

Der JAVA™ –COACH

Trainingsziel — Just a valid application

Hinrich E. G. Bonin¹

¹Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. Bonin lehrt Informatik an der Leuphana Universität Lüneburg, Email: bonin@uni-lueneburg.de, Adresse: An der Eulenburg 6, D-21391 Reppenstedt, Germany.

Zusammenfassung

```

    \, \,
    () ()
    () () +-----+
    ( o o ) | Java 2 |
    ^^ ( @_ )_| Platt- |
    || ( )_| form! |
    ++==( )\ \ +-----+
    ( )\ \
    ( ) vv
    ( )
    _//~~\ \
    ( ) ( )
  
```

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™ 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™ –COACH¹ versucht ein solches Verständnis Schritt 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™ –COACH Ihnen helfen Java™ als ein Akronym für *Just a valid application* zu verstehen.

¹**Hinweis:** Dieses Dokument ist ein Entwurf und wird laufend fortgeschrieben. Die aktuelle Version befindet sich unter: <http://as.uni-lueneburg.de/publikation/anwdall.pdf>. Anmerkungen und Kommentare schicken Sie bitte an: <mailto:bonin@uni-lueneburg.de> (**Remark:** This is a **draft document** and continues to be revised. The latest version can be found at <http://as.uni-lueneburg.de/publikation/anwdall.pdf>. Please send comments to <mailto:bonin@uni-lueneburg.de>)

Vorwort

Sie wollen und/oder müssen sich mit der Softwareentwicklung in und mit Java™ befassen? Das ist gut so! Schreiben von Programmen in Java™ macht Freude. Als alter Fan von *List Processing* (LISP) (\leftrightarrow [Bonin91b]) ergeht es mir seit Oktober 1997 ähnlich. Ob nun Java™ 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.

```

    ' '
    () ()
    () ()
    ( . o )
    ( @_ ) -----| Java |
    ( )             | ist  |
                   | gut! |
                   +-----+
// ( ) \\
// ( ) \\
vv ( ) vv
( )
_//~~\\_
( ) ( )

```

Ziel ist es, das breite Spektrum der Java™ 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™-COACH das Modellieren in der Standardsprache *Unified Modeling Language* (UML).

Ohne eine Dokumentation ist ein Programm weder 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* (HTML²) zum Dokumentieren genutzt. Dabei werden Erfahrungen zur Sicherung einer Einheitlichkeit (Links, Namensvergabe, Layoutvorgabe) vermittelt. Die Bausteine und -Konstruktionen werden stets im Wechselspiel zur Fachwelt (\approx Anforderungen) und zur Dokumentation verdeutlicht. Der JAVA™-COACH umfaßt daher folgende Aspekte:

- UML**
- Von unscharfen Vorgaben zum fachlichen Modell in Form von Klassendiagrammen mit Darstellung der Beziehungen zwischen den Objekten (Vererbung, Assoziation, Aggregation und Komposition).
 - Umsetzung des fachlichen Modells in Java-Bausteine

²Präzise formuliert: XHTML Version 1.0 (Exensible Hypertext Markup Language — Eine Spezifikation von HTML 4.0 in XML —) \leftrightarrow <http://www.w3.org/TR/2000/REC-xhtml1-2000126/> (Zugriff: 23-Oct-2001)

- 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™ –COACH wendet sich an alle, die auf einer fundierten Theoriebasis Schritt für Schritt anhand von praxisnahen Beispielen Java™ 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™ –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™ –COACH vermittelt mehr als das üblichen Java-Einführungsbuch mit vielen Web-Seiten-Beispielen. Er befaßt sich eingehend mit mächtigen Konstruktionen, die in klassischen Sprachen³, 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 Poet⁴. Vertieft werden Konzeptionen und Möglichkeiten von Komponentenmodellen. Dazu werden *JavaBeans™* und *EJB (Enterprise JavaBeans™)* erklärt.

Der JAVA™ –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™ 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™ –COACH vorab die „UML-Boxologie“ ehe das Buch von Quellcodebeispielen bestimmt wird.

Der JAVA™ –COACH wurde 1997 begonnen und in einigen Etappen den jeweiligen Java™ Entwicklungen angepasst. Im Jahr 1997 war die Welt noch recht einfach. Einen *Component Transaction Monitor* in der Form von *EJB* kannte das damals verfügbare *Java Development Kit* (JDK 1.0) noch nicht. Im Dezember 2002 wurde dieses Komponentenmodell der Java™ 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

1997:

JDK

1 . x

2002:

J2EE

³Exemplarisch ist hier die Programmiersprache COBOL zu nennen.

⁴↔ <http://www.fastobjects.de> (Zugriff 18.12.2001)

⁵Dieses Beispiel befindet sich im JAVA™ –COACH erst im Abschnitt 5.1.1 S. 76.

die Unzulänglichkeiten Schritt für Schritt weiter zu verringern.

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 „Artefakten“⁶ eines Softwaresystems. Der UML-Standard wird von der OMG⁷ betreut \leftrightarrow Abschnitt 3.7 S. 62. 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. \leftrightarrow [Broy/Siedersleben02] S. 5).

Für die Notation von („maschinennahen“) Modellen beziehungsweise Algorithmen wird auch im Text JavaTM 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 XHTML⁸-Konstrukte verwendet — soweit möglich. Zum Beispiel generiert `java-doc` noch kein valides XHTML. Die Layout-Spezifikation erfolgt mit Hilfe von *Cascading Style Sheets* (CSS).

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 Aufmunterung und zum Schmunzeln im Text *Internet Smileys*

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

⁷OMG \equiv Object Management Group

⁸XHTML \equiv Extensible Hypertext Markup Language

UML

XHTML

- : -) 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: Internet Smileys

benutzt. Ihre Bedeutung erläutert Tabelle 1 S. 7.

Hinweis:

Literaturangaben zum Vertiefen des Stoffes dieses Manuskripts sind vor grauem Hintergrund ausgewiesen.

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™ –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, 5. Oktober 1997 – 19. Juli 2007

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


Inhaltsverzeichnis

1	Java™ -Training — Mehr als Web-Seiten entwickeln	17
1.1	J2SE	19
1.2	J2EE	20
1.3	APIs & Standarddienste	22
2	Eine Welt voller Objekte	25
2.1	Denkwelt der Objekt-Orientierung	26
2.2	Wurzeln der Objekt-Orientierung	29
2.2.1	Polymorphismus	30
2.2.2	Daten-gesteuerte Programmierung	31
2.2.3	Muster-gesteuerter Prozeduraufruf	31
2.3	Ausrichtung objekt-orientierter Sprachen	32
3	Modellieren mit UML	41
3.1	UML-Boxologie	42
3.2	Basiselement: Klasse	45
3.2.1	Beschreibung	45
3.2.2	Paket von Elementen	49
3.3	Beziehungselement: Assoziation	49
3.3.1	Beschreibung	49
3.3.2	Multiplizität	50
3.3.3	Referentielle Integrität	51
3.3.4	Schlüsselangabe	54
3.4	Beziehungselemente: Ganzes ⇔ Teile	55
3.4.1	Aggregation	55
3.4.2	Komposition	56
3.5	Beziehungselement: Vererbung	58
3.5.1	Vererbung	58
3.5.2	Randbedingungen (<i>Constraints</i>)	58

3.6	Pragmatische UML-Namenskonventionen	60
3.7	OMG & UML	62
4	Java™ ≈ mobiles Code-System	65
4.1	Java™ im Netz	66
4.2	Bytecode: Portabilität ⇔ Effizienz	68
4.3	Sicherheit	70
4.3.1	Prüfung des Bytecodes (<i>Bytecode Verifier</i>)	70
4.3.2	Traue Niemandem!	71
4.4	The Road To Java	72
5	Konstrukte (Bausteine zum Programmieren)	75
5.1	Einige Java-Kostproben	76
5.1.1	Kostprobe HelloWorld	76
5.1.2	Kostprobe Foo — Parameterübergabe der Applikation	81
5.1.3	Kostprobe FahrzeugApp — Konstruktor	85
5.1.4	Kostprobe Counter — Eingabe von Konsole	95
5.1.5	Kostprobe Essen — Eingabe von Konsole	98
5.1.6	Kostprobe Ei & Huhn — Compilieren	101
5.1.7	Kostprobe MyNetProg — Internetzugriff	105
5.1.8	Kostprobe ImpulseGenerator — Thread	114
5.1.9	Kostprobe TypApplication — Interface	115
5.1.10	Kostprobe AbClassApplication — Abstrakte Klasse	119
5.1.11	Kostprobe ActionApplet — GUI	122
5.2	Applet-Einbindung in ein Dokument	127
5.2.1	Applet ⇔ Applikation	127
5.2.2	HTML-Marken: <object> und <applet>	128
5.2.3	Beispiel PruefeApplet.html	132
5.2.4	Beispiel CounterApplet.html	134
5.2.5	Beispiel MyProgressBar.html	139
5.3	Syntax & Semantik & Pragmatik	144
5.3.1	Attribute für Klasse, Schnittstelle, Variable und Methode	144
5.3.2	Erreichbarkeit bei Klasse, Schnittstelle, Variable und Methode	149
5.3.3	Operator — Priorität und Assoziativität	152

6 Konstruktionen (Analyse und Synthese)	155
6.1 Nebenläufigkeit (<i>Multithreading</i>)	157
6.1.1 Unterbrechung (<i>sleep</i>)	166
6.1.2 Synchronisation (<i>wait()</i> , <i>notify()</i> , <i>synchronized</i> , <i>join</i>)	166
6.2 Ereignisbehandlung (Delegationsmodell)	167
6.2.1 <i>ActionListener</i> — Beispiel <i>SetFarbe</i>	170
6.2.2 <i>Event</i> → <i>Listener</i> → <i>Method</i>	175
6.2.3 <i>KeyListener</i> — Beispiel <i>ZeigeTastenwert</i>	176
6.3 Persistente Objekte	182
6.3.1 <i>Serialization</i> — Beispiel <i>PersButton</i>	186
6.3.2 Rekonstruktion — Beispiel <i>UseButton</i>	188
6.3.3 <i>JAR (Java Archiv)</i>	190
6.4 Geschachtelte Klassen (<i>Inner Classes</i>)	195
6.4.1 Beispiel <i>Aussen</i>	209
6.4.2 Beispiel <i>BlinkLicht</i>	211
6.5 Internen einer Klasse (<i>Reflection</i>)	216
6.5.1 <i>.class</i> -Datei laden und analysieren	217
6.5.2 Methode aufrufen	225
6.6 Referenzen & <i>Cloning</i>	228
6.7 Integration eines ODBMS — Beispiel <i>FastObjects</i>	233
6.7.1 Transaktions-Modell	233
6.7.2 Speichern von Objekten mittels Namen	234
6.7.3 Referenzierung & Persistenz	235
6.7.4 <i>Collections</i>	236
6.7.5 <i>Extent</i>	237
6.7.6 Transientes Objekt & <i>Constraints</i>	237
6.7.7 Objekt <i>Resolution</i>	238
6.7.8 Abfragesprache (<i>OQL</i>)	241
6.7.9 <i>Enhancer ptj</i>	242
6.7.10 Beispiel: <i>Bind</i> , <i>Lookup</i> und <i>Delete</i>	243
6.8 Zusicherung über Werte	249
6.9 Applikation mit großem Speicherbedarf	252
6.10 Verteilte Objekte	257
6.10.1 Beispiel <i>Stub & Skeleton</i>	257
6.10.2 Beispiel <i>RMI</i>	265
6.11 XML-Daten aggregieren	282
6.12 Komposition mittels <i>Interface</i> -Konstruktion	298

6.13	Komponentenmodelle	312
6.13.1	JavaBeans™	313
6.13.2	EJB (<i>Enterprise JavaBeans™</i>)	319
7	Konstruktionsempfehlungen	331
7.1	Einsatz einer teamfähigen IDE	333
7.1.1	Eclipse — Überblick	334
7.1.2	Eclipse — Edieren	336
7.1.3	Eclipse — Fehleranalyse	336
7.1.4	Eclipse — CVS (Concurrent Versions System)	338
7.1.5	Eclipse — <i>Refactoring</i>	342
7.2	Einsatz von speziellen Werkzeugen	342
7.2.1	JUnit — <i>Unit Tests</i>	342
7.2.2	Ant — <i>Build Tool</i>	348
7.2.3	Logging — <code>java.util.logging</code>	351
7.3	Konventionen zur Transparenz	355
7.3.1	Code-Konventionen	355
7.3.2	Tipps zur Kodierung	361
7.3.3	Rahmen für Geschäftsobjekte und -prozesse	378
8	Dokumentieren mit HTML	381
8.1	XHTML	382
8.2	Cascading Style Sheets (CSS)	384
8.2.1	CSS-Konstrukte	385
8.2.2	HTML-Dokument ⇔ CSS	386
8.2.3	Gruppierung & Vererbung	386
8.2.4	Selektor: <code>class</code> & <code>id</code>	388
8.2.5	Kontextabhängige Selektoren	389
8.2.6	Kommentare im CSS	390
8.2.7	Pseudo-Konstrukte (<code>a:link</code> , <code>p:first-letter</code> , usw.)	390
8.2.8	Die Cascade & Konflikte	391
8.2.9	CSS-Beispiel	393
9	Java™ — OO-Anspruch und OO-Wirklichkeit	399
9.1	OO-Paradigma — unvollständige Umsetzung	400
9.2	Strikte Objekt-Orientierung	401
10	Java™ N Plattform: Hoffnungen & Visionen	403

A	Übungen	405
A.1	Modellierung einer Stückliste	405
A.1.1	Klassendiagramm für die Montagesicht	406
A.1.2	Diagrammerweiterung um den Montageplatz	406
A.2	Klassendiagramm für mehr Transparenz	406
A.2.1	Klassendiagramm notieren	408
A.2.2	Diagrammergänzung um zusätzlichen Aspekt	408
A.3	Shell-Kommando „echo“ programmieren	408
A.3.1	Abbildung als Applikation	408
A.3.2	Unterschiede zum Shell-Kommando	408
A.4	Applikation Wert	409
A.5	Applikation Scoping	410
A.6	Applikation Controlling	410
A.7	Applikation Iteration	412
A.8	Applikation LinieProg	413
A.9	Applikation Inheritance	416
A.10	Applikation TableProg	420
A.11	Applikation Rekursion	422
A.12	Applikation Durchschnitt mit HashMap	425
A.13	Assoziation: Foo ↔ Bar	430
A.14	Gleichnamige Attributen: SlotI	432
A.15	Applikation QueueProg — Fall I	434
A.16	Applikation QueueProg — Fall II	438
A.17	Applet SimpleThread	443
A.18	Applet DemoAWT	446
A.19	Innere Klasse	452
A.19.1	Erzeugte Klassen feststellen	454
A.19.2	Vervollständigen des Protokollauszuges	454
A.20	Zwei Main-Methoden	454
A.21	Anonyme Klassen	455
A.22	Konstruktionsalternative	457
A.22.1	Quellcode von <i>Emil Cody</i>	457
A.22.2	Quellcode von <i>Emma Softy</i>	460
A.22.3	Ergebnis feststellen	463
A.22.4	Quellcode kürzen	463
A.22.5	Bewerten der Alternativen	463
A.23	FastObjects-Beispielprogramm Buch	463
A.24	Vererbung	471
A.24.1	Erzeugte Dateien angeben	474

A.24.2	Java-Aufruf angeben	474
A.24.3	Ergebnis des java-Aufrufes angeben	474
A.25	Read-Write-File-Programm schreiben	474
A.25.1	Ergebnis von java TelefonBuchProg an- geben	478
A.25.2	Programmieren von TelefonLookupProg	478
A.26	Fachsprache verstehen	479
A.26.1	Objektbeziehungen in Java™ abbilden	479
A.26.2	Getter- und Setter-Methoden ergänzen	479
A.27	Paket mit Klassen- & Interface-Dateien notieren	479
A.27.1	Aussagen als Klassendiagramm in UML-Notation abbilden	480
A.27.2	Aussagen in Java-Quellcode abbilden	480
A.27.3	Aufruf der Datei K1.java	481
A.28	HTML-Dokument mit CSS	481
A.28.1	Header-Konstrukt interpretieren	482
A.28.2	Hervorhebungsspezifikation	482
A.29	CSS-Datei und <style>-Konstrukt	482
A.29.1	Fehler finden und korrigieren	484
A.29.2	Cascading Style Sheet auswerten	484
A.29.3	Beschreibung einer angezeigten Überschrift	484
A.30	Standardgerechtes Programmieren in Java	484
A.30.1	Beurteilung von Hund	486
A.30.2	Ergebnis von Hund	486
A.30.3	Reengineering von Hund	487
A.31	Side Effect bei Objekt-Orientierung	487
A.32	XML-Daten bearbeiten	489
A.32.1	DTD aufstellen	491
A.32.2	XML-Datei erzeugen	491
A.32.3	XML-Datei visualisieren	491
B	Lösungen zu den Übungen	493
C	Hinweise zur Nutzung von J2SE SDK	575
C.1	Java™ auf der AIX-Plattform	575
C.2	Java™ auf der Windows-Plattform	577
D	Quellen	579
D.1	Literaturverzeichnis	579
D.2	Web-Quellen	586

D.3	Anmerkungen zum JAVA™ –COACH	589
D.4	Abkürzungen und Akronyme	589
E	Index	607

Java-Coach

Java-Coach

Kapitel 1

Java™ -Training — Mehr als Web-Seiten entwickeln

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 objekt-orientierten 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™ –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™ als ein Akronym für *Just a valid application* betrachten können.

