System.out.println(
                          "\t\t" +
                          attr1.getName() + ": " +
                          attr1.getValue());
                    System.out.println(
                          "\t\t" +
                          attr2.getName() + ": " +
                          attr2.getValue());
                    System.out.println(
                          "\t\t" +
                          attr3.getName() + ": " +
                          attr3.getValue());
                }

             }
          } catch (JDOMException e)
          {
             e.printStackTrace();
          } catch (IOException e)
          {
             e.printStackTrace();
          }

       }


       public static void main(String[] args)
       {
          Partner1 partner = new Partner1();
          partner.showXML(
                "de/fhnon/as/moritz/Partner.xml");
       }

    }
```

Ausgabedatei `PartnerList.txt`

```
Firma: Otto
        Rechtsform: GmbH
        Gerichtsstand : Berlin
        adresse:
                plz: D-21391
                ort: Reppenstedt
                strasse: Eulenburg 6
        adresse:
                plz: D-21339
                ort: Lüneburg
                strasse: Volgershall 1
        kontakt:
                telefon: 04131-677175
                fax: 04131-677140
                email: info@otto-lueneburg.com
Firma: Meyer
        Rechtsform: AG
        Gerichtsstand : Hamburg
        adresse:
                plz: D-21000
                ort: Hamburg
                strasse: Alsterweg 18
        adresse:
                plz: D-21000
                ort: Hamburg
                strasse: Vogelsburg 2
        kontakt:
                telefon: 040-11111
                fax: 040-11112
                email: meyer@marktplatz-hamburg.de
```

Protokolldatei `Partner.log`

```
D:\bonin\anwd\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\anwd\code>javac de/fhnon/as/moritz/Partner0.java
```

```
D:\bonin\anwd\code>java de.fhnon.as.moritz/Partner0

D:\bonin\anwd\code>javac de/fhnon/as/moritz/Partner1.java

D:\bonin\anwd\code>java de.fhnon.as.moritz/Partner1
  >de/fhnon/as/moritz/PartnerList.txt

D:\bonin\anwd\code>
```

Anhang C

Hinweise zur Nutzung von J2SE SDK

C.1 Java^{TM} auf der AIX-Plattform

Es wird als Erläuterungsbeispiel angenommen, daß die Datei `Foo.java` mit dem Java-Quellcode sich auf dem AIX-Rechner mit der IP-Nummer `193.174.32.3` im Verzeichnis `/u/bonin/myjava` befindet. Der Benutzer befindet sich auf diesem Rechner (`rzserv2`) in der Korn-Shell im Verzeichnis `/home/bonin`. Mit dem Kommando:

```
. java.env
```

werden die Umgebungsvariablen gesetzt (↪ Tabelle C.1 S. 570). (Hinweis: Punkt nicht vergessen, damit in der aktuellen Shell die Variablen gesetzt sind!) Mit dem Kommando:

```
export CLASSPATH=/home/bonin/myjava:$CLASSPATH
```

CLASS-PATH

wird dafür gesorgt, daß die Klasse `Foo` beim Aufruf gefunden werden kann. Zum Compilieren und Anwenden von `Foo` sind dann folgende Kommandos einzugeben (↪ Abschnitt 6.1.2 S. 90):

```
> javac myjava/Foo.java
> java Foo
```

```
#!/usr/bin/ksh -x
# Startdatei zum Setzen der Environmentvariablen
# Bonin: 17-Oct-1997
#   Update: 21-Oct-1997; 24-Oct-1997; 01-Nov-1997; 26-Mar-1998;
#           08-Jun-1998
#
# Klare Anfangsposition
unset LD_LIBRARY_PATH
unset JAVA_THREADS
unset JAVA_COMPILER
unset JAVA_HOME
unset  PATH
# Pfad fuer javac, java usw. setzen
export PATH=/usr/lpp/J1.1.6/bin:/usr/bin:/etc:/usr/sbin:/usr/ucb:$HOME/bin:\
/usr/bin/X11:/sbin:/usr/local/emacs/etc:.:/usr/local/bin
#
# Klasenzugriffspfad setzen
unset CLASSPATH
export CLASSPATH=/usr/lpp/J1.1.6/lib/classes.zip:/usr/lpp/J1.1.6/lib:.
# Steht die Anwendungsklasse zum Beispiel unter /home/bonin/myjava
#   und arbeitet man nicht unter diesem Verzeichnis
#   dann
#     export CLASSPATH=/home/bonin/myjava:$CLASSPATH
#
#     ausfuehren vor Aufruf aus dem beliebigen Verzeichnis.
#   (Ist Anwendungsklasse im aktuellen Verzeichnis, dann
#    durch obigen Punkt im CLASSPATH ausreichender Verweis.)
#
# Erlauerterung siehe Abschnitt Java auf der AIX Plattform
export JAVA_COMPILER=jitc
export JAVA_THREADS=gt
# End of file cl3:/u/bonin/java.env
```

Tabelle C.1: Java-Umgebungsvariablen für die AIX-Plattform

C.2 Java^{TM} auf der Windows-Plattform

Es wird als Erläuterungsbeispiel wieder[1] angenommen, daß die Datei `Foo.java` mit dem Java-Quellcode sich auf dem NT-Rechner mit der IP-Nummer `193.174.33.100` im Verzeichnis `C:\myjava` befindet. Der Benutzer befindet sich auf diesem Rechner in einer DOS-Shell im Verzeichnis `C:\temp`. Das *Java Development Kit* befindet sich hier unter:

[1] ↪ Abschnitt 6.1.2 S. 90.

`C:\jdk1.1.3\bin`

Der notwendige Zugriffspfad wird mit folgendem Kommando gesetzt:

```
path=C:\jdk1.1.3\bin;%path%
```

Zum Setzen der Umgebungsvariablen wird folgendes Kommando verwendet, weil die Java-Standardklassen sich in der Datei `classes.zip` befinden:

```
set CLASSPATH=C:\myjava;C:\jdk1.1.3\lib\classes.zip;C:\jdk1.1.3\lib
```

Zur Compilation und Applikation werden folgende Kommandos auf der DOS-Shell-Ebene eingegeben:

```
C:\temp>java -version
java version "1.1.3"
C:\temp>javac C:\myjava\Foo.java
C:\temp>java Foo
Kein Argument: Java ist ...! Fri Oct 24 10:59:42 GMT+01:00 1997
C:\temp>java Foo is my %CLASSPATH%
Eingabeteil:0
+is+
Eingabeteil:1
+my+
Eingabeteil:2
+CLASSPATH=C:\myjava;C:\jdk1.1.3\lib\classes.zip;C:\jdk1.1.3\lib+
Neuer Wert: Java ist ...! Fri Oct 24 11:02:13 GMT+01:00 1997
C:\temp>echo %CLASSPATH%
CLASSPATH=C:\myjava;C:\jdk1.1.3\lib\classes.zip;C:\jdk1.1.3\lib
C:\temp>
```

Hinweis: Für den GNU Emacs auf Windows ist in der Emacs-Shell ein besondere Kombination aus UNIX und DOS-Shell-Kommandos erforderlich. Da der Emacs dem Benutzer UNIX-Kommandos emuliert gibt es „Probleme“ mit Sonderzeichen. Es sind folgende Zeichenkombinationen zu wählen:

```
# Fuer die Emxacs Shell
# Achtung mit Backslash und Semikolon zum Trennen
export CLASSPATH=C:/myjava\;C:/jdk1.1.5/lib/classes.zip\;C:/jdk1.1.5/lib\;.
# Bonin 7-Mai-1998
```

Hinweis: Um die Windows-IP-Konfiguration festzustellen ist das DOS-Kommando `ipconfig /all` hilfreich. Die folgende Datei zeigt das Ergebnis auf einem Toschiba Notebook Tecra:

```
D:\bonin\anwd>ipconfig /all

Windows-IP-Konfiguration

        Hostname. . . . . . . . . . . . : BONIN-XP-S1NB
        Primäres DNS-Suffix . . . . . . :
        Knotentyp . . . . . . . . . . . : Hybrid
        IP-Routing aktiviert. . . . . . : Nein
        WINS-Proxy aktiviert. . . . . . : Nein

Ethernetadapter LAN-Verbindung 2:

        Verbindungsspezifisches DNS-Suffix: fhnon.de
        Beschreibung. . . . . . . . . . . : Grey Cell 2200-Ethernetkarte
        Physikalische Adresse . . . . . . : 00-47-43-60-93-54
        DHCP aktiviert. . . . . . . . . . : Nein
        IP-Adresse. . . . . . . . . . . . : 193.174.33.66
        Subnetzmaske. . . . . . . . . . . : 255.255.255.0
        Standardgateway . . . . . . . . . : 193.174.33.66
        DNS-Server. . . . . . . . . . . . : 193.174.32.18
                                            193.174.32.23
        Primärer WINS-Server. . . . . . . : 193.174.33.243

Ethernetadapter LAN-Verbindung:

        Medienstatus. . . . . . . . . . . : Es besteht keine Verbindung
        Beschreibung. . . . . . . . . . . : Intel(R) PRO/100 VE
                                              Network Connection
        Physikalische Adresse . . . . . . : 00-A0-D1-D2-7A-E2

Ethernetadapter Drahtlose Netzwerkverbindung:

        Medienstatus. . . . . . . . . . . : Es besteht keine Verbindung
        Beschreibung. . . . . . . . . . . : Atheros AR5001X Mini PCI
                                              Wireless Network Adapter
        Physikalische Adresse . . . . . . : 00-90-96-40-01-95

D:\bonin\anwd>
```

Anhang D

Quellen

D.1 Literaturverzeichnis

Literaturverzeichnis

[Abelson85] Harold Abelson / Gerald Jay Sussman / Julie Sussman; Structure and Interpretation of Computer Programs, Cambridge, Massachusetts and others (The MIT Press/McGraw-Hill) 1985.

[Arnold/Gosling96] Ken Arnold / James Gosling; The Java Programming Language (Addison-Wesley) 1996.

[Alur/Crupi/Malks01] Deepak Alur / John Crupi / Dan Malks; Core J2EE Patterns, Sun Microsystems Press, Prentice Hall PTR, 2001, in deutsch von Frank Langenau; Core J2EE Patterns — Die besten Praxislösungen und Design-Strategien, 2002, ISBN 3-8272-6313-1.

[Belli88] Fevzi Belli; Einführung in die logische Programmierung mit Prolog, Mannheim Wien Zürich (Bibliographisches Institut), 2. Auflage 1988.

[Bonin88] Hinrich Bonin; Die Planung komplexer Vorhaben der Verwaltungsautomation, Heidelberg (R. v. Decker & C. F. Müller), 1988 (Schriftenreihe Verwaltungsinformatik; Bd. 3).

[Bonin89] Hinrich E. G. Bonin; Objektorientierte Programmierung in LISP – Standard-LISP und objektorientierte Erweiterungen, in: Handbuch der Modernen Datenverarbeitung (HMD), Heft 145, Januar 1989, 26. Jahrgang, S. 45 – 56.

[Bonin91a] Hinrich Bonin; Cooperative Production of Documents, in: [Traunmüler91], pp. 39–55.

[Bonin91b] Hinrich E. G. Bonin; Software-Konstruktion mit LISP, Berlin New York (Walter de Gruyter), 1991.

[Bonin92a] Hinrich E. G. Bonin; Arbeitstechniken für die Softwareentwicklung, (3. überarbeitete Auflage Februar 1994), FINAL, 2. Jahrgang Heft 2, 10. September 1992, [FINAL].

[Bonin92b] Hinrich E. G. Bonin; Teamwork between Non-Equals — Check-in & Check-out model for Producing Documents in a Hierarchy, in: SIGOIS Bulletin, Volume 13, Number 3, December 1992, (ACM Press), pp. 18–27.

[Bonin92c] Hinrich E. G. Bonin; Object-Orientedness – a New Boxologie, FINAL Heft 3, 1992 (ISSN 0939-8821).

[Bonin93] Hinrich E. G. Bonin; The Joy of Computer Science, — Skript zur Vorlesung EDV —, Unvollständige Vorabfassung (4. Auflage März 1995), FINAL, 3. Jahrgang Heft 5, 20. September 1993,[FINAL].

[Bonin94] Hinrich E. G. Bonin; Groupware-Systeme: Eine Perspektive für die öffentliche Verwaltung, in: Verwaltungsführung / Organisation / Personal (VOP), Heft 3, 1994, S. 170–176.

[Bonin96] Hinrich Bonin; <HTML>-Ratgeber — Multimediadokumente im World-Wide Web programmieren, München Wien (Carl Hanser Verlag), 1996.

[Booch94] G. Booch; Object-oriented analysis and design with applications, 2nd ed., Redwood City (Benjamin/Cummings), 1994. Deutsche Ausgabe: Objektorientierte Analyse und Design, Mit praktischen Anwendungsbeispielen, Bonn (Addision-Wesley), 1994.