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



Legende: Quelle ↔ [ITS97], Seite 134.

Abbildung 1.1: Java — wunderschöne Insel Indonesiens

Trainingsplan

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

- die Java 2 Standard Edition (J2SE)
↪ Seite 19 ...
 - die Java 2 Enterprise Edition (J2EE) und
↪ Seite 20 ...
 - einige Application Programming Interfaces (APIs) und Standard-
dienste der Java 2 Plattform.
↪ Seite 22 ...
-

<p>Java™ ist eine:</p> <ul style="list-style-type: none"> • „einfache, • objektorientierte, • verteilte, • interpretierte, • robuste, • sichere, • architektur-unabhängige, • portable, • hochperformante, • multithread-fähige und • dynamische Sprache“ [Arnold/Gosling96, JavaSpec].
--

Tabelle 1.1: Java-Beschreibung von Sun Microsystems, Inc. USA

1.1 J2SE

Java ist eine wunderschöne Insel Indonesiens mit exotischer Ausstrahlung und großem Kaffeeanbau. In der Softwarewelt ist Java™ ein Begriff für eine objekt-orientierte Softwareentwicklung. Als das Projektteam um Bill Joy und James Gosling (Sun Microsystems, Inc. USA) Java™ konzipierten ging es zunächst um die Steuerung von Haushaltsgeräten und danach um dynamische Web²-Seiten und um eine allgemeine Programmiersprache mit vielen guten Eigenschaften (↔ Tabelle 1.1 S. 19). In dieser Zeit wurde Java™ 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.

Kontinuierlich reifte Java™ 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 (Ser-

B. Joy

J. Gosling

²Web ≡ *World Wide Web*

ver) bis hin zum Großrechner (*Host*) ist Java™ 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™ umgetaufte wurde [Hist97], ist eine ganze Java-Technologie entstanden. Die derzeitige Basisplattform zur Entwicklung für betriebswirtschaftlich relevante Software, also beispielsweise für sogenannte *Enterprise Information 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 "1.5.0_08"`

JRE Java 2 Runtime Environment, Standard Edition

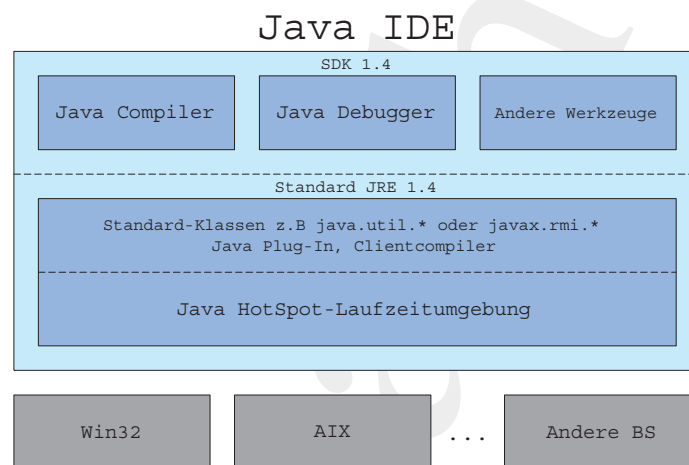
Zum Beispiel: `Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_08-b03) Java HotSpot(TM) Client VM (build 1.5.0_08-b03, mixed mode, sharing)`

J2SE SDK, früher als JDK bezeichnet, bietet die Sprachfunktionalität von Java™ und die Kernbibliotheken, die für die „übliche“ Anwendungsentwicklungen notwendig sind (↔ Abbildung 1.2 S. 21). Diese Kernklassen sind in den Paketen `java.*` enthalten. Zusätzlich bietet J2SE SDK Bibliotheken für Erweiterungen. Diese stehen als `javax.*`-Pakete zur Verfügung (↔ Tabelle 1.2 S. 21).

1.2 J2EE

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

- 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



Legende:

IDE *I*ntegrated *D*evelopment *E*nvironment

JRE *J*ava *R*untime *E*nvironment

SDK *S*oftware *D*eveloper's *K*it

Abbildung 1.2: J2SE: Skizze der Komponenten

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

Legende:

Grober Überblick über java version "1.4.0_01"

Tabelle 1.2: J2SE: Pakete und ihre Aufgabe

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 Management Systems (DBMS), das heißt die Datenbanken, Daten und externe Dienste. Sie gewährleistet die Persistenz von Daten.

1.3 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 Registrieren und Suchen von (Geschäfts-)Komponenten. Dazu dienen das Lightweight Directory Access Pro-

tol (LDAP), der CORBA³ 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 JavaTM geschrieben sein müssen, zu kommunizieren.

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

³CORBA ≡ Common Object Request Broker Architecture

Java-Coach

Kapitel 2

Eine Welt voller Objekte

Die objekt-orientierte Denkwelt von Java™ 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 26 ...
 - die Wurzeln der Objekt-Orientierung und
↪ Seite 29 ...
 - die Ausrichtung von objekt-orientierten Programmiersprachen.
↪ Seite 32 ...
-

2.1 Denkwelt der Objekt-Orientierung

Objekt-Orientierung¹ 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),
3. *Durchschaubarkeit & Wartbarkeit*,
das heißt, das Programm kann auch von anderen Programmierern als dem Autor verstanden, verbessert und auf geänderte Verhältnisse eingestellt werden,

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

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 Paradigma³ (\approx 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 (\leftrightarrow [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:

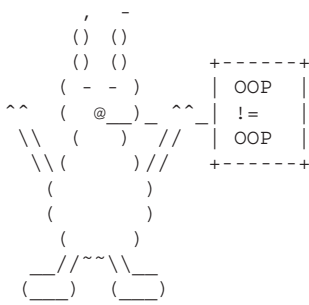
Ein „Objekt“ der realen (oder erdachten) Welt bleibt stets erhalten. Es ist über die verschiedenen Abstraktionsebenen leicht verfolgbar. Das gewachsene Verständnis über die Objekte der realen Welt verursacht eine größere Modelltransparenz.

²Das 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.

³Ein 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 \leftrightarrow [Bonin88]).

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 (\approx 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 Nachrichtenaustausch⁴ 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 (\equiv gerichteten Kanten eines Graphen). Die gemalten Kästchen (\equiv 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 Zusammenhang mit anderen Modellierungstechniken. Die aus dem griechischen stammende Nachsilbe ... *logie* bedeutet *Lehre, Kunde, Wissenschaft* wie sie zum Beispiel beim Wort Ethnologie (\equiv Völkerkunde) zum Ausdruck kommt. Boxologie ist einerseits spassig-provokativ gemeint. Andererseits soll damit die Wichtigkeit aussagekräftiger Zeich-

⁴engl. message passing

⁵Angemerkt 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.

nungen hervorgehoben werden.

Im imperativen Paradigma umfaßt die Boxologie zum Beispiel Ablaufpläne oder Struktogramme. Geht es um nicht-prozedurale Sachverhalte (zum Beispiel 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:

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

Diagramme

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 2.3 S. 35). Als Kriterien dienen einerseits die prognostizierte Menge der Objekte und andererseits ihre Komplexität und Struktur.

2.2 Wurzeln der Objekt-Orientierung

Java™, 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“)

⁶Nur 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 9 S. 399.

3. Muster-gesteuerter Prozeduraufruf⁷

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

2.2.1 Polymorphismus

Ein polymorpher Operator ist ein Operator, dessen Verhalten oder Implementation abhängt von der Beschreibung seiner Argumente.⁸ Java™ 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.

[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);
...
```

⁷engl. pattern directed procedure invocation

⁸Schon 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, \dots, 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™ oder C++ (\leftrightarrow [Stroustrup86, Stroustrup89]) möglich.

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™ unterstützt den Compilierungszeit-Polymorphismus. Smalltalk ist beispielsweise ein Vertreter des Laufzeit-Polymorphismus.

Zeitpunkt

2.2.2 Daten-gesteuerte Programmierung

Daten-gesteuerte Programmierung ist keine leere Worthülse oder kein Pleonasmus⁹, 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 (\equiv anwendungsspezifischen Daten) mit aktiven Systemkomponenten (\equiv 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 (\leftrightarrow [Abelson85]). Der Programmierer definiert selbst eine *dispatch*-Funktion¹¹, die den Zugang zu den datentypabhängigen 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 (\leftrightarrow [Bonin91b]).

2.2.3 Muster-gesteuerter Prozeduraufruf

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*,

**Ziel:
Paßt!**

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

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

¹¹engl. dispatch \equiv Abfertigung

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

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 unbekanntem Ziel. Anhand der Kontrollstruktur kann die Verknüpfung von Daten und Prozedur mittels der Technik des Mustervergleichs¹³ als Vorwärtsverkettung¹⁴ und/oder als Rückwärtsverkettung¹⁵ erfolgen.

Aus der Tradition des muster-gesteuerten Prozeduraufrufs sind Schritt für Schritt leistungsfähige regel- und logik-orientierte Sprachen mit Rückwärtsverkettung¹⁶, wie zum Beispiel PROLOG¹⁷, entstanden.

2.3 Ausrichtung objekt-orientierter Sprachen

Bei einem groben, holzschnittartigen Bild von objekt-orientierten Sprachen ist die Zusammenfassung der passiven Komponente (\equiv Daten) mit der aktiven Komponente (\equiv 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:

Objekt-Orientierung \equiv Datentypen
+ Nachrichtenaustausch

Das Bild von Objekten, die miteinander Nachrichten austauschen, ver-

¹³engl. „pattern matching“

¹⁴engl. „forward chaining“

¹⁵engl. „backward chaining“

¹⁶engl. backward chaining rule languages

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

passiv

aktiv

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

Legende:

Actor-Sprachen ↔ [Lieberman81]

C++ ↔ [Stroustrup86, Stroustrup89]

CLOS (Common Lisp Object System) ↔ [Bobrow/Moon88]

Daten-gesteuerte Programmierung ↔ [Abelson85]

Java™ ↔ [Arnold/Gosling96]

SELF ↔ [Ungar/Smith91]

Smalltalk ↔ [Goldberg/Robson83, Goldberg83]

PROLOG (*PRO*gramming in *LOG*ic) ↔ [Belli88, Clocksin/Mellish87]

Tabelle 2.1: Ausrichtung von objekt-orientierten Ansätzen

deutlicht 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.

```

      ' '
      () ()
      () () +-----+
      ( o . ) | Klasse |
      ( @ _ ) | oder   |
      (   )   | P.-Typ! |
      //(   )\\ +-----+
      //(-----)\\
vv (-----) vv
      (   )
      _//~\\_
      (___) (___)

```

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 2.1 S. 33 zeigt die vier skizzierten Schwerpunkte als die zwei Dimensionen: Klassen / Prototyp und Operation / Nachrichtenversand (ähnlich [Gabriel91]). Eine schlagwortartige Übersicht (\leftrightarrow Tabelle 2.2 S. 36) vergleicht JavaTM 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 (\equiv Klassen) von „realen“ Einheiten (\equiv Objekten) beschreiben. Da Objektklassen von anderen Klassen Eigenschaften (interne Variablen, Methoden) *erben* können, lassen sich aus allgemeinen Objekten spezielle anwendungsspezifische konstruieren. Vererbungsgraphen, gezeichnet mit beschrifteten Kästchen, Kreisen und Pfeilen, bilden dabei das Modell ab. (Ein Beispiel zeigt \leftrightarrow Abbildung 3.11 S. 61.) 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 JavaTM 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.

Da auch der Programmcode objekt-orientiert ist, ist die erreichbare (Laufzeit-)Effizienz ebenfalls ein Entscheidungskriterium. Müssen wir mangels heute schon bewährter objekt-orientierter Datenbank-Mana-

Effizienz

Terminologie (Leistungen) bei der Objekt-Orientierung				
Aspekte ¹		I Java	II Smalltalk	III CLOS
1	Abstraktion	class		
2	Objekt	member of class	instance	
3	Vererbung	superclass		direct superclasses
		subclass		
4	Objektstruktur	member data elements	instance variables	slots
5	Strukturbeschreibung (Slot)	name		
		type initial value forms		initial value forms, accessor functions, initializer keywords
6	Operationsaufruf	calling a method	sending a message	calling a generic function
7	Operationsimplemmentation	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	Direktzugriff auf internen 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	Operationskombination	Nein		standard method combination

Legende:¹ ähnlich [Kczales91] S. 253.

Tabelle 2.2: Java im Vergleich mit Smalltalk und CLOS

Aspekte für die Beurteilung von OO-Anwendungsfelder				
„Daten“-Struktur			„Daten“-Menge (Datenvolumen)	
		<i>gleichartige Fälle</i>	I <i>relativ beschränkt</i> (ladbar in Arbeitsspeicher)	II <i>sehr groß</i> (DBMS erforderlich)
<i>einfach</i>	A ₁	wenige	[0] OO → Effizienzverlust	[-] Objekt ↔ Datensatz Konvertierung
	A ₂	viele		
<i>komplex</i>	B ₁	wenige	[+] OO plus XPS	[-] ohne OO-DBMS nicht machbar
	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 2.3: Aspekte Menge und Struktur

gementsysteme erst die Objekte aus den Daten bei jedem Datenbankzugriff erzeugen und beim Zurückschreiben wieder in Tabellen (Daten) konvertieren, dann entstehen zumindest bei komplex strukturierten Objekten Laufzeitprobleme. Die Objektmenge und die Komplexität der Objektstruktur sind relevante Kriterien, um sich für die Objekt-Orientierung entscheiden zu können (Tabelle 2.3 S. 37). 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.

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 diese 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 (\leftrightarrow Abschnitt 2.2.1 S. 30) ganz verzichten. Nicht nur der Konvertierungsaufwand ist offensichtlich abhängig von der Komplexität der Objektstruktur, sondern auch die Konstruktion und Abarbeitung eines Vererbungsgraphens.

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

Rüstzeit

**Stück-
Kosten**

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.

Java-Coach

Java-Coach

Kapitel 3

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 „Artefakten 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 42 ...
 - die verschiedenen Typen von Klassen mit ihren Variablen und Methoden,
↪ Seite 45 ...
 - die Verbindung zwischen Klassen in Form der Assoziation,
↪ Seite 49 ...
 - die Beziehungen zwischen dem Ganzen und seinen (Einzel-)Teilen,
↪ Seite 55 ...
 - die Vererbung von Eigenschaften (Variablen und Methoden),
↪ Seite 58 ...
 - gibt Empfehlungen für die Namensvergabe für UML-Elemente und
↪ Seite 60 ...
 - skizziert die Weiterentwicklung von UML.
↪ Seite 62 ...
-

3.1 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]

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

Rational

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

von Artefakten eines Softwaresystems. UML ist besonders gut geeignet für die Modellierung von Geschäftsvorgängen (*business modeling*). Zunehmend gewinnt 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 2.x (↔ Abschnitt 3.7 S. 62) eine Menge von Diagrammen. Die Tabelle 3.1 S. 44 klassifiziert die Diagramme mit ihren üblichen Ausprägungen.¹ Die UML-Diagramme ermöglichen ver-

**Dia-
gram-
me**

¹Hinweis: Der UML-Standard definiert die UML-Modelle und das hinter den Diagrammen stehende *Repository*. Diagramme, also spezielle Sichten auf das *Repository*, können relativ frei gestaltet werden.

UML-Diagramme
<ul style="list-style-type: none"> • <i>Strukturdiagramme</i> <ul style="list-style-type: none"> – Klassendiagramm Zeigt Klassen (\leftrightarrow Abschnitt 3.2 S. 45) und ihre Beziehungen (\leftrightarrow Abschnitt 3.3 S. 49 bis 3.5 S. 58) untereinander. – Objektdiagramm Zeigt Objekte mit beispielhaften Werten (Inhalten). – Anwendungsfalldiagramm Zeigt Akteure, Anwendungsfälle und ihre Beziehungen zueinander. – Paketdiagramm Zeigt Ablagestruktur und Abhängigkeiten der Modellelemente. • <i>Architekturdiagramme</i> <ul style="list-style-type: none"> – Kompositionsstrukturdiagramm Zusammensetzung und Schnittstellengruppierung von Komponenten. – Komponentendiagramm Zeigt Komponenten und ihre Verbindungen untereinander (fachliche Architektur). – Subsystemdiagramm Zeigt architektonische Zusammenhänge von Komponenten und Ähnlichem. – Einsatz- und Verteilungsdiagramm Zeigt Artefakte und ihre Verteilung auf Knoten und Komponenten. • <i>Verhaltensdiagramme</i> <ul style="list-style-type: none"> – Aktivitätsdiagramm Beschreibt Abläufe mit Hilfe von Aktionen, Transitionen und Verzweigungen. – Zustandsdiagramm Beschreibt Objektzustände und mögliche Zustandsübergänge. <ul style="list-style-type: none"> * Protokollautomat Beschreibt zulässige Reihenfolgen von Operationen. – <i>Interaktionsdiagramme</i> <ul style="list-style-type: none"> * Sequenzdiagramm Zeigt zeitlich geordnet den Nachrichtenaustausch zwischen Objekten. * Kommunikationsdiagramm Zeigt topologisch geordnet den Nachrichtenaustausch zwischen Objekten. * Zeitdiagramm Beschreibt zeitliche Bedingungen im Kontext von Objektzuständen. * Interaktionsübersicht Kombination von Sequenz- und Aktivitätsdiagramm.

Legende: In UML können Diagramme, also spezielle Sichten auf das spezifizierte *Repository*, relativ frei gestaltet werden (zum Beispiel \leftrightarrow [Oestereich06] S. 207–345).

Tabelle 3.1: Klassifikation von UML-Diagrammen

schiedene 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.

3.2 Basiselement: Klasse

3.2.1 Beschreibung

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 3.2 S. 46 zeigt die UML-Notation einer Klasse. `class`

[Hinweis: Ein verwandter Begriffe für die Klasse ist der Begriff (Daten-)Typ. In Java™ ist ein einfacher Datentypen wie zum Beispiel `float` von einem „zusammengesetzten“ Datentype (*ReferenceType*) wie zum Beispiel `FahrzeugApp` (\leftrightarrow Abschnitt 5.1.3 S. 85) 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*), also keine Instanzen (Objektexemplare). Sie wird als eine (normale) Klasse mit dem Merkmal `{abstract}` notiert².

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)

Ein Variable benennt einen Speicherplatz in einer Instanz (\equiv Instanzva- **Variable**

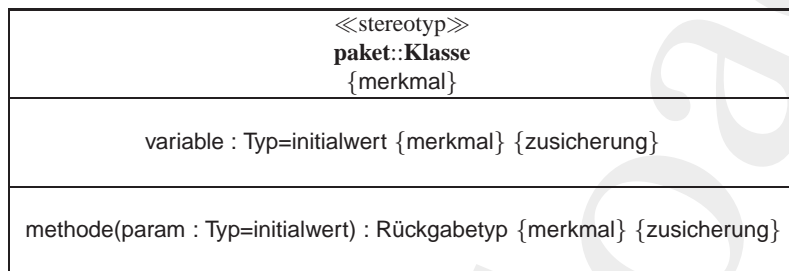
²Alternativ zu dieser Kennzeichnung kann der Klassenname auch *kursiv* dargestellt werden.

³Näheres zur Programmierung mit Metaobjekten \leftrightarrow [Kczales91].

Klassensymbol:



Klassensymbol mit Variablen und Methoden:



Beispiel: Fahrzeug (↔ Abschnitt 5.1.3 S. 85)

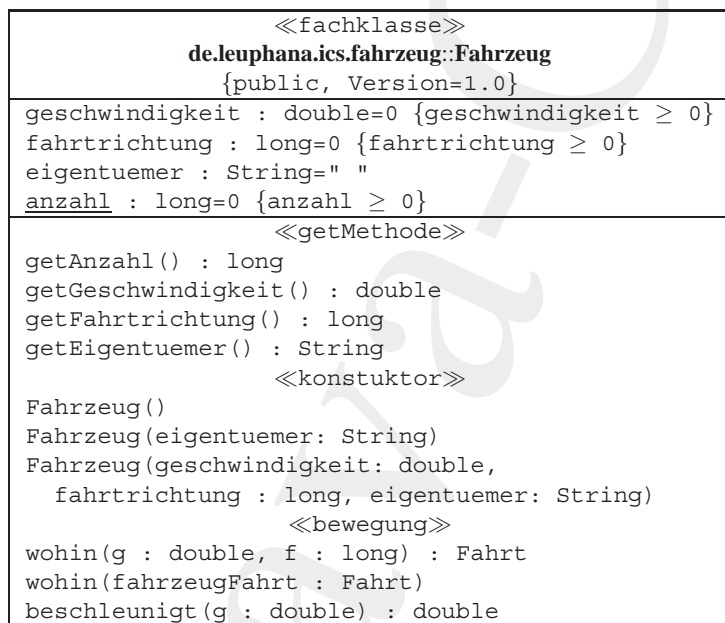


Tabelle 3.2: UML-Basiselement: Klasse

UML-Notation	Erläuterung
+publicVariable	allgemein zugreifbare Variable
–privateVariable	private, nicht allgemein zugreifbare Variable
#protectedVariable	geschützte, bedingt zugreifbare Variable
/abgeleiteteVariable	hat einen Wert aus anderen Variablen
<u>klassenVariable</u>	Variable einer Klasse (<i>static</i>)

Hinweis: Sichtbarkeit (Zugriffsrechte) im Java-Kontext \leftrightarrow Tabelle 5.7 S. 150.

Tabelle 3.3: Kennzeichnung einer Variablen in UML

riable) oder in der Klasse selbst (\equiv 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 3.3 S. 47 zeigt diese Möglichkeiten einer Kennzeichnung.

Beschreibung: Methode

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

Methode

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

Beispiel:

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

UML-Notation	Erläuterung
+publicMethode()	allgemein zugreifbare Methode
–privateMethode()	private, nicht allgemein zugreifbare Methode
#protectedMethode()	geschützte, bedingt zugreifbare Methode
<u>klassenMethode()</u>	Methode einer Klasse (<i>static</i>)

Hinweis:

Sichtbarkeit (Zugriffsrechte) im Java-Kontext ↔ Tabelle 5.7 S. 150.

Tabelle 3.4: Kennzeichnung einer Methode in UML

[**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 3.4 S. 48 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}

**Zu-
sicher-
ung**

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

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.]

Stereotyp**3.2.2 Paket von Elementen**

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, dann steht der Paketname auf der Aktenregisterlasche.

Beispiel: de.leuphana.ics.fahrzeug

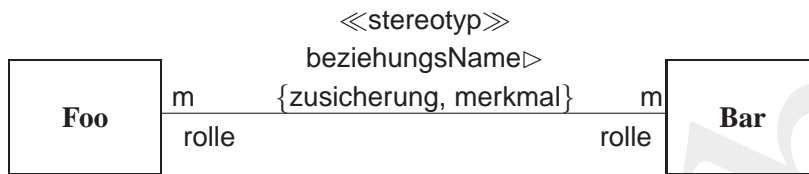
[Hinweis: Verwandte Begriffe für das Paket sind die Begriffe Klassenkategorie, Subsystem und *Package*.]

Paket**3.3 Beziehungselement: Assoziation****3.3.1 Beschreibung**

Eine Assoziation beschreibt eine Verbindung zwischen Klassen (\leftrightarrow Abbildung 3.1 S. 50). 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 (\leftrightarrow Abbildung 3.2 S. 51). Jedoch kann eine Assoziation auch von rekursiver Art sein, das heißt beispielsweise als Beziehung zwischen zwei Instanzen derselben Klasse formuliert werden (Beispiel

Legende:

beziehungsName	Name der Assoziation
{zusicherung}	↔ Abschnitt 3.2.1 S. 48
{merkmal}	↔ Abschnitt 3.2.1 S. 48
m	Multiplizität (↔ Tabelle 3.5 S. 52)
rolle	Sichweise durch das gegenüberliegende Objekt
\triangleright	(Lese)-Richtung für die Beziehung; hier: FoobeziehungsNameBar

Abbildung 3.1: UML-Beziehungselement: Assoziation

↔ Abbildung 3.3 S. 52) oder eine Assoziation zwischen mehreren Klassen sein. Spezielle Formen der Assoziation sind die Aggregation (↔ Abschnitt 3.4.1 S. 55) und die Komposition (↔ Abschnitt 3.4.2 S. 56).

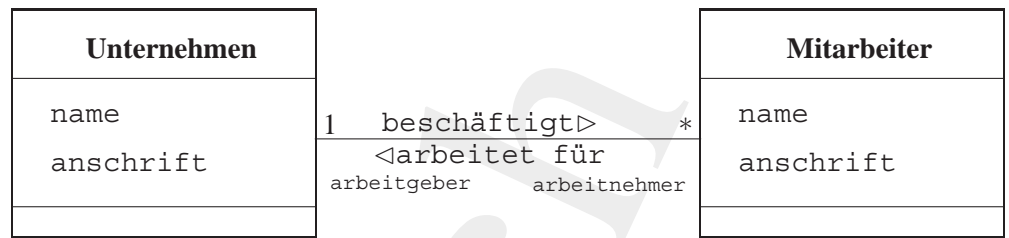
[**Hinweis:** Verwandte Begriffe für die Assoziation sind die Begriffe Relation, Aggregation, Komposition, Link und Objektverbindung.]

3.3.2 Multiplizität

Die Multiplizität m gibt an mit wievielen Instanzen der gegenüberliegenden Klasse **Bar** eine Instanz der Klasse **Foo** assoziiert ist⁴. Dabei kann eine Bandbreite durch den Mini- und den Maximumwert angegeben werden (↔ Tabelle 3.5 S. 52).

Ein Assoziation wird in der Regel so implementiert, daß die beteiligten Klassen zusätzlich entsprechende Referenzvariablen bekommen. Im Beispiel „Ein Unternehmen beschäftigt viele Mitarbeiter“ (↔ Abbildung 3.2 S. 51) erhält die Klasse **Unternehmen** die Variable `arbeiter` und die Klasse **Mitarbeiter** die Variable `arbeitgeber`. Aufgrund der angegebenen Multiplizität * („viele Mitarbeiter“) muß die Variable `arbeiter` vom Typ einer Behälterklasse sein, damit sie viele Objekte aufnehmen kann. Ein *Set* (Menge ohne Duplizität) oder ein *Bag*

⁴... beziehungsweise assoziiert sein kann.

Legende:

↔ Abbildung 3.1 S. 50

Abbildung 3.2: Beispiel einer Assoziation: Ein Unternehmen beschäftigt viele Mitarbeiter

(Menge mit Duplizität) sind beispielsweise übliche Behälterklassen.

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

Eine Assoziation kann selbst Variablen haben. Im Beispiel „Ein Unternehmen beschäftigt viele Mitarbeiter“ (↔ Abbildung 3.2 S. 51) kann dies beispielsweise die Historie der Beschäftigungsverhältnisse 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 3.4 S. 53).

3.3.3 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 angegeben:

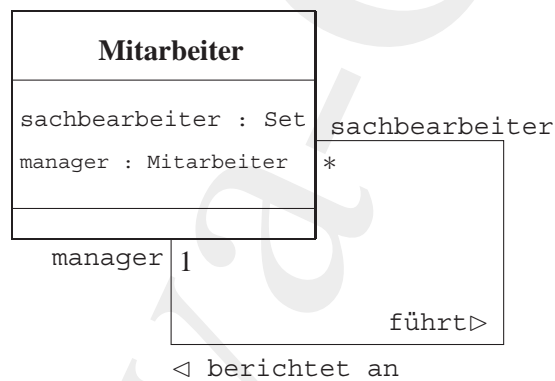
**Integri-
tät**

Multiplizität	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:

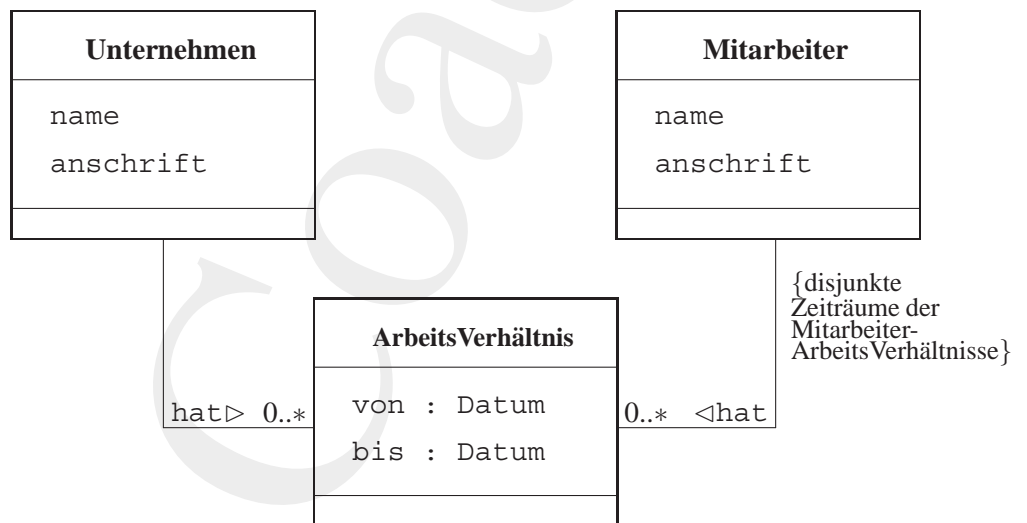
- , Komma ist Trennzeichen für die Aufzählung
- * beliebig viele
- 0 optional

Tabelle 3.5: Angabe der Multiplizität

Legende:

Assoziation ↔ Abbildung 3.1 S. 50

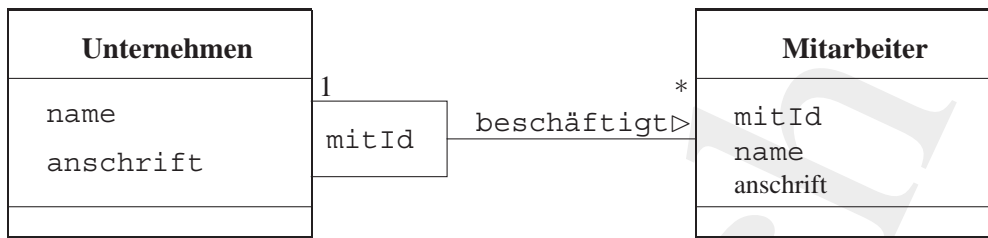
Abbildung 3.3: Beispiel einer direkten rekursiven Assoziation



Legende:

↔ Abbildung 3.2 S. 51

Abbildung 3.4: Beispiel einer Assoziationsklasse: ArbeitsVerhältnis

Legende:

↔ Abbildung 3.2 S. 51

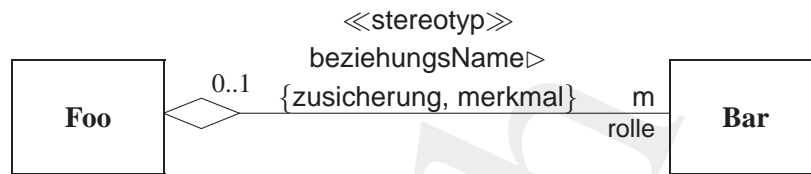
Unternehmen		Mitarbeiter		
name	anschrift	mitId	name	anschrift
Otto KG	Nürnberg	Boe	Boesewicht	Bonn
Otto KG	Nürnberg	Gu	Gutknecht	Lüneburg
Emma AG	Berlin	Fr	Freund	Lüneburg

Abbildung 3.5: Beispiel einer qualifizierenden Assoziation (mitId)

- {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.

3.3.4 Schlüsselangabe

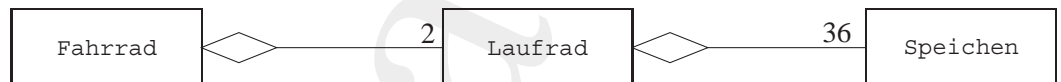
Beziehungen mit der Multiplizität $m = *$ werden in der Regel mittels einer Behälterklasse implementiert (↔ Abschnitt 3.3.2 S. 50). Dabei kann es sinnvoll sein schon im Klassenmodell mit den Assoziationen den Zugriffsschlüssel darzustellen. Ein solcher Schlüssel, auch als qualifizierendes Attribut der 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 3.5 S. 54.



Legende:

↔ Abbildung 3.1 S. 50

Abbildung 3.6: UML-Beziehungselement: Aggregation



Legende:

↔ Abbildung 3.6 S. 55

Abbildung 3.7: Beispiel einer Aggregation: Ein Fahrrad hat zwei Laufräder mit jeweils 36 Speichen

3.4 Beziehungselemente: Ganzes ⇔ Teile

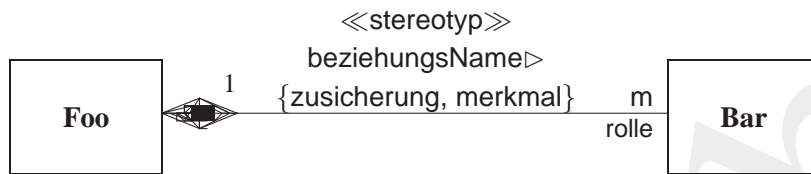
3.4.1 Aggregation

Eine Aggregation beschreibt eine „Ganzes⇔Teile“-Assoziation. (↔ Abbildung 3.6 S. 55) Das Ganze nimmt dabei Aufgaben stellvertretend für seine Teile wahr. Im Unterschied zur normalen Assoziation haben die beteiligten Klassen keine gleichberechtigten Beziehungen. Die Aggregationsklasse hat eine hervorgehobene Rolle und übernimmt die „Koordination“ ihrer Teilklassen. Zur Unterscheidung zwischen Aggregationsklasse und Teilklassen wird die Beziehungslinie durch eine Raute auf der Seite der Aggregationsklasse ergänzt. Die Raute symbolisiert das Behälterobjekt, das die Teile aufnimmt.

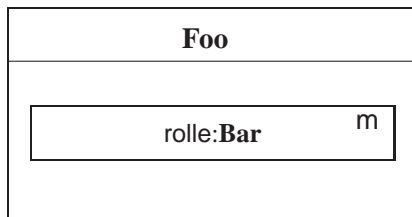
Aggregation

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

Die Abbildung 3.7 S. 55 zeigt den Fall: „Ein Fahrrad hat zwei Laufräder mit jeweils 36 Speichen“. Dieses Beispiel verdeutlicht, daß ein Teil (Laufrad) selbst wieder eine Aggregation sein kann.



Alternative Notation:



Legende:

↔ Abbildungen 3.6 S. 55 und 3.1 S. 50

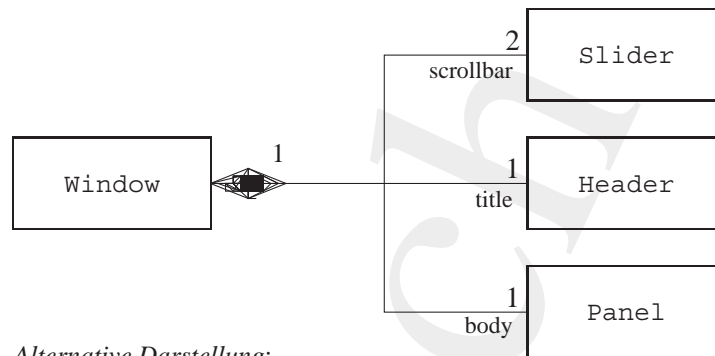
Abbildung 3.8: UML-Beziehungselement: Komposition

3.4.2 Komposition

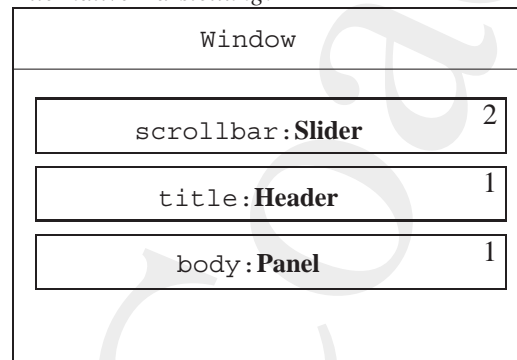
Komposition

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 3.8 S. 56). 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 Teilklassen wie bei der (normalen) Aggregation. Zur Unterscheidung zwischen Kompositionsklasse und Teilklass(e)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 Teilklassen auch direkt in den Kasten der Kompositionsklasse geschrieben werden.

⁵In Java™ also nicht mehr referenzierbar.



Alternative Darstellung:



Legende:

Ein Window besteht aus zwei Slider, einem Header und einem Panel

\leftrightarrow Abbildung 3.6 S. 55 (Beispielidee [Rational97])

Abbildung 3.9: Beispiel einer Komposition: Window mit Slider, Header und Panel

[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. Diese Teile bestehen nicht unabhängig vom Ganzen, dem *Window* (\leftrightarrow Abbildung 3.9 S. 57).

3.5 Beziehungselement: Vererbung

3.5.1 Vererbung

Als Vererbung bezeichnet man einen Mechanismus, der die Eigenschaften (Variablen und Methoden) einer Klasse (\equiv Oberklasse) für eine andere Klasse (\equiv 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 (\leftrightarrow Abbildung 3.10 S. 59). Welche gemeinsamen Eigenschaften von Unterklassen zu einer Oberklasse zusammengefaßt, also generalisiert werden, und umgekehrt, welche 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 Diskriminator 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** \leftrightarrow **Spezialisierung-Relation** ab.

Bei der Modellierung einer Vererbung ist es zweckmäßig den Diskriminator 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 Generalisierung, Spezialisierung und *Inheritance*.]

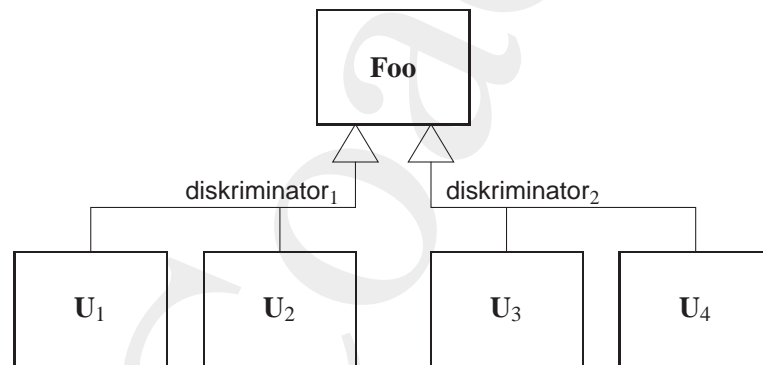
3.5.2 Randbedingungen (*Constraints*)

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

Ver-
erbung

Diskrimi-
nator

Legende:

- \triangle \equiv Kennzeichnung der Oberklasse
Vererbungsrichtung
- diskriminator_j \equiv charakteristisches Gliederungsmerkmal für die
Generalisierungs \leftrightarrow Spezialisierungs-Beziehung
- Foo** \equiv Oberklasse von **U₁**, **U₂**, **U₃** und **U₄**
Eigenschaften von **Foo** sind in **U_i** zugreifbar
- U_i** \equiv **U_i** ist Unterklasse von **Foo**

Abbildung 3.10: UML-Beziehungselement: Vererbung

einer anderen Unterklasse sein.

In dem Beispiel „Schiffe“ (\leftrightarrow Abbildung 3.11 S. 61) könnte ein Objekt Emma-II sowohl ein Objekt der Unterklasse Tanker wie auch der Unterklasse ContainerSchiff sein, wenn `{overlapping}` angegeben wäre.

`disjoint`

- `{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.

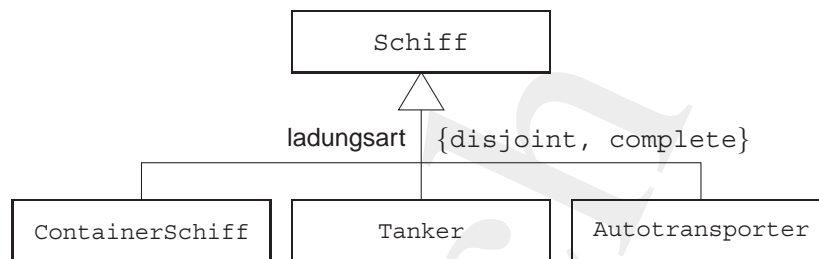
In unserem Schiffsbeispiel könnte also keine Unterklasse Kreuzfahrtschiff auftauchen, weil `{complete}` angegeben ist.

- `{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}` angeben, wenn nur aus zeichnerischen Gründen eine Unterklasse fehlt.

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

3.6 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.

Legende:

↔ Abbildung 3.10 S. 59

Eine Oberklasse `Schiff` vererbt an die Unterklassen `ContainerSchiff`, `Tanker` und `Autotransporter`.

Abbildung 3.11: Beispiel einer Vererbung

Beispiele zum Wechsel der Groß/Kleinschreibung:

```

FahrzeugApp
getGeschwindigkeitFahrrad()
setFahrtrichtung()
  
```

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

- *Paket*: Der Name beginnt mit einem kleinen Buchstaben. **package**
Beispiel: `java.lang`
- *Klasse* oder *Schnittstelle*: Der Name beginnt mit einem Großbuchstaben. **class**
Beispiel: `Fahrzeug`
- *(Daten-)Typ* (zum Beispiel Rückgabetyt): Der Name beginnt in der Regel mit einem Großbuchstaben, weil eine (Daten-)Typangabe eine Klasse benennt. Bei einfachen Java-Datentypen (*Primitive-Type* ↔ Tabelle 5.5 S. 146) 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 JavaTM exakt den Namen der Klasse haben. Deshalb beginnt ein Konstruktor stets mit einem Großbuchstaben. **Variable**
Methode

Beispiel: Fahrzeug(), beschleunigt()

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

3.7 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 standardisierte sie UML 1 als eine umfassende Modellierungssprache, die bewährte Modellierungstechniken integrierte und dies auf der Grundlage einer einheitlichen graphische Notation. Dann arbeitete die OMG an einer wesentlichen Überarbeitung und Fortschreibung des Standards zu einem UML 2. Dabei ging es um Lösungsvorschläge für eine Menge neuer Anforderungen (*52 Requirements*, ↔ [Miller02]). In diesem Kontext wurden von fünf Gruppen *Proposals*⁶ eingereicht (Oktober 2002):

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

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

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

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

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

OMG's *Model Driven Architecture* (MDA) Initiative ist ein Ansatz für Industriestandards, die geprägt sind von der Überzeugung, dass an-

⁶UML 2 Proposals ↔ <http://www.community-ML.org/>

statt der Programme die *primären Artifacts* der Softwareentwicklung zu kreieren sind. Mit Werkzeugen sind die Programme dann mehr oder weniger automatisch generierbar.

Im Jahr 2006 existiert der Standard UML 2.x (mit $x = 1$). Er ist in die folgende Teilbereiche gegliedert:

UML 2.x

- *Infrastructure* (Kern der Spracharchitektur, Profiles, Stereotypen)
- *Superstructure* (Statische und dynamische Modellelemente)
- *Object Constraint Language* (OCL)
- *Diagram Interchange* (UML Diagrammaustauschformat)

Die OMG hat mit dem Standard UML 2.x auch Qualifikationsstandards in Form eines dreistufigen Zertifizierungsprogramms eingeführt. „Dieses Zertifizierungsprogramm soll sicherstellen, das UML-Anwender Trainer, Berater, Werkzeughersteller u. a. ein einheitliches UML-Verständnis und eine Mindestqualifikation aufweisen.“ (↔ [Oestereich06] S. 210)

Java-Coach

Kapitel 4

Java™ \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™ ist jedoch mehr als eine objekt-orientierte Programmiersprache. Java™ ist (fast) ein *mobiles Code-System*.

Java™ 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™ ≈ mobiles Code-System“ erläutert:

- das Zusammenspiel von Java™ und dem Web,
↪ Seite 66 ...
- die Portabilität aufgrund des Bytecodes,
↪ Seite 68 ...
- das Sicherheitskonzept und
↪ Seite 70 ...
- skizziert den Weg zur Softwareentwicklung mit Java.
↪ Seite 72 ...

4.1 Java™ im Netz

```

      \ ( )
      ( ) ( ) +-----+
      ( o o )   | Java |
      ( @_ ) ___| == |
      ( )       | Web! |
      //( ) ==++ +-----+
      //( ) ||
      vv ( ) vv
      ( )
      _//~\_\_
      ( ) ( )
  
```

Java™ 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™ Bytecode laden und diesen direkt ausführen. Über einen vorgegebenen Fensterausschnitt des Browsers kann dann dieses Bytecodeprogramm mit dem Benutzer kommunizieren.

Java™ ist daher auch ein *mobiles Code-System*.¹ Ein solches System

¹Ein 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 \leftrightarrow Server-Interaktionen lässt sich mit folgenden Schritten skizzieren:

1. **Anfordern des Applets** (*request*-Schritt)

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.

Request

2. **Empfangen des Applets** (*download*-Schritt)

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.

Download

3. **Laden und ausführen des Applets** (*execute*-Schritt)

Der Browser lädt 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.

Execute

4. **Stoppen und löschen des Applets** (*delete*-Schritt)

Der Browser stoppt die Ausführung des Applet und gibt den Arbeitsspeicher des Client wieder frei. Dies geschieht beim „Verlassen“ des HTML-Dokumentes.

Delete

Jedes mobile Code-System, also auch Java™, 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 inklusive 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.

4.2 Bytecode: Portabilität ⇔ Effizienz

Java™ realisiert die Portabilität indem der Java-Quellcode übersetzt wird in primitive Instruktionen eines virtuellen Prozessors. Diese ma-

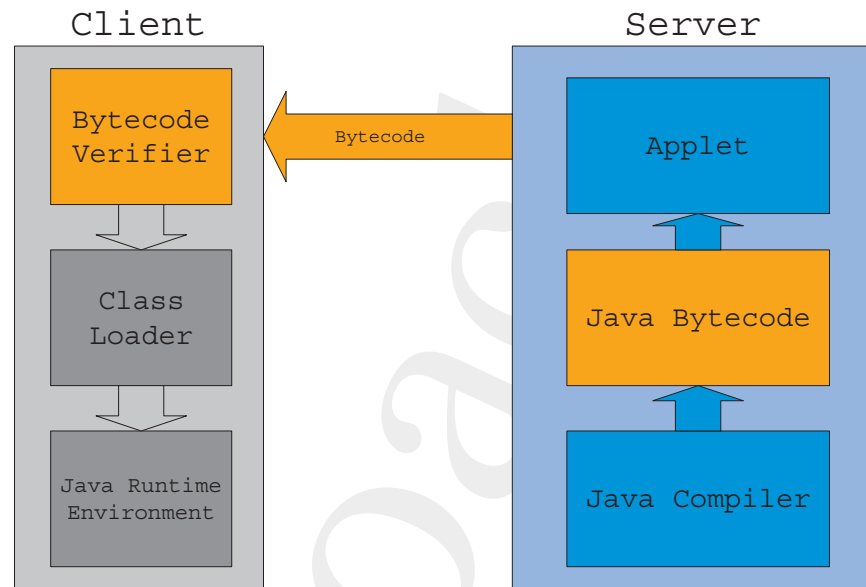


Abbildung 4.1: Von der Bytecode-Produktion bis zur Ausführung

schinennäheren, primitiven Instruktionen nennt man Bytecode. Das Com-**Bytecode** pilieren bezieht sich bei JavaTM 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 JavaTM die Größe seiner einfachen Datentypen (*PrimitiveType* \leftrightarrow Tabelle 5.5 S. 146) 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.

Der Bytecode macht JavaTM zu einer sogenannten „partiell-compilierten“ Sprache (\leftrightarrow Abbildung 4.1 S. 69).

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 JavaTM 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.

80/20-Mix

Trotz alledem ist der Bytecode beim Interpretieren über 15mal² langsamer als maschinenspezifisch kompilierter Code (\equiv *native code*). Um diesen Nachteil an Effizienz auszuräumen gibt es auch für Java™ heute *Just-In-Time-Compiler*³ (JIT) und reguläre, maschinenabhängige Compiler.

Bei genauer Betrachtung läuft jedes Java-Programm nur einen sehr geringen Prozentsatz seiner Zeit wirklich in Java. Java™ schafft nur den Eindruck über jede Plattform-Architektur Alles exakt zu kennen. Wie soll Java™ beispielsweise wissen wie eine Linie auf dem Bildschirm für jede möglich Plattform gezogen wird. Jedes Betriebssystem in dem Java™ 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™ tun kann, ist das jeweilige Betriebssystem zu beauftragen diese Dinge zu tun. So wird letztlich eine Java™ -Anwendung auch über die Abarbeitung von Routinen in anderen Programmiersprachen, beispielsweise in C-Code, realisiert.

4.3 Sicherheit

4.3.1 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 kompiliert. 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.

²↔ [Orfali/Harkey97] page 32.

³Ein 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 Anwender. Ein mehr passender Name wäre daher „*Wait to the Last Minute Holding Everybody Up Compiler*“ [Tyma98] p. 42.

⁴Näheres dazu ↔ Abschnitt 6.1 S. 157.

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 (\leftrightarrow Abbildung 4.1 S. 69).

4.3.2 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 JavaTM-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“.

Sicherheit durch Verzicht auf Zeiger JavaTM 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. JavaTM kennt keine Speicherzellen, die ihrerseits wieder Adressen zu anderen Zellen speichern. JavaTM 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.

**keine
Zeiger**

Sicherheit durch eigene Namensräume Der Klassenlader unterteilt die Menge der Klassen in unterschiedliche Namensräume. Eine Klasse

Namen

kann nur auf Objekte innerhalb ihres Namensraumes zugreifen. Java™ 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™ 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 ordnungsgemäß ausweisen.

4.4 The Road To Java

Sun Microsystems formliert unter dem Motto „The Road to Java“⁵ fünf Meilensteine, die den Weg hin zu einer Java™ Computing Architektur markieren:

⁵↔ [Sun97] S. 4

1. **Erhebung** (*Investigate*):
Sammlung von Informationen über Java™ und über die Geschäftsauswirkungen von Java™ Computing.
2. **Bewertung** (*Evaluate*):
Bewertung von Technologien und Geschäftsauswirkungen im Rahmen des jeweiligen Unternehmens.
3. **Gestaltung** (*Architect*):
Entwicklung einer um Java™ 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™ Computing Architektur.

Integrated Development Environments (IDEs) für Java™ (z. B. Forte™ for Java™, Community Edition von Sun Microsystems, Inc. oder Visual Age for Java™ 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.

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

- *AIX-Plattform:*
IBM⁶ RISC⁷-Workstation RS⁸/6000, Typ 250 und Typ 43p, Betriebssystem AIX⁹ 4.1
- *NT-Plattform:*
Intel PC, Betriebssystem Microsoft Windows NT¹⁰ und Micro- NT & XP

⁶IBM ≡ International Business Machines Corporation

⁷RISC ≡ Reduced Instruction Set Computer

⁸RS ≡ RISC System

⁹AIX ≡ Advanced Interactive Executive — IBM's Implementation eines UNIX-Operating System

soft Windows XP [Version 5.1.2600] (C) Copyright 1985-2001
Microsoft Corp.

Aufgrund der langen Fortschreibungsdauer wurden für JAVA™ –COACH ei-
ne Vielzahl von Java-Produkten genutzt. An dieser Stelle sind beispiele-
weise 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

2. **Editor und Shell:**

*Emacs*¹¹ in verschiedene Versionen zum Beispiel XEmacs, GNU¹²
Emacs und jEdit Version 4.1 pre 5 (↔ <http://www.jedit.org>).

3. **Browser:**

Netscape Communicator Version ≥ 4.03 & Microsoft Internet Explorer Version ≥ 4.0

4. **Datenaustausch** im Netz:

File Transfer Program (FTP) auf Basis von TCP/IP¹³

5. **Prozeß- und Produkt-Dokumentation:**

XHTML-Dateien auf einem Web-Server, erstellt mit ↔ Punkt 2
S. 74.

¹⁰NT ≡ New Technology, 32-Bit-Betriebssystem mit Multithreading und Multitasking

¹¹Emacs ≡ Editng Macros — gigantic, functionally rich editor

¹²GNU ≡ Free Software Foundation: GNU stands for „GNU’s Not Unix“

¹³TCP/IP ≡ Transmission Control Protocol / Internet Protocol — communications protocol in UNIX environment

Kapitel 5

Konstrukte (Bausteine zum Programmieren)

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

¹Der 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™-COACH wird jeder verwendbare „Baustein“, der bestimmte Eigenschaften hat, als Konstrukt bezeichnet.

²Zur Beschreibung der einzelnen Bausteine, also der Java™ 2 Plattform Standard Edition 5.0 Application programming interface (API) Specification:

\leftrightarrow <http://java.sun.com/j2se/1.5.0/docs/api/> (online 26-Jul-2007)

Trainingsplan

Das Kapitel „Java™ -Konstrukte“ erläutert:

- Klassen, Variablen, Methoden, Parameter, Konstruktoren, Inter-
netzgriff, Threads und GUI-Bausteine anhand von ersten Kost-
proben,
↪ Seite 76 ...
 - die Aufgaben von Applet und Applikation,
↪ Seite 127 ...
 - das Einbinden eines Applets in ein XHTML-Dokument,
↪ Seite 127 ...
 - die Syntax, die Semantik und die Pragmatik.
↪ Seite 144 ...
-

5.1 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™. Er ist nicht im Hinblick auf Effizienz optimiert oder entsprechend eines einheitlichen (Firmen-)Standards formuliert.

[Hinweis: Die Zeilennummerierung ist kein Quellcodebestandteil.]

5.1.1 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 5.1 S. 77 zeigt das Klassendiagramm für die Applikation HelloWorld.

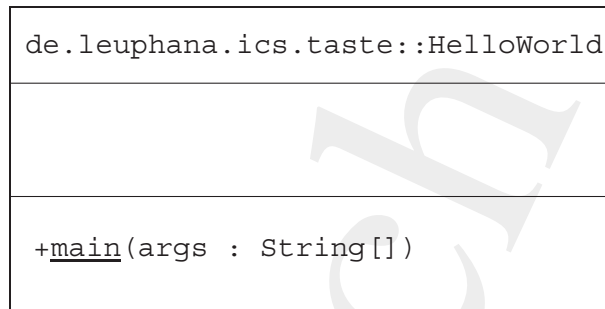


Abbildung 5.1: Klassendiagramm für HelloWorld

Da weltweit alle Java-Programmierer zunächst einmal „Hello World“ notieren, gilt es die hier gezeigte Kostprobe von allen anderen zu unterscheiden. Dazu verwendet man eine weltweit eindeutige Bezeichnung und zwar in Form eines Paketnamens. Das Paket erhält daher üblicherweise einen Namen auf der Basis der eigenen Web-Adresse, hier ist es der Paketname:

Paket