[Bobrow/Moon88] Daniel G. Bobrow / David Moon u. a.; Common Lisp Object Systems Specification, ANSI X3J13 Document 88-002R, American National Standards Insitute, Washington, DC, June 1988 (veröffentlicht in: SIG-PLAN Notices, Band 23, Special Issus, September 1988).

[Broy/Siedersleben02] Manfred Broy / Johannes Siedersleben; Objektoientierte Programmierung und Softwareentwicklung — Eine kritische Einschätzung, in: Informatik-Spektrum, Band 25, Heft 1, Februar 2002, S. 3–11.

[Broy/Siedersleben02] Erwiderung zu ↪ [Jähnichen/Herrmann02], in: Informatik-Spektrum, Band 26, Heft 1, Februar 2003, S. 56–58.

[Clocksin/Mellish87] W. F. Clocksin / C. S. Mellish; Programming in Prolog, Berlin New York u.a. (Springer-Verlag) Third Edition, 1987.

[Dahl+67] O.-J. Dahl / KB. Myrhaug / K. Nygaard; Simula 67 — Common Base Language. Technical Report N. S-22, Norsk Regnesentral (Norwegian Computing Center, Oslo 1967. [Hinweis: Orginalarbeit zitiert nach ↪ [Broy/Siedersleben02].]

[Eckel02] Bruce Eckel; Thinking in Java — The Definitive Introduction to Object-Oriented Programming in the Language of the World-Wide-Web, Upper Saddle River, NJ 07458 (Prentice Hall PTR), 3rd. edition, ISBN 0-13-100287-2.

[Embley92] David W. Embley / Barry D. Kurtz / Scott N. Woodfield; Object-Oriented Systems Analysis – A Model-Driven Approach, Englewood Cliffs, New Jersey (Yourdon Press), 1992.

[Flanagan96] David Flanagan; Java in a Nutshell, Deutsche Übersetzung von Konstantin Agouros, Köln (O´Reilly), 1996.

[Flanagan97] David Flanagan; Java in a Nutshell, Second Edition, updated for Java 1.1, Köln (O´Reilly), May 1997.

[FINAL] Fachhochschule Nordostniedersachsen, Informatik, Arbeitsberichte, Lüneburg (FINAL) herausgegeben von Hinrich E. G. Bonin, ISSN 0939-8821, ab 7. Jahrgang (1997) auf CD-ROM, beziehbar: FH NON, Volgershall 1, D-21339 Lüneburg, Germany.

[Forbrig01] Peter Forbrig; Objektorientierte Softwareentwicklung mit UML, Informatik interaktiv, München Wien (Fachbuchverlag Leipzig / Carl Hanser), 2001, ISBN 3-446-21572-7.

[Fowler99] Martin Fowler; Refactoring: Improving the Design of Existing Code, Addison Wesley 1999.

[Freeman/Ince96] Adam Freeman / Darrel Ince; active java — Object-Oriented Programming for the World Wide Web, Harlow, England. u. a. (Addison-Wesley) 1996. [Hinweis: Einige Fehler in den Java-Quellecode-Beispielen.]

[Gabriel91] Richard P. Gabriel / John L. White / Daniel G. Bobrow; CLOS: Integrating Object-Oriented and Functional Programming, in: Communications of the ACM, Vol. 34, No. 9, September 1991, pp. 29 – 38.

[Goldberg83] Adele Goldberg; Smalltalk-80: The Interactive Programming Environment, Reading 1983 (Addison-Wesley) [Im Smalltalk-Jargon genannt: „das orangefarbene Buch“].

[Goldberg/Robson83] Adele Goldberg / Dave Robson; Smalltalk-80: the lanugage, Reading, Massachusetts u. a. (Addision-Wesley) 1983. [Im Smalltalk-Jargon genannt: „das blaue Buch“].

[Hau/Mertens02] . Michael Hau / Peter Mertens; Computergestützte Auswahl komponentenbasierter Anwendungssysteme, in: Informatik-Spektrum, Band 25, Heft 5, Oktober 2002, S. 331–340.

[Hist97] Jason English (1997); It all started with a blunt letter,
`http://www.javasoft.com/nav/whatis/index.html`
(Zugriff: 20-Sep-1997)
Michael O’Connell (Sun World Online, 1995); Java: The inside story — We interview Java’s creators to find what they had in mind.
`http://www.sun.com/sunworldonline/swol-07-1995/swol-07-java.html`
(Zugriff: 20-Sep-1997)

[HTML4.0] Dave Raggett / Arnaud Le Hors / Ian Jacobs ; HTML 4.0 Spezifikation, W3C Recommendation 18-Dec-1997,
`http://www.w3.org/TR/REC-html40-971218.html`
(Zugriff: 29-Jan-1998).

[Hoff/Shaio96] Arthur van Hoff / Sami Shaio / Orca Starbuck; HOOKED ON JAVA, (Addision-Wesley Publishing Company) 1996.

[IBM-Francisco98] IBM Corporation; San Francisco Projekt, Java Coding Tips
`http://www.ibm.com/Java/Sanfrancisco/tips/javatips.html`
(Zugriff: 09-Jul-1998)

[ITS97] ITS, Kalatog Fernreisen Sommer 97.

[JavaSpec] James Gosling / Bill Joy / Guy Steele; The Java Language Specification, (Addison-Wesley) 1996;
`http://www.javasoft.com/docs/books/jls/html/index.html`
Änderungen für Java 1.1;
`http://www.javasoft.com/docs/books/jls/html/1.1Update.html`
(Zugriff: 20-Sep-1997)

[Jacobsen92] Ivar Jacobsen / M. Christerson / P. Jonsson / G. Övergaard; Object-Oriented Software Engineering, A Use Case Driver Approach, Workingham (Addision-Wesley) 1992.

[Jähnichen/Herrmann02] Stefan Jähnichen / Stephan Herrmann; Was, bitte, bedeutet Objektorientierung? in: Informatik-Spektrum, Band 25, Heft 4, August 2002, S. 266–275. [Hinweis: Ein Diskussionsbeitrag zu ↪ [Broy/Siedersleben02].]

[Kczales91] Gregor Kiczales / Jim des Rivieres / Daniel G. Bobrow; The Art of the Metaobject Protocol, Cambridge, Massachusetts, London (The MIT Press) 1991.

[Kim/Lochovsky89] Won Kim / Frederick H. Lochovsky (Eds.); Object-Oriented Concepts, Databases, and Applications, Reading, Massachusetts (Addison-Wesley) 1989.

[Larman98] Craig Larman; Applying UML and Patterns — An Introduction to Object-Oriented Analysis and Design, New Jersey (Prentice Hall) 1998.

[LieBos96] Hakon Wium Lie / Bert Bos; Cascading Style Sheets, level 1, W3C Recommendation 17-Dec-1996
`http://www.w3.org/StyleSheets/core/examples/REC-CSS1-961217.html`
(Zugriff: 23-Jun-1998).

[Lieberman81] H. Lieberman; Thinking About Lots of Things at Once Without Getting Confused - Parallelism in ACT-1, Cambridge, MIT AIMemo 626, May 1981.

[Miller02] Joaquin Miller; What UML Should Be, in: Communications of the ACM, Vol. 45, No. 11, November 2002, pp. 67–69. [Hinweis: „Which of the proposed revisions will bring it closer to meeting user needs and winning tool-vendor commitment?„]

[Monson01] Richard Monson-Haefel; Enterprise JavaBeans^{TM}, third edition, Beijing u. a. (O´Reilly), 2001, ISBN 0-596-00226-2.

[Oechsle01] Rainer Oechsle; Parallele Programmierung mit Java Threads, Informatik interaktiv, München Wien (Fachbuchverlag Leipzig / Carl Hanser), 2001, ISBN 3-446-21780-0.

[Oestereich97] Bernd Oestereich; Objekt-orientierte Softwareentwicklung mit der Unified Modeling Language,(3. aktualisierte Auflage) München Wien (R. Oldenbourg Verlag) 1997.

[Orfali/Harkey97] Robert Orfali / Dan Harkey; Client/Server Programming with JAVA and CORBA, New York u. a. (John Wiley & Sons) 1997.

[Ourosoff02] Nick Ourosoff; Primitive Types in Java Considered Harmful — Expression evalutation raises doubts about Java as an exemplar programming language when teaching the object-oriented paradigm, in: Communications of the ACM, August 2002, Vol.45, No. 8, pp. 105–106. [Remark: “Why should one start a text focusing on teaching the OO paradigm by featuring the part of Java that is not object-oriented?”.]