```
de.leuphana.ics.taste.3
```

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

- Sie ist von „Außen“ zugreifbar \leftrightarrow Kennwort `public`. `public`
- Da ihr Klassenname beim Aufruf der Java™ 2 Runtime Environment zugeben ist (— und nicht eine Instanz ihrer Klasse —) ist `main()` selbst eine Klassenmethode \leftrightarrow Kennwort `static`. `static`
- Da keine Rückgabe eines Wertes organisiert ist (— Wohin damit? —), ist `main()` ohne Rückgabewert definiert werden \leftrightarrow 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`

³leuphana \equiv Leuphana Universität Lüneburg; ics \equiv Institute of Computer Sciences

Die Quellcodedatei `HelloWorld.java` abgebildet in der IDE *Eclipse* von IBM \leftrightarrow Abbildung 5.2 S. 79. Näheres zur IDE *Eclipse* \leftrightarrow Abschnitt 7.1 S. 333.

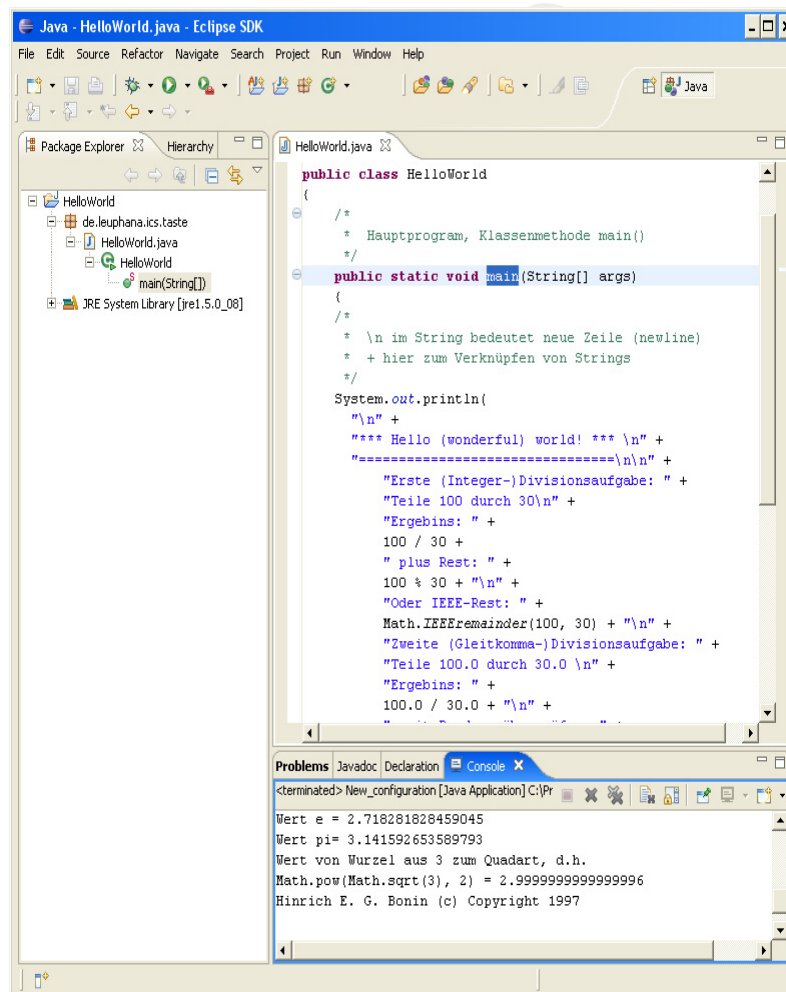
Listing 5.1: HelloWorld

```

/**
2  * Java-Kostprobe "Hello World"
  *
4  * @since      01-Jan-1997, 16-Apr-2001, 01-Jun-2007
  * @author     Hinrich E. G. Bonin
6  * @version    1.3
  */
8  package de.leuphana.ics.taste;

10 public class HelloWorld
  {
12     /*
  * Hauptprogram, Klassenmethode main()
14     */
  public static void main(String [] args)
16     {
18         /*
  * \n im String bedeutet neue Zeile (newline)
  * + hier zum Verknüpfen von Strings
20         */
  System.out.println(
22     "\n" +
  "****Hello (wonderful) world!****\n" +
24     "=====\n\n" +
  "Erste (Integer-)Divisionsaufgabe : " +
26     "Teile 100 durch 30\n" +
  "Ergebnis : " +
28     100 / 30 +
  " plus Rest : " +
30     100 % 30 + "\n" +
  "Oder IEEE-Rest : " +
32     Math.IEEEremainder(100, 30) + "\n" +
  "Zweite (Gleitkomma-)Divisionsaufgabe : " +
34     "Teile 100.0 durch 30.0\n" +
  "Ergebnis : " +
36     100.0 / 30.0 + "\n" +
  " mit Rundungsberprüfung " +
38     "(100.0 - ((100.0 / 30.0) * 30.0)) = " +
  "(100.0 - ((100.0 / 30.0) * 30.0)) + "\n" +
40     "Erste (Gleitkomma-)Multiplikationsaufgabe : \n" +
  " Multipliziere 3.33333 mit 30.0\n" +
42     "Ergebnis : " +
  "(3.33333 * 30.0) + "\n" +
44     "Werte von mathematischen Konstanten : \n" +

```



Legende:

IDE *Eclipse* Version: 3.2.0, (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 7.1 S. 333.

Abbildung 5.2: HelloWorld.java in *Eclipse Platform*

80KAPITEL 5. KONSTRUKTE (BAUSTEINE ZUM PROGRAMMIEREN)

```
46     "Wert_e=" +
      Math.E + "\n" +
48     "Wert_pi=" +
      Math.PI + "\n" +
      "Wert_von_Wurzel_aus_3_zum_Quadrat,d.h.\n" +
50     "Math.pow(Math.sqrt(3),2)=" +
      Math.pow(Math.sqrt(3), 2) + "\n" +
52     "Hinrich E. G. Bonin (c) Copyright 1997\n");
    }
54 }
```

Compilation und Ausführung von HelloWorld:

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/taste/HelloWorld.java

D:\bonin\anwd\code>java de.leuphana.ics.taste.HelloWorld

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

Erste (Integer-)Divisionsaufgabe: Teile 100 durch 30
Ergebnis: 3 plus Rest: 10
Oder IEEE-Rest: 10.0
Zweite (Gleitkomma-)Divisionsaufgabe: Teile 100.0 durch 30.0
Ergebnis: 3.3333333333333335
  mit Rundungsüberprüfung (100.0 - ((100.0 / 30.0) * 30.0)) = 0.0
Erste (Gleitkomma-)Multiplikationsaufgabe:
  Multipliziere 3.33333 mit 30.0
Ergebnis: 99.9999
Werte von mathematischen Konstanten:
Wert e = 2.718281828459045
Wert pi= 3.141592653589793
Wert von Wurzel aus 3 zum Quadrat, d.h.
Math.pow(Math.sqrt(3), 2) = 2.9999999999999996
Hinrich E. G. Bonin (c) Copyright 1997

D:\bonin\anwd\code>javadoc de/leuphana/ics/taste/HelloWorld.java
Loading source file de/leuphana/ics/taste/HelloWorld.java...
Constructing Javadoc information...
Standard Doclet version 1.5.0_08
```



```

Building tree for all the packages and classes...
Generating de/leuphana/ics/taste/\HelloWorld.html...
Generating de/leuphana/ics/taste/\package-frame.html...
Generating de/leuphana/ics/taste/\package-summary.html...
Generating de/leuphana/ics/taste/\package-tree.html...
Generating constant-values.html...
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 help-doc.html...
Generating stylesheet.css...

```

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

Die Abbildung 5.3 S. 82 zeigt einen Ausschnitt der Dokumentation von HelloWorld, die mit javadoc generiert wurde.

5.1.2 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 5.4 S. 83 zeigt das Klassendiagramm für die Applikation Foo.

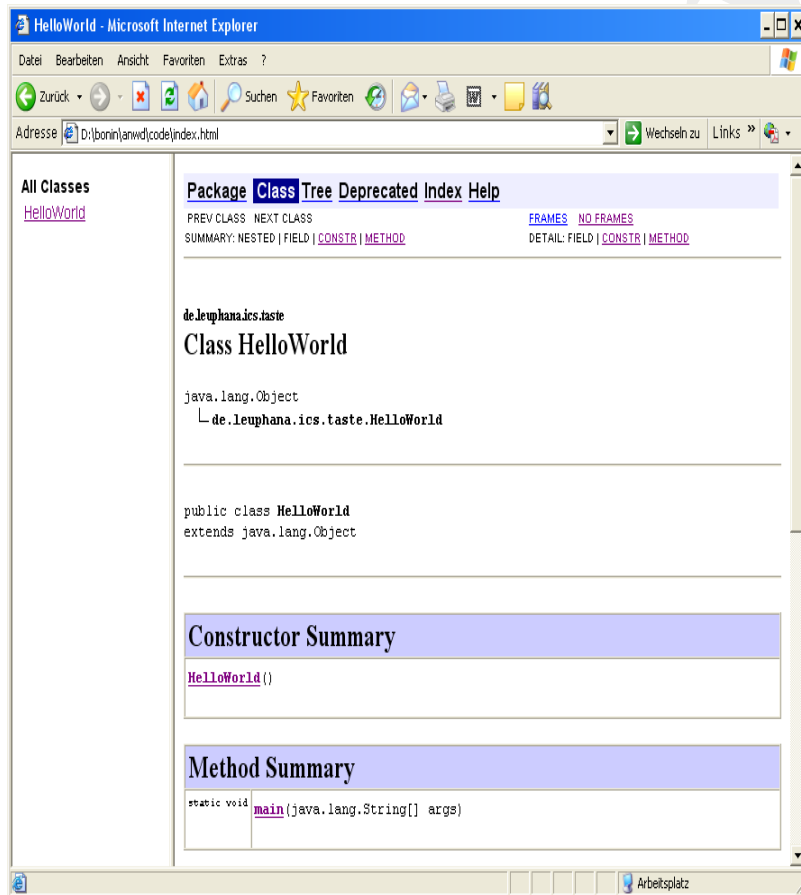
```
public static void main(String[] args) { ... } args
```

Listing 5.2: Foo

```

/**
2  * Kleiner Java Spass:
   * Demonstration der Bindung des
4  * Parameters an die Argumente
   * ("call by value")
6  *
   * @since      16-Apr-2001, 26-Nov-2002, 01-Jun-2007
8  * @author    Hinrich E. G. Bonin

```



Legende:

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

Abbildung 5.3: Beispiel HelloWorld — javadoc

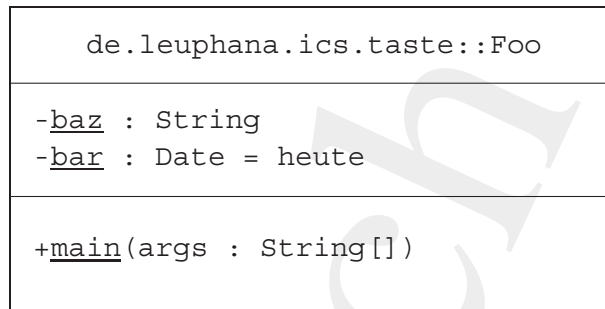


Abbildung 5.4: Klassendiagramm für Foo

```

10  * @version    2.1
11  */
12  package de.leuphana.ics.taste;
13
14  import java.util.Date;
15
16  public class Foo
17  {
18      private static String baz;
19      private static Date bar = new Date();
20
21      public static void main(String [] args)
22      {
23          for (int i = 0; i < args.length; i = i + 1)
24          {
25              System.out.println(
26                  "Eingabeteil:" +
27                  i + "\n" +
28                  "*" +
29                  args[i] +
30                  "*");
31          }
32          if (args.length != 0)
33          {
34              args[args.length - 1]
35                  = "Neuer_Wert: ";
36              baz = args[args.length - 1];
37          } else
38          {
39              baz = "Kein_Argument: ";
40          }
41          System.out.println(
42              baz +

```

```
42         "Java ist interessant!" +
43         "\n" +
44         bar.toString());
45     }
46 }
```