[Partl98] Hubert Partl; Java — Einführung; Kursunterlage, Version April 1998,
`http://www.boku.ac.at/javaeinf/`
(Zugriff: 08-Mai-1998).

[Pooley/Wilcox04] Rob Pooley / Pauline Wilcox; Applying UML: Advanced Application, Amsterdam u. a. (Elsevier, Butterworth, Heinemann) 2004, ISBN 0-7506-5683-2. [Remark: This book addresses the practical issues people face when adopting the UML.]

[Rational97] Rational Software; Unified Modeling Language, Version 1.1, 01-Sep-1997, UML Summary, UML Metamodel, UML Notation Guide, UML Semantics, UML Extension Business Modeling, Object Constraint Specification,
`http://www.rational.com/uml/1.1/`
(Zugriff: 11-Nov-1997)

[Rumbaugh91] J. Rumbaugh / M. Blaha / W. Premerlani / F. Eddy / W. Lorenson; Objekt-oriented Modelling and Design, Englewood Cliffs (Prentice-Hall), 1991

[RRZN97] Regionales Rechenzentrum für Niedersachsen / Universität Hannover (RRZN); Java — Begleitmaterial zu Vorlesungen / Kursen, 1. Auflage Juni 1997, RRZN-Klassifikationsschlüssel: SPR.JAV2; beziehbar: FH NON, Volgershall 1, D-21339 Lüneburg, Germany. [Hinweis: Einige Fehler in den Java-Quellecode-Beispielen.]

[Schader+03] Martin Schader / Lars Schmidt-Thieme; Java — Eine Einführung, Berlin Heidelberg (Springer), 4. Auflage 2003, ISBN 3-540-00663-X. {Hinweis: Das Buch enthält gelungene Übungen mit Lösungen (auf der beigefügten CD-ROM).}

[Shavor+03] Sherry Shavor / Jim D'Anjou / Scott Fairbrother / Dan Kehn / John Kellerman / Pat McCarty; The Java^{TM} Developer's Guide to Eclipse, Boston u. a. (Addison-Wesley), ISBN 0-321-15964-0. {Hinweis: "This Book does an excellent job of helping you learn Eclipse."}

[Sommerville89] Ian Sommerville; Software Engineering, Wokingham, England u.a. (Addison Wesley) Third Edition, 1989.

[Stroustrup86] Bjarne Stroustrup; The C++ Programming Language, Reading Massachusetts (Addison-Wesley) 1986 (corrected reprinting, 1987).

[Stroustrup89] Bjarne Stroustrup; The Evolution of C++: 1985 to 1989, in: Computing Systems, 2(3) Summer 1989, pp. 191 – 250.

[Sun97] Sun Microsystems; Schwerpunkt: The Road To Java, in: SunNews, November 1997, S. 4–5, Sun Microsystems GmbH, Bretonischer Ring 3, D-85630 Grasbrunn
`http://www.sun.de`
(Zugriff: 05-Dec-1997).

[Sun98] Sun Microsystems; SunNews — Das Magazin für Network Computing, Mai 1998, Sun Microsystems GmbH, Bretonischer Ring 3, D-85630 Grasbrunn
`http://www.sun.de`
(Zugriff: 05-Dec-1997).

[SunRMI98] Sun Microsystems; Java Remote Method Invocation Specification, Revision 1.42, JDK 1.2 Beta 1, Oktober 1997
`http://java.sun.com:80//products/jdk/rmi/index.html`
(Zugriff: 16-Jun-1998).

[TakeFive97] TakeFive Software; SNiFF+J, Release 2.3.1 for Unix and Windows, SNiFF+ Java Solution at a Glance, Product Number SNiFF-TG1-231, 19-Jun-1997, Europa: TakeFive Software GmbH, A-5020 Salzburg, email: `info@takefive.co.at`

[Traunmüler91] Roland Traunmüller (Editor); Governmental and Municipal Information Systems, II, IFIP, Amsterdam, u. a. (North-Holland), 1991.

[Tyma98] Paul Tyma; Why are we using Java again?, in: Communications of the ACM, June 1998, Vol.41, No. 6, pp. 38–42.