Nach dem Aufruf von:

```
>java de.leuphana.ics.taste.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:

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/taste/Foo.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.taste.Foo
Kein Argument: Java ist interessant!
Fri Jun 01 11:32:11 CEST 2007
```

```
D:\bonin\anwd\code>echo %PROCESSOR_IDENTIFIER%
x86 Family 6 Model 14 Stepping 8, GenuineIntel
```

```
D:\bonin\anwd\code>java
  de.leuphana.ics.taste.Foo %PROCESSOR_IDENTIFIER%
Eingabeteil:0
*x86*
Eingabeteil:1
*Family*
Eingabeteil:2
*6*
Eingabeteil:3
*Model*
Eingabeteil:4
*14*
```

```

Eingabeteil:5
*Stepping*
Eingabeteil:6
*8,*
Eingabeteil:7
*GenuineIntel*
Neuer Wert: Java ist interessant!
Fri Jun 01 11:32:38 CEST 2007

D:\bonin\anwd\code>

```

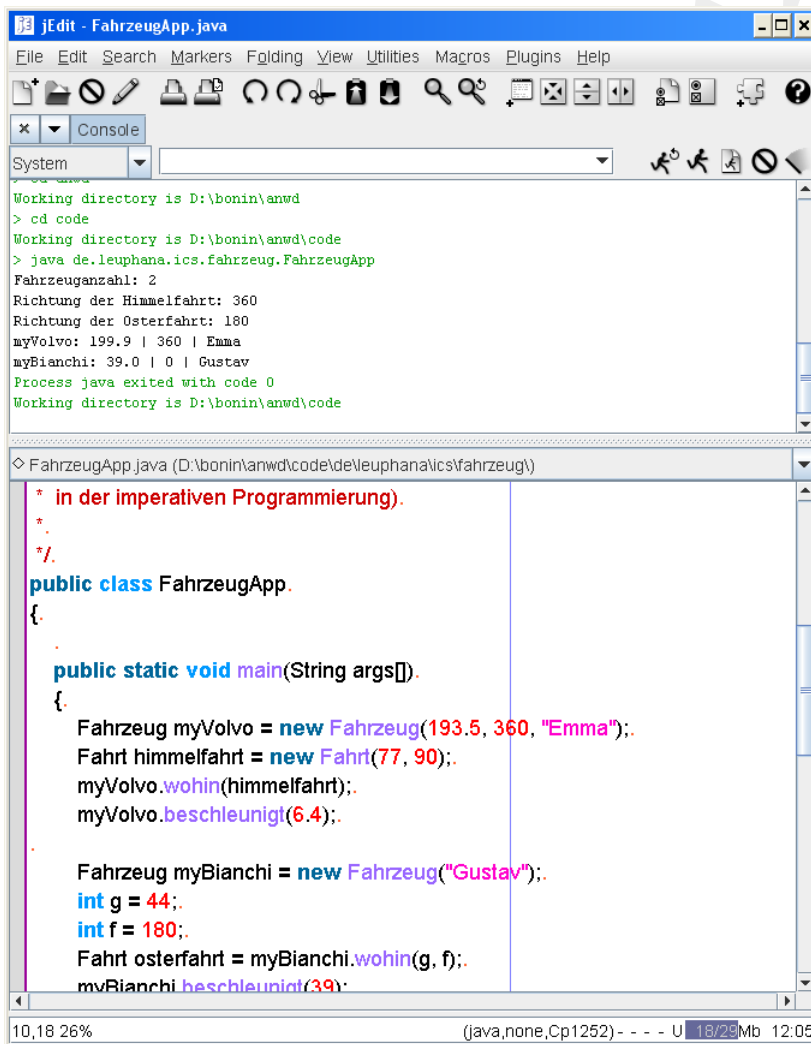
5.1.3 Kostprobe FahrzeugApp — Konstruktor

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 3.2 S. 46):
 - Geschwindigkeit
 - Fahrtrichtung
 - Eigentümer
- `class FahrzeugApp`
 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 zusammengefaßt 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 5.5 S. 86 und im GNU Emacs ↔ Abbildung 5.6 S. 87). Die Quellcodedatei `FahrzeugApp.java` (`App` ≡ Applikation) enthält die namensgleiche Klasse `FahrzeugApp` mit der `main`-Methode⁴ enthält.

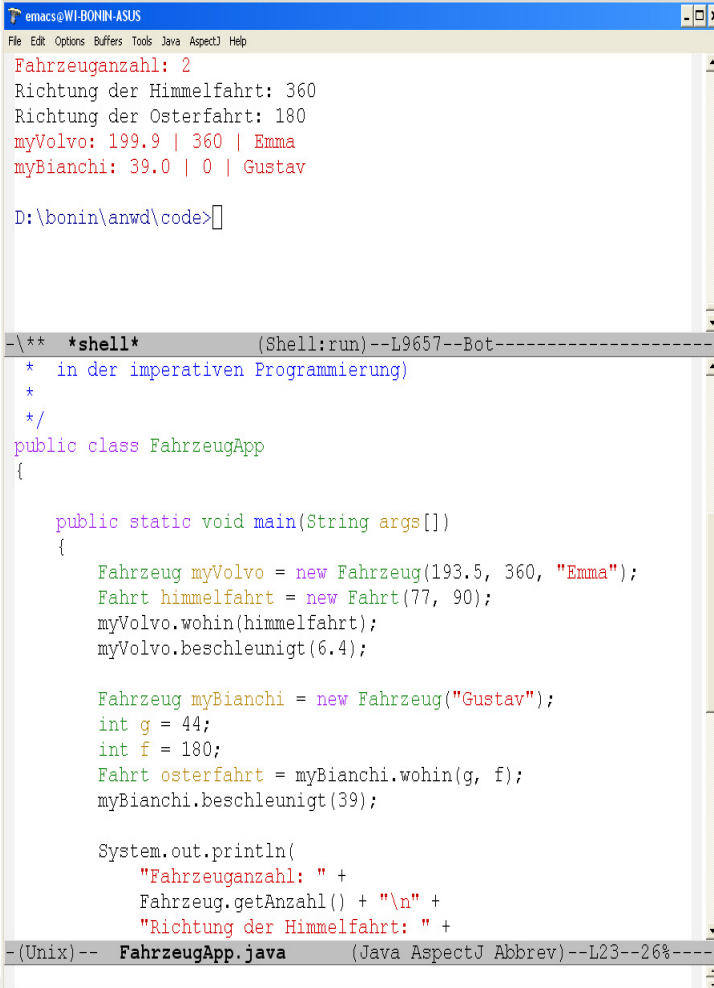
⁴[Hinweis: Jede Klasse kann — zum Beispiel zum Testen — eine Methode `main()` enthalten. Entscheidend ist die Methode `main()` der Klasse, die vom Java-Interpreter aufgerufen wird.]



Legende:

Editor jEdit Version 4.2 final ↔ <http://www.jedit.org>

Abbildung 5.5: Beispieldateien im Editor jEdit



The screenshot shows the GNU Emacs editor window titled 'emacs@WI-BONIN-ASUS'. The main text area displays the output of a Java program:

```
Fahrzeuganzahl: 2
Richtung der Himmelfahrt: 360
Richtung der Osterfahrt: 180
myVolvo: 199.9 | 360 | Emma
myBianchi: 39.0 | 0 | Gustav
```

The cursor is at the prompt 'D:\bonin\anwd\code>'. Below the main text area, a shell buffer is visible, containing the following Java code:

```
-\** *shell* (Shell:run)--L9657--Bot-----
* in der imperativen Programmierung)
*
*/
public class FahrzeugApp
{
    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: " +

```

The status bar at the bottom indicates '(Unix)-- FahrzeugApp.java (Java AspectJ Abbrev)--L23--26%----

Legende:

Editor GNU Emacs Version 21.3.1

↔ <http://www.gnu.org/software/software.html>

Abbildung 5.6: Beispieldateien im Editor GNU Emacs

Beim Compilieren der Quellcodedateien entstehen die drei Klassendateien:

- `Fahrt.class`
- `Fahrzeug.class`
- `FahrzeugApp.class`

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

Paket

`de.leuphana.ics.fahrzeug`

gewählt. Die drei Klassendateien sind relativ zum Pfad, der in `CLASS-PATH` angegeben ist, zu speichern. Die Abbildung 5.7 S. 89 zeigt das Klassendiagramm für die Applikation `FahrzeugApp`.

Die Compilierung und der Aufruf der Klasse `FahrzeugApp` erfolgt mit dem vorangestellten Paketnamen, das heißt für diese Beispiel auf einer Windows-Plattform (XP) wie folgt (↔ Protokoll S.95):

```
>javac de/leuphana/ics/fahrzeug/FahrzeugApp.java
>java java de.leuphana.ics.fahrzeug.FahrzeugApp
```

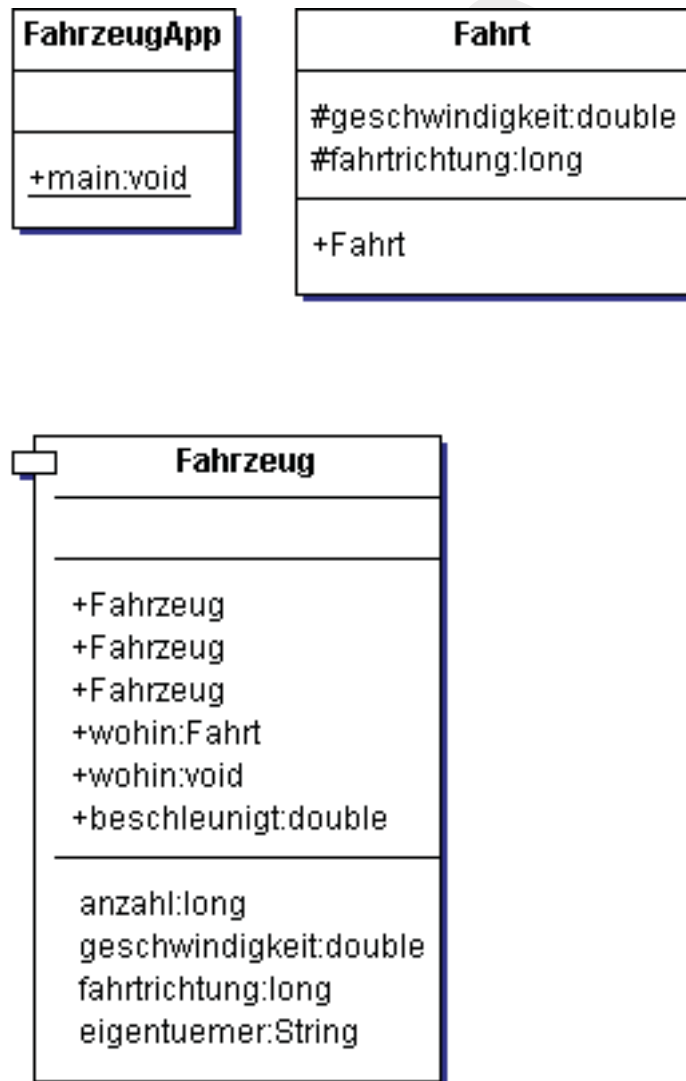
Dieses Beispiel wird auf einer Unix-Plattform (IBM AIX) folgendermaßen compiliert und appliziert:

```
c13:/home/bonin:>echo $CLASSPATH %$
/u/bonin/myjava:
/usr/lpp/J1.1.6/lib/classes.zip:/usr/lpp/J1.1.6/lib:.
c13:/home/bonin:>javac
./myjava/de/leuphana/ics/fahrzeug/FahrzeugApp.java
c13:/home/bonin:>java
de.leuphana.ics.fahrzeug.FahrzeugApp
```

[Hinweis: Ursprünglich wurden die Namensteile der Angabe für der Java™ Runtime Environment mit einem Punkt (.) getrennt angegeben, während die Angabe für den Java-Compiler die übliche Pfadtrennzeichen aufweist, also mit Slash (/) in der UNIX-Welt bzw. Backslash (\) in der Windows-Welt.]

Listing 5.3: `FahrzeugApp`

```
/**
2  * Beispiel üfr Objekterzeugung (Konstruktor) und
3  * destruktive Methoden ("call by reference") Grundidee:
4  * [RRZN97] S. 40 ff (jedoch stark modifiziert und
5  * korrigiert)
6  *
```

Legende:

Notation in *Unified Modeling Language* (UML) *Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung 5.7: Klassendiagramm für FahrzeugApp

90KAPITEL 5. KONSTRUKTE (BAUSTEINE ZUM PROGRAMMIEREN)

```
8  *
9  * @since      27-Oct-1997;...; 01-Jan-2007
10 * @author     Hinrich E. G. Bonin
11 * @version    2.2
12 */
13 package de.leuphana.ics.fahrzeug;
14 /*
15  * Klasse FahrzeugProg
16  * beschreibt primär die Kommunikation (Steuerblock)
17  * (entspricht dem Hauptprogramm
18  * in der imperativen Programmierung)
19  */
20 public class FahrzeugApp
21 {
22
23     public static void main(String args[])
24     {
25         Fahrzeug myVolvo = new Fahrzeug(193.5, 360, "Emma");
26         Fahrt himmelfahrt = new Fahrt(77, 90);
27         myVolvo.wohin(himmelfahrt);
28         myVolvo.beschleunigt(6.4);
29
30         Fahrzeug myBianchi = new Fahrzeug("Gustav");
31         int g = 44;
32         int f = 180;
33         Fahrt osterfahrt = myBianchi.wohin(g, f);
34         myBianchi.beschleunigt(39);
35
36         System.out.println (
37             "Fahrzeuganzahl:_" +
38             Fahrzeug.getAnzahl() + "\n" +
39             "Richtung_ der _Himmelfahrt:_" +
40             himmelfahrt.fahrtrichtung + "\n" +
41             "Richtung_ der _Osterfahrt:_" +
42             osterfahrt.fahrtrichtung + "\n" +
43             "myVolvo:_" +
44             myVolvo.getGeschwindigkeit() + " |_" +
45             myVolvo.getFahrtrichtung() + " |_" +
46             myVolvo.getEigentuemer() + "\n" +
47             "myBianchi:_" +
48             myBianchi.getGeschwindigkeit() + " |_" +
49             myBianchi.getFahrtrichtung() + " |_" +
50             myBianchi.getEigentuemer());
51
52         /*
53          * Referenz auf ein Objekt freigeben
54          */
55         myVolvo = null;
```

```

56      /*
        *   Garbage Collector aufrufen
58      */
        System.gc();
60    }
    }

```

Listing 5.4: Fahrzeug

```

/**
2  *   Beispiel üfr Objekterzeugung (Konstruktor) und
    *   destruktive Methoden ("call by reference") Grundidee:
4  *   [RRZN97] S. 40 ff (jedoch stark modifiziert und
    *   korrigiert)
6  *
    *
8  *   @since      27-Oct-1997;...; 01-Jan-2007
    *   @author     Hinrich E. G. Bonin
10 *   @version    2.2
    */
12 package de.leuphana.ics.fahrzeug;
    /*
14 *   Klasse Fahrzeug als "fachliches Objekt"
    *
16 */
    class Fahrzeug
18 {
        private double geschwindigkeit;
20     private long fahrtrichtung;
        private String eigentuemer;
22
        private static long anzahl;
24     /*
        *   Static Initialization Block zum
26     *   setzten der Anfangszuweisungen
        *   (kein Rueckgabewert!)
28     */
        static
30     {
            anzahl = 0;
32     }

        public static long getAnzahl()
34     {
36         return anzahl;
        }
38     /*
40     *   Datenkapselung in Klasse Fahrzeug, daher

```

92KAPITEL 5. KONSTRUKTE (BAUSTEINE ZUM PROGRAMMIEREN)

```
42     * Selektoren als Methoden definiert.
43     */
44     public double getGeschwindigkeit()
45     {
46         return geschwindigkeit;
47     }
48
49     public long getFahrtrichtung()
50     {
51         return fahrtrichtung;
52     }
53
54     public String getEigentuemer()
55     {
56         return eigentuemer;
57     }
58
59     /*
60     * Standardkonstruktor üfr "fachliche Objekte"
61     */
62     public Fahrzeug()
63     {
64         anzahl = anzahl + 1;
65     }
66
67     /*
68     * Konstruktor mit einem Parameter eigentuemer
69     * nutzt Standard-Konstruktor um
70     * Instanz zu gruenden.
71     */
72     public Fahrzeug(String eigentuemer)
73     {
74         this();
75         this.eigentuemer = eigentuemer;
76     }
77
78     /*
79     * Konstruktor nutzt Konstruktor-Hierarchie
80     * Fahrzeug(eigentuemer) --> Fahrzeug()
81     */
82     public Fahrzeug(double geschwindigkeit,
83                     long fahrtrichtung,
84                     String eigentuemer)
85     {
86         this(eigentuemer);
87         this.geschwindigkeit = geschwindigkeit;
88         this.fahrtrichtung = fahrtrichtung;
89     }
```

```

90  /*
    *  Gibt neue Instanz von Fahrt mit gewuenschter
92  *  Geschwindigkeit und Fahrtrichtung zurueck
    *  Bei einem einfachen Datentyp wird
94  *  der Wert uebergeben.
    *  Entspricht "call by value".
96  */
    public Fahrt wohin(double g, long f)
98  {
        Fahrt fahrzeugFahrt = new Fahrt(g, f);
100     return fahrzeugFahrt;
    }
102
    /*
104  *  Modifiziert eine uebergebene Instanz mit
    *  den Werten des Objektes auf das die
106  *  Methode angewendet wurde.
    *  Ein Objekt und ein Array werden
108  *  als Referenz uebergeben.
    *  Entspricht "call by reference"
110  */
    public void wohin(Fahrt fahrzeugFahrt)
112  {
        fahrzeugFahrt.geschwindigkeit = geschwindigkeit;
114     fahrzeugFahrt.fahrtrichtung = fahrtrichtung;
    }
116
    /*
118  *  Erhoeht die Geschwindigkeit um einen festen Wert
    */
120     public double beschleunigt(double g)
    {
122         geschwindigkeit = geschwindigkeit + g;
        return geschwindigkeit;
124     }
}

```

Listing 5.5: Fahrt

```

/**
2  *  Beispiel üfr Objekterzeugung (Konstruktor) und
    *  destruktive Methoden ("call by reference") Grundidee:
4  *  [RRZN97] S. 40 ff (jedoch stark modifiziert und
    *  korrigiert)
6  *
    *  @since      27-Oct-1997;...; 01-Jan-2007
8  *  @author     Hinrich E. G. Bonin
    *  @version    2.2
10  */

```

```

package de.leuphana.ics.fahrzeug;
12  /*
    * Die Klasse Fahrt als "Hilfsobjekt".
14  * Sie ist definiert, um über eine Methode den
    * Rückgabewert einer solchen Instanz zu haben.
16  */
class Fahrt
18  {
    protected double geschwindigkeit;
20  protected long fahrtrichtung;

22  /*
    * Konstruktor eines Objektes Fahrt
24  * mit zwei Parametern, die den gleichen Namen
    * wie die Datenkomponenten (Slots) haben.
26  * this-Referenzierung daher erforderlich
    */
28  public Fahrt(double geschwindigkeit,
                long fahrtrichtung)
30  {
    this.geschwindigkeit = geschwindigkeit;
32  this.fahrtrichtung = fahrtrichtung;
34  }
}

```

Im obigen Beispiel FahrzeugApp 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 5.5 S. 146) 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 (*Primitive-Type*). 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 (\leftrightarrow Tabelle 5.6 S. 147) werden stets durch ihren Wert übergeben. Bei einem „zusammengesetzten“ Objekt und einem Array (*ReferenceType* \leftrightarrow Tabelle 5.5 S. 146) wird die Referenz auf das Objekt übergeben.]