[Ungar/Smith91] David Ungar / Randall B. Smith; SELF: The Power of Simplicity, in: LISP and Symbolic Computation (Kluwer Academic Publishers), Volume 4, Number 3, July 1991, pp. 187 – 205.

[Vanderburg97] Glenn Vanderburg, et al.; Maximum Java 1.1, Indianapolis (sams net) 1997. [Hinweis: „The ultimate source for advanced Java programming techniques.„]

D.2 Web-Quellen

JBoss

Application Server — JBoss

JBoss ist ein sehr verbreiteter *Open Source Application Server* (↪ Bild D.1 S. 581). JBoss ist eine *Middleware* mit J2EE-Unterstützung auf der Basis von *Java Management eXtensions* (JMX). JBoss implementiert eine Sicherheitsschicht, integriert JAAS und hat eine Unterstützung für *Aspect Oriented Programming* (AOP).
`http://www.jboss.org/overview` (online 23-Jan-2004)

Castor

Data Binding Framework — Castor

Castor ist ein *Open Source Data Binding Framework* für Java^{TM}. Es ist ein zweckmäßiges Werkzeug für das Zusammenspiel von Java-Objekten, XML-Dokumenten und SQL-Tabellen. Castor unterstützt Java ⇒ XML-*Binding*, Java ⇒ SQL-Persistenz und in diesem Kontext nützliche Funktionen.
`http://castor.exolab.org/` (online 23-Jan-2004)

JDOM

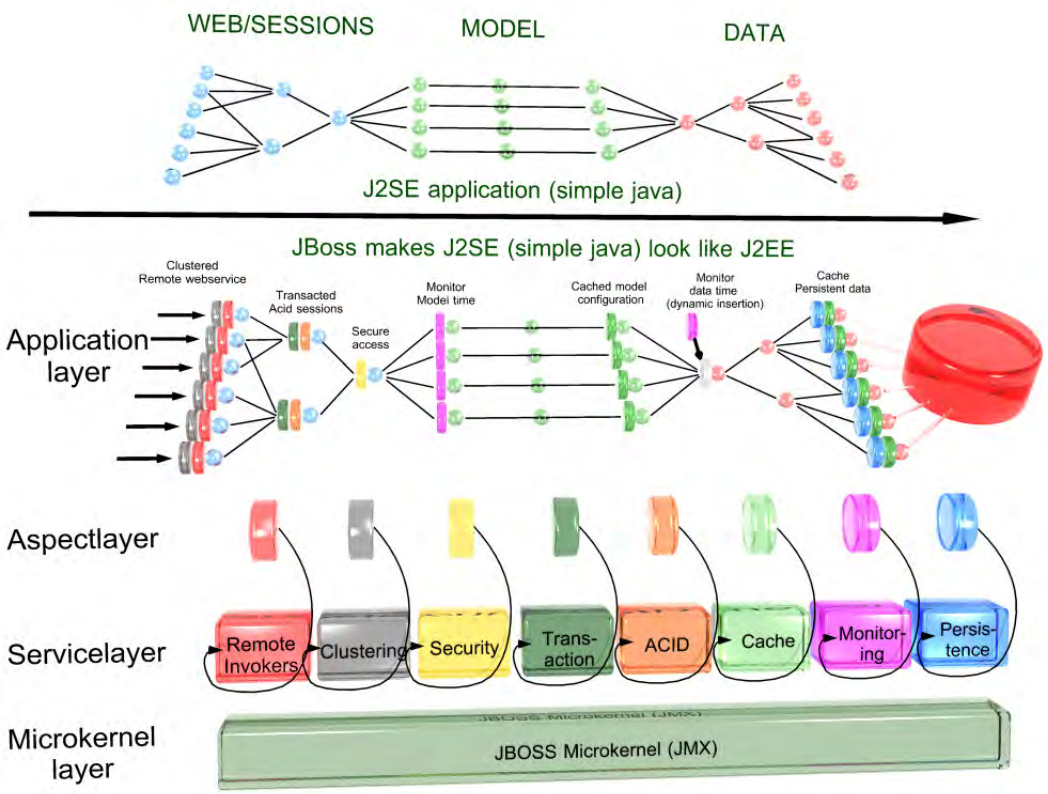
JDOM — *SAX & DOM*

JDOM stellt Klassen für SAX (*Simple API for XML*) und DOM (*Document Object Model*) bereit.
`http://www.jdom.org` (online 19-May-2004)

Ant

Ant — *Build Tool*

Ant ist ein Werkzeug zum Erstellen einer Anwendung auf der Basis einer XML-Datei.



Legende:

Quelle: `http://www.jboss.org/overview` (online 23-Jan-2004)

Abbildung D.1: JBoss

`http://ant.apache.org/`
(online 10-Jun-2004)

JUnit

JUnit **— automatisierte Tests**
JUnit stellt ein Rahmenwerk zum Testen bereit.
`http://www.junit.org/index.htm` (online 10-Jun-2004)

J2SE-SDK-Dokumentation
`http://java.sun.com/docs/index.html`

EJB-Homepage
`http://java.sun.com/products/ejb/`

JDBC-Homepage
`http://java.sun.com/products/jdbc/`

JMS-Homepage
`http://java.sun.com/products/jms/`

JNDI-Homepage
`http://java.sun.com/products/jndi/`

JSP-Homepage
`http://java.sun.com/products/jsp/`

Homepage zu *Servlets*
`http://java.sun.com/products/servlet/`

DTD für EJB *Deployment Descriptors*
`http://java.sun.com/dtd/ejb-jar_2_0.dtd`

Zur Geschichte von Java^{TM}
`http://java.sun.com/nav/whatis/storyofjava.html`

Java^{TM}-Erweiterung — *Pizza Compiler*
Der *Pizza Compiler* erweitert Java um drei neue Feeatures:

- *Generics* — aka Parametric polymorphism
- *Function pointers* — aka First-class functions
- *Class cases and pattern matching* — aka Algebraic types

`http://pizzacompiler.sourceforge.net/`

D.3 Anmerkungen zum JAVA^{TM}–COACH

Mein Web-Server: `http://as.fhnon.de/`
Unter diesem Web-Server werden weitere Informationen zu diesem Buch angeboten.

Mit folgender Software wurde das Dokument JAVA^{TM}–COACH erstellt:

Editor: GNU Emacs 21.2.1; jEdit 4.1 final

Layout: TeX, Version 3.14159 (Web2c 7.3.7x), LaTeX2e <2000/06/01>; Document Class: book 2001/04/21 v1.4e Standard LaTeX document class

Hardcopy: Corel CAPTURE 11; Corel PHOTO-PAINT 11 (version 10.427)

Figure: Microsoft Visio 2000 SR1 (6.0.2072)

Index: makeindex, version 2.13 [07-Mar-1997] (using kpathsea)

DVI→PS: LaTeX-File (Device Independent) to Postscript: dvips(k) 5.90a Copyright 2002 Radical Eye Software (www.radicaleye.com)