Compilation und Ausführung von `FahrzeugApp` :

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/fahrzeug/FahrzeugApp.java

D:\bonin\anwd\code>java
  de.leuphana.ics.fahrzeug.FahrzeugApp
Fahrzeuganzahl: 2
Richtung der Himmelfahrt: 360
Richtung der Osterfahrt: 180
myVolvo: 199.9 | 360 | Emma
myBianchi: 39.0 | 0 | Gustav

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

5.1.4 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 \leftrightarrow [Schader+03] S. 2–3)

Listing 5.6: Counter

```
/**
2 * Beispiel: Counter
  *
```

```

4  * @since      29-Oct-2003, 24-Nov-2006, 28-May-2007
   * @author     Hinrich E. G. Bonin
6  * @version    1.1
   */
8  package de.leuphana.ics.counter;

10 public class Counter
   {
12     private int value;

14     public int getValue()
       {
16         return value;
       }
18     public void setValue(int i)
       {
20         value = i;
       }
22     public void increment()
       {
24         ++value;
       }
26     public void decrement()
       {
28         --value;
       }
30 }

```

Listing 5.7: CounterApplication

```

/**
2  * Beispiel: Counter
   *
4  * @since      29-Oct-2003, 24-Nov-2006, 28-May-2007
   * @author     Hinrich E. G. Bonin
6  * @version    1.1
   */
8  package de.leuphana.ics.counter;

10 import java.io.IOException;

12 public class CounterApplication
   {
14     public static void main(String [] args)
       {
16         Counter c = new Counter();
           int input = -1;
18         do {
           System.out.println

```



```

20         (
21             "Counter_State :_"
22             + c.getValue()
23             + "\n"
24             + "_Please_the_next_action_(+/-/e)?_"
25         );
26     do {
27         try {
28             input = System.in.read();
29         } catch (IOException e)
30         {
31             System.exit(1);
32         }
33     } while (input != '+'
34             && input != '-'
35             && input != 'e');
36     if (input == '+')
37     {
38         c.increment();
39     } else if (input == '-')
40     {
41         c.decrement();
42     }
43 }
44 while (input != 'e');
45 }
46 }

```

Protokoll CounterApplication.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/counter/CounterApplication.java

D:\bonin\anwd\code>java
  de.leuphana.ics.counter.CounterApplication
Counter State: 0
Please the next action (+/-/e)?
+
Counter State: 1
Please the next action (+/-/e)?
+

```

ET-Essen		R1	R2	R3	R4
B1	Gewicht ≤ 75 [kg]?	J	J	N	N
B2	Körperlänge ≤ 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 5.1: Beispiel: ET-Ernährung

```
Counter State: 2
Please the next action (+/-/e)?
-
Counter State: 1
Please the next action (+/-/e)?
e

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

5.1.5 Kostprobe Essen — Eingabe von Konsole

Die Tabelle 5.1 S. 98 zeigt einfache Regeln zur Thematik „Essen“. Die Klasse Essen bildet diese Entscheidungstabelle mit Hilfe der Klasse Console ab.

Listing 5.8: Essen

```
/**
2  * Beispiel: Abbildung der ET-Essen
3  * Dateneingabe per Konsole
4  *
5  * @since      03-Nov-2002, 01-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.2
8  */
package de.leuphana.ics.et;

import java.io.IOException;

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

```

18     {
19         final int kgGrenze = 75;
20         final int cmGrenze = 180;
21
22         int gewicht =
23             Console.readInteger(
24                 "Gewicht in [kg] eingeben : ");
25         int laenge =
26             Console.readInteger(
27                 "Länge in [cm] eingeben : ");
28
29         if (gewicht <= kgGrenze)
30         {
31             if (laenge <= cmGrenze)
32             {
33                 System.out.println("Weiter so!");
34             }
35             else
36             {
37                 System.out.println("Mehr essen!");
38             }
39         }
40         else
41         {
42             if (laenge <= cmGrenze)
43             {
44                 System.out.println(
45                     "Dringend abnehmen!");
46             }
47             else
48             {
49                 System.out.println("Weiter so!");
50             }
51         }
52     }

```

In Java wird ein Objekt von welchem man eine Sequence von Bytes lesen kann als *Input Stream* charakterisiert. `System.in` ist ein vordefiniertes Objekt einer Unterklasse der Klasse `InputStream` und ermöglicht Daten über die Tastatur (*Keyboard*) einzulesen.

Listing 5.9: Console

```

/**
2  * Beispiel: Abbildung der ET-Essen---
3  * Dateneingabe per Console
4  *
5  * @since      03-Nov-2002, 01-Jun-2007
6  * @author     Hinrich E. G. Bonin

```

100KAPITEL 5. KONSTRUKTE (BAUSTEINE ZUM PROGRAMMIEREN)

```

8   * @version    1.2
9   */
10  package de.leuphana.ics.et;
11
12  import java.io.BufferedReader;
13  import java.io.InputStreamReader;
14  import java.io.IOException;
15
16  public class Console
17  {
18      private static BufferedReader
19          input =
20          new BufferedReader (
21              new InputStreamReader(System.in));
22
23      public static String readString(String message)
24          throws IOException
25      {
26          System.out.println(message);
27          return input.readLine();
28      }
29
30      public static boolean readBoolean(String message)
31          throws IOException
32      {
33          System.out.println(message);
34          return Boolean.valueOf(
35              input.readLine()).
36              booleanValue();
37      }
38
39      public static char readChar(String message)
40          throws IOException
41      {
42          System.out.println(message);
43          return input.readLine().charAt(0);
44      }
45
46      public static double readDouble(String message)
47          throws IOException
48      {
49          System.out.println(message);
50          return Double.parseDouble(input.readLine());
51      }
52
53      public static int readInteger(String message)
54          throws IOException
55      {
56          System.out.println(message);

```

```

56     return Integer.parseInt(input.readLine());
    }
58     public static long readLong(String message)
60         throws IOException
    {
62         System.out.println(message);
        return Long.parseLong(input.readLine());
64     }
}

```

Compilation von Essen, Console und Ausführung:

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/et/Essen.java

D:\bonin\anwd\code>java de.leuphana.ics.et.Essen
Gewicht in [kg] eingeben:
67
Körperlänge in [cm] eingeben:
185
Mehr essen!

D:\bonin\anwd\code>

```

5.1.6 Kostprobe Ei & Huhn — Compilieren

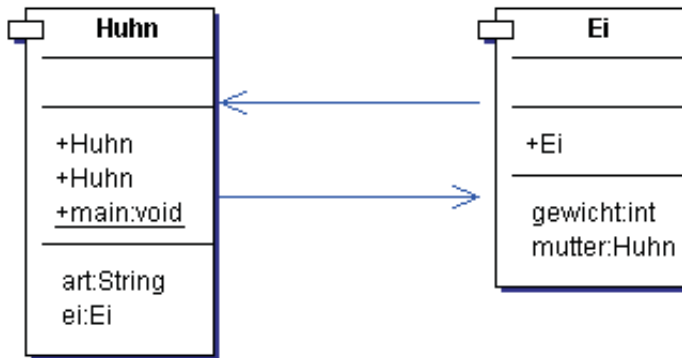
Beziehen sich zwei Klassen aufeinander (*Assoziation*) dann wird beim Compilieren einer Klasse die referenzierte Klasse ebenfalls compiliert. Beim sogenannten „Ei-Huhn-Problem“ (↔ Abbildung 5.8 S. 102) geht es um eine Klasse Huhn (↔ S. 103) die sich auf eine Klasse Ei (↔ S. 101) bezieht und umgekehrt. 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. 104) verdeutlicht dieses implizite Compilieren.

Listing 5.10: Ei

```

/**
2  * Beispiel: Kompilierung von Klassen
  * --- das sogenannte

```



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center*™ 6.2.

Abbildung 5.8: Ei-Huhn-Reihenfolgenproblem

```

4  * "Ei-Huhn-Problem" ---
5  *
6  * @since      23-Dec-2002, 24-Oct-2006, 01-Jun-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    1.3
9  */
10 package de.leuphana.ics.huhn;
11
12 public class Ei
13 {
14     private int gewicht;
15     private Huhn mutter;
16
17     public int getGewicht()
18     {
19         return gewicht;
20     }
21
22     public void setGewicht(int gewicht)
23     {
24         this.gewicht = gewicht;
25     }
26
27     public Huhn getMutter()
28     {
29         return mutter;
30     }
31 }
    
```

```

32     public void setMutter(Huhn mutter)
33     {
34         this.mutter = mutter;
35     }
36
37     public Ei(int gewicht, Huhn mutter)
38     {
39         this.setGewicht(gewicht);
40         this.setMutter(mutter);
41     }
42 }

```

Listing 5.11: Huhn

```

/**
2  * Beispiel: Kompilierung von Klassen
3  * ---- das sogenannte
4  * "Ei-Huhn-Problem"----
5  *
6  * @since    23-Dec-2002, 24-Oct-2006, 01-Jun-2007
7  * @author   Hinrich E. G. Bonin
8  * @version  1.3
9  */
10 package de.leuphana.ics.huhn;
11
12 public class Huhn
13 {
14     private String art;
15     private Ei ei;
16
17     public String getArt()
18     {
19         return art;
20     }
21
22     public void setArt(String art)
23     {
24         this.art = art;
25     }
26
27     public Ei getEi()
28     {
29         return ei;
30     }
31
32     public void setEi(Ei ei)
33     {
34         this.ei = ei;

```

```

    }
36
    public Huhn(String art, Ei ei)
38    {
        this.setArt(art);
40        this.setEi(ei);
    }
42
    public Huhn()
44    {
    }
46
    public static void main(String [] args)
48    {
        System.out.println(
50            "Das Ei wiegt " +
                new Huhn("hybrid",
52                    new Ei(60,
                        new Huhn())).
54            getEi().getGewicht() +
                "g.");
56    }
}

```

Compilation von Ei und Ausführung von Huhn:

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>dir de\leuphana\ics\huhn\*.class

Datei nicht gefunden

D:\bonin\anwd\code>javac de/leuphana/ics/huhn/Ei.java

D:\bonin\anwd\code>dir de\leuphana\ics\huhn\*.class
    674 Ei.class
    1.332 Huhn.class

D:\bonin\anwd\code>java de.leuphana.ics.huhn.Huhn
Das Ei wiegt 60g.

D:\bonin\anwd\code>

```


5.1.7 Kostprobe MyNetProg — Internetzugriff

In diesem Beispiel wird auf ein dynamisches⁵ Dokument von einem Web-Server⁶ zugegriffen. Verwendet wird dabei die Klasse `URL`⁷ des Standardpaketes:

URL

```
java.net.
```

java.net

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 Methode `getInputStream()` wird der HTTP⁸-Datenstrom gelesen. Die Abbildung 5.9 S. 106 zeigt das Klassendiagramm der Applikation `MyNetProg`. Die Abbildung 5.10 S. 107 zeigt die nachgefragte Datei `spass.ksh` direkt dargestellt im Browser.

Listing 5.12: MyNet

```

/**
 2  * Beispiel fuer HTTP-Kommunikation;
 3  * Grundidee : [RRZN97] S. 111 ff
 4  *
 5  * @since      28-Oct-1997, 24-Dec-2002, 12-Jun-2007
 6  * @author     Hinrich E. G. Bonin
 7  * @version    2.1
 8  */
 9
10 package de.leuphana.ics.mynet;
11
12 import java.net.URL;
13 import java.net.URLConnection;
14
15 import java.io.DataInputStream;
16 import java.io.IOException;
17
18 class MyNet
19 {
20     private URL homePage;

```

⁵„Dynamisches“ Dokument über CGI-Skript angestoßen (CGI ≡ Common Gateway Interface)

⁶ Ursprünglich `http://193.174.33.106:80` danach `http://as.fhnon.de`

⁷URL ≡ Uniform Resource Locator

⁸HTTP ≡ HyperText Transfer Protocol

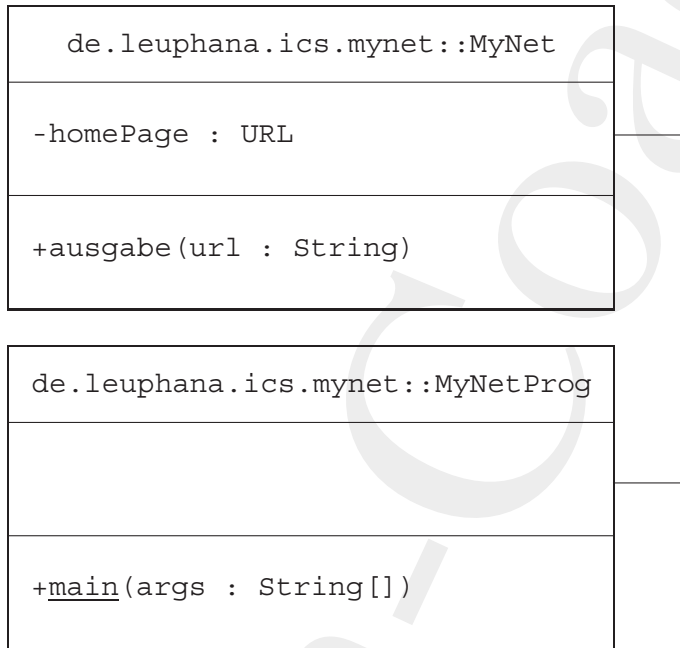


Abbildung 5.9: Klassendiagramm der Applikation MyNetProg

```

Das Datum meines Rechners IBM RS/6000 (cl3, IP=193.174.33.106) ist.