PS→PDF: Postscript file to PDF-File: Adobe Acrobat Distiller 6.0 Professional

Security: Adobe Acrobat 6.0 Professional (Version 5.0)

D.4 Abkürzungen und Akronyme

ACID	Atomicity, Consistency, Isolation, Durability
AIX	Advanced Interactive Executive IBM's Implementation eines UNIX-Operating Systems
AJDE	AspectJ Development Environment
AOP	The term aspect-oriented programming is attributed to Kiczales et. a.
API	Application programming interface

BDK	Beans Development Kit
BNF	Backus-Naur-Form
CICS	Customer Information Control System
CGI	Common Gateway Interface
CLOS	Common Lisp Object System
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
CVS	Concurrent Versions System
CTM	Component Transaction Monitor
DBMS	Data Base Manangement System
DOM	Document Object Model
DTD	Document Type Definition
DTP	Desktop Publishing
DVI	LaTeX's Device Independent File Format
EIS	Enterprise Infomation Systems (Server)
EJB	Enterprise JavaBeans
Emacs	Editing Macros gigantic, functionally rich editor
ERP	Enterprise Resource Planing Systems
FTP	File Transfer Program
GJ	A Generic Java Language Extension
GNU	GNU's Not Unix
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IANA	Internet Assigned Numbers Authority
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
IIOP	Internet Inter-Operability Protocol
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAR	Java Archiv
JDBC	Java Database Connectivity
JDK	Java Development Kit
JIT	Just-In-Time-Compiler

JMI	Java Metadata Interface
JMS	Java Messaging Service
JNI	Java Native Method Interface
JNDI	Java Naming and Directory Interface
JPEG	Joint Photographics Expert Group
JRE	Java Runtime Environment
JSP	Java Server Pages
JTA	Java Transaction API
JVM	Java Virtual Maschine
JXTA	Juxtapose (pronounced *juxta* — Peer-to-Peer-Framework)
LDAP	Lightweigth Directory Access Protocol
LISP	List Processing
MDA	OMG's Model Driven Architecture
MOF	OMG's Meta-Object Facility
MOM	Message-Oriented Middleware
MOP	Metaobject protocol; a programmer could override the default behavior of the dispatch method in order to affect what happens when a virtual function is called.
`NaN`	Not a Number
NT	Microsoft Windows New Technology 32-Bit-Betriebssystem mit Multithreading und Multitasking
OAK	Object Application Kernel
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
OMG	Object Management Group
OOP	Object-oriented Programming
OQL	Object Query Language
P2P	Peer-to-Peer
PGP	Pretty Good Privacy
POP	Post object-oriented programming
POP3	Post Office Protocol 3
PROLOG	PROgramming in LOGic
PS	Postscript file
RAS	Reliability, Availability, Serviceability
RISC	Reduced Instruction Set
RMI	Java Remote Method Invocation Protocol

RCS	Revision Control System
RPC	Remote Procedure Call
SAX	Simple API for XML
SCCS	Source Code Control System
SDK	Software Developer's Kit
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transport Protocol
SSH	Secure Shell
SSL	Secure Socket Layer
SQL	Standard Query Language
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URI	Universal Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XHTML	Extensible Hypertext Markup Language
XMI	XML Metadata Interchange format
XML	Extensible markup language

Anhang E

Index

Index

Index