Tue Jun 12 09:12:24 DFT 2007

Berechnung der Fakultät von n=100


$$fac(n) = \begin{cases} 1 & : n < 1 \\ n * fac(n-1) & : n > 1 \end{cases}$$


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
:::
::: Fakultätsfunktion
(begin

(define fac (lambda (n)
  (cond ((< n 1) 1)
        (#T (* n (fac (- n 1))))))

(fac 100))

:Value: 9332621544394415268169923885626670049071596826438162146859296389521

```

Legende:

CGI-File `spass.ksh` direkt angezeigt mit Browser *Microsoft Internet Explorer*, Version 6.0.2900, auf einer Windows-XP-Plattform.

Abbildung 5.10: Darstellung der CGI-Datei `spass.ksh`

```

22 public void ausgabe(String url)
    {
24     try
        {
26         URL homePage = new URL(url);
28         System.out.println(
30             "URL:␣" +
32             homePage + "\n" +
34             "WWW-Server:␣" +
36             homePage.getHost());
38         /*
39          * Verbindung zum Dokument
40          */
42         URLConnection homePageConnection =
44             homePage.openConnection();
46         /*
47          * Den gelieferten
48          * HTTP-Datenstrom in ein
49          * DataInputStream wandeln
50          */
52         DataInputStream in =
54             new DataInputStream(
56                 homePageConnection.
58                 getInputStream());
60         /*
61          * Ausgeben zeilenweise
62          */
64         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
            }
    }
}

```

Listing 5.13: MyNetProg

/**

```

2  * Beispiel fuer HTTP-Kommunikation;
3  * Grundidee : [RRZN97] S. 111 ff
4  *
5  * @since      28-Oct-1997, 24-Dec-2002, 12-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    2.1
8  */
9  package de.leuphana.ics.mynet;
10
11 public class MyNetProg
12 {
13     public static void main(String [] args)
14     {
15         MyNet netObject = new MyNet();
16         /*
17          * Fest verdrahtete URL-Angabe
18          */
19         netObject.ausgabe(
20             "http://193.174.33.106:80/cgi-bin/spass.ksh"
21         );
22     }
23 }

```

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

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac -Xlint:deprecation
de/leuphana/ics/mynet/MyNetProg.java
.\de\leuphana\ics\mynet\MyNet.java:50:
warning: [deprecation] readLine()

```

110KAPITEL 5. KONSTRUKTE (BAUSTEINE ZUM PROGRAMMIEREN)

```
in java.io.DataInputStream has been deprecated
    String line = in.readLine();
                    ^
```

1 warning

```
D:\bonin\anwd\code>java de.leuphana.ics.mynet.MyNetProg
URL: http://193.174.33.106:80/cgi-bin/spass.ksh
WWW-Server: 193.174.33.106
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: (c13, IP=193.174.33.106) ist:</p> <pre>
11: Tue Jun 12 09:04:27 DFT 2007
12: </pre>
13: <p> Berechnung der Fakult&auml;t von n=100</p>
14: <p></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))
37:
38:
39: ;Value: 93326215443944 ...
...
```

```
84: </html>
```

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

Alternativlösung Klasse MyGetWebPage [Hinweis: Die Alternativlösung arbeitet direkt mit der Port-Nummer 80, die standardgemäß⁹ für das Hypertext Transfer Protocol (HTTP) vorgesehen ist.]

Listing 5.14: MyGetWebPage

```
/**
2  * Beispiel üfr den Zugriff auf eine Web-Page
  * Idee: Java Technology Fundamentals
4  * Newsletter 8-Mar-2004
  *
6  * @since      11-Mar-2004, 12-Jun-2007
  * @author     Hinrich E. G. Bonin
8  * @version    1.2
  */
10 package de.leuphana.ics.mynet;

12 import java.net.InetAddress;
import java.net.Socket;
14
import java.io.BufferedReader;
16 import java.io.InputStream;
import java.io.InputStreamReader;
18 import java.io.PrintWriter;
import java.io.OutputStream;
20 import java.io.OutputStreamWriter;

22 public class MyGetWebPage
  {
24     public static void main(String [] args)
        throws Exception
26     {
        if (args.length != 2)
28         {
```

⁹Rechner 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

```

30         System.err.println (
31             "java_␣MyGetWebPage_␣" +
32             "hostname_␣document");
33         return ;
34     }
35     String host = args [0];
36     String document = args [1];
37     InetAddress addr =
38         InetAddress .getByName (host);
39     Socket socket = new Socket (addr, 80);
40     InputStream is = socket .getInputStream ();
41     OutputStream os = socket .getOutputStream ();
42     BufferedReader br = new BufferedReader (
43         new InputStreamReader (is));
44     PrintWriter pw = new PrintWriter (
45         new OutputStreamWriter (os));
46     pw.print ("GET_␣/" +
47             document +
48             "_␣HTTP/1.0_␣\n_␣\n");
49     pw.flush ();
50     String line;
51     while ((line = br.readLine ()) != null)
52     {
53         System.out.println (line);
54     }
55     pw.close ();
56     br.close ();
57 }

```

Protokoll MyGetWebPage.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/mynet/MyGetWebPage.java

D:\bonin\anwd\code>java de.leuphana.ics.mynet.MyGetWebPage
  193.174.33.106 cgi-bin/spass.ksh
HTTP/1.1 200 OK
Date: Tue, 12 Jun 2007 07:28:29 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
(c13, IP=193.174.33.106) ist:</p> <pre>
Tue Jun 12 09:28:29 DFT 2007
</pre>
<p> Berechnung der Fakultät von n=100</p>
<p></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
;;;
;;; Fakultätsfunktion
(begin

(define fac (lambda (n)
  (cond ((< n 1) 1)
        (#T (* n (fac (- n 1))))))

(fac 100))

;Value: 9332621544 ...

1 ]=> ;;;; 129-digit RSA

;;; Faktorisierungsergebnis von 129-digit RSA
;;; (bekanntgegeben am April 26, 1994)
;;; Beteiligt ca. 600 Freiwillige mit 1600 Rechnern im Internet
;;; von Workstation ueber Mainframe bis zum Supercomputer.
;;; (Benutzte Quelle der Zahlen:

```

```

;;; Simson Garfinkel; Pretty Good Privacy, 1995 (O'Reilly) Page 5)
;;;
;;; Richtigkeit der Zahlen n, p, q mit n=p*q gegeben,
;;; da die Variable gleichheit? den Wert #t (True) hat.
;;;
(begin ...

</html>

D:\bonin\anwd\code>

```

5.1.8 Kostprobe ImpulseGenerator — Thread

Java™ 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*. Ein einfaches Beispiel 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.

Listing 5.15: ImpulseGenerator

```

/**
2  * Example: ImpulseGenerator
   *
4  *
   * @since      03-Nov-2006, 01-Jun-2007
6  * @author     Hinrich E. G. Bonin
   * @version    1.2
8  */
package de.leuphana.ics.timer;

10 import java.io.IOException;
12 import java.util.Timer;
import java.util.TimerTask;
14
16 public class ImpulseGenerator extends TimerTask
{
    int tick = 0;
18
    public void run()
20 {
        this.tick++;
22        System.out.println("Tick:␣" + this.tick);
    }
}

```

```

24     public static void main(String args[])
25         throws IOException
26     {
27         Timer timer = new Timer();
28
29         timer.schedule(new ImpulseGenerator(),
30                        0,
31                        1000);
32
33         System.out.println("Press ENTER to stop");
34
35         // Reads a byte of data
36         //and returns the byte read-->ASCII 13.
37         System.out.println(System.in.read());
38
39         timer.cancel();
40     }
41 }
42 }

```

Protokoll ImpulseGenerator.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM (
build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de/leuphana/ics/timer/ImpulseGenerator.java

D:\bonin\anwd\code>java
de.leuphana.ics.timer.ImpulseGenerator
Press ENTER to stop
Tick: 1
Tick: 2
Tick: 3

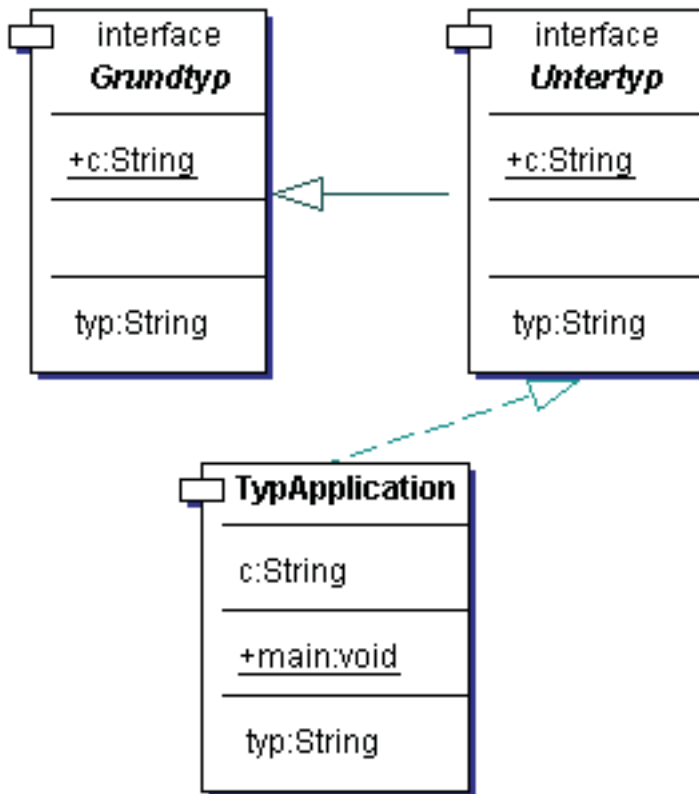
13

D:\bonin\anwd\code>

```

5.1.9 Kostprobe TypApplication — Interface

Mit einem Interface lässt sich vorgeben welche Methoden eine Klasse zu enthalten hat, vorausgesetzt die Klasse implementiert das Interface.



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center™ 6.2*.

Abbildung 5.11: Interface-Vererbung

Präziser gesagt, mit welcher Signatur, also welcher Typ von Rückgabewert einzuhalten ist und welche Typen die einzelnen Parameter der jeweiligen Methode haben müssen. Zusätzlich lassen sich Konstanten über das Interface für die implementierende Klasse definieren.

Dabei können Interfaces untereinander in einer Vererbungsbeziehung stehen (\leftrightarrow Abbildung 5.11 S. 116). Das Beispiel `TypApplication` (\leftrightarrow S. 117) verdeutlicht die Aufgabe eines Interfaces als Spezifikation eines Typs für ein Objekt. Die Protokolldatei (\leftrightarrow S. 119) verdeutlicht einerseits die Konstanten (implizit `final`) und andererseits die Vererbung zwischen dem Interface `Untertyp` (\leftrightarrow S. 118) und Grundtyp (\leftrightarrow S. 118).

Listing 5.16: `TypApplication`

```
/**
2  * Beispiel:
3  * Interfaces mit Konstanten
4  *
5  * @since    24-Oct-2006
6  * @author  Hinrich Bonin
7  * @version 1.0
8  */
10 package de.unilueneburg.as.typ;
12 public class TypApplication implements Untertyp
13 {
14     String c = "TypApplication";
15
16     public String getTyp()
17     {
18         return c;
19     }
20     public void setTyp(String c)
21     {
22         this.c = c;
23     }
24
25     public static void main(String [] args)
26     {
27
28         TypApplication t = new TypApplication();
29         t.setTyp("Attention");
30
31         Untertyp u = (Untertyp) t;
32         Grundtyp g = (Grundtyp) u;
33
34         g.setTyp("OK");
35     }
36 }
```

```

36     System.out.println (
37         "1_" + t.getTyp() + "\n" +
38         "2_" + u.getTyp() + "\n" +
39         "3_" + g.getTyp() + "\n" +
40         "4_" + t.c      + "\n" +
41         "5_" + u.c      + "\n" +
42         "6_" + g.c);

44     // Hinweis :
45     //     g.c = "cannot assign!";
46     //
47     // de/unilueneburg/as/typ/Application.java:33:
48     // cannot assign a value to final variable c
49     // stets "final" im Interface
50
51 }
52 }

```

Listing 5.17: Grundtyp

```

/**
2  * Beispiel:
3  * Interfaces mit Konstanten
4  *
5  * @since      24-Oct-2006
6  * @author     Hinrich Bonin
7  * @version    1.0
8  */

10 package de.unilueneburg.as.typ;

12 public interface Grundtyp
13 {
14     String c = "Grundtyp";

16     public String getTyp();

18     public void setTyp(String typ);
19 }

```

Listing 5.18: Untertyp

```

/**
2  * Beispiel:
3  * Interfaces mit Konstanten
4  *
5  * @since      24-Oct-2006
6  * @author     Hinrich Bonin

```

```

    * @version    1.0
8   */
10  package de.unilueneburg.as.typ;
12  public interface Untertyp extends Grundtyp
13  {
14      String c = "Untertyp";
16      public String getTyp();
18  }

```

Compilation von TypApplication und Ausführung:

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de/unilueneburg/as/typ/TypApplication.java

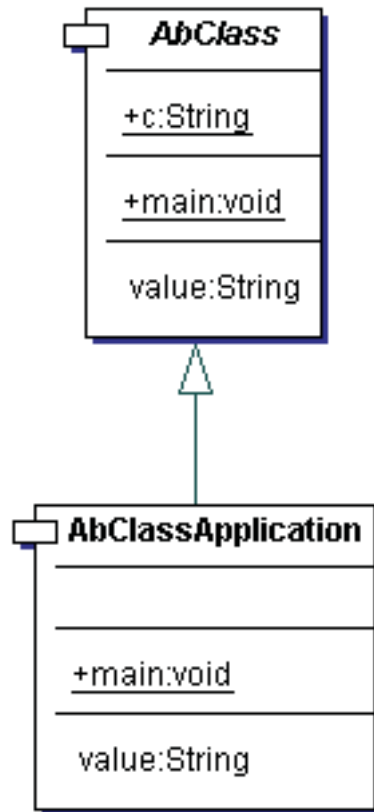
D:\bonin\anwd\code>java
de.unilueneburg.as.typ.TypApplication
1 OK
2 OK
3 OK
4 OK
5 Untertyp
6 Grundtyp

D:\bonin\anwd\code>

```

5.1.10 Kostprobe AbClassApplication — Abstrakte Klasse

Mit einer abstrakten Klasse können wie bei einem Interface Methoden in ihrer Signatur spezifiziert werden. Zusätzlich kann eine abstrakte Klasse auch Attribute (Slots), das heisst Instanz- und Klassenvariablen, spezifizieren und sogar die Methode `main()` aufweisen um direkt appliziert werden zu können (\leftrightarrow Abbildung 5.12 S. 120). Das Beispiel `AbClassApplication` (\leftrightarrow S. 121) verdeutlicht die Aufgabe einer



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung 5.12: Abstrakte Klasse

abstrakten Klasse als vorgegebene Eigenschaften für eine Vererbung. Die Protokolldatei (↔ S. 122) verdeutlicht beispielhaft diese Aufgabe.

Listing 5.19: AbClassApplication

```

/**
2  * Beispiel:
  * Abstract Class
4  *
  * @since    24-Oct-2006
6  * @author  Hinrich Bonin
  * @version  1.0
8  */

10 package de.unilueneburg.as.abclass;

12 public class AbClassApplication extends AbClass
  {
14     public String getValue()
      {
16         return c;
      }
18     public void setValue(String c)
      {
20         AbClass.c = c;
      }
22
24     public static void main(String [] args)
      {
26         AbClassApplication a =
          new AbClassApplication ();
          a.setValue ("OK" );
28
          System.out.println (
30             "1_" + a.getValue () + "\n" +
              "2_" + c);
32     }
  }

```

Listing 5.20: AbClass

```

/**
2  * Beispiel:
  * Abstract Class
4  *
  * @since    24-Oct-2006
6  * @author  Hinrich Bonin
  * @version  1.0
8  */

```

```

10 package de.unilueneburg.as.abclass;
12 public abstract class AbClass
13 {
14     public static String c = "Value";
16     public abstract String getValue();
18     public abstract void setValue(String c);
20     public static void main(String [] args)
21     {
22         System.out.println(c);
23     }
24 }

```

Compilation von AbClassApplication und Ausführung:

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

```

```

D:\bonin\anwd\code>javac
de/unilueneburg/as/abclass/AbClassApplication.java

```

```

D:\bonin\anwd\code>java
de.unilueneburg.as.abclass.AbClassApplication
1 OK
2 OK

```

```

D:\bonin\anwd\code>java
de.unilueneburg.as.abclass.AbClass
Value

```

```

D:\bonin\anwd\code>

```

5.1.11 Kostprobe ActionApplet — GUI

Implementierungsvererbung:

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

Verhaltensvererbung:

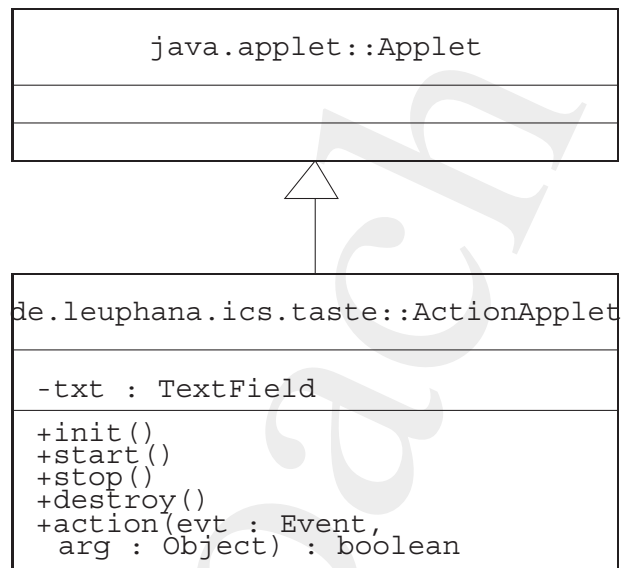


Abbildung 5.13: Klassendiagramm für ActionApplet

Wer von einem Interface erbt,
 der muss etwas tun,
 denn er verpflichtet sich,
 das in der Schnittstelle definierte Verhalten
 zu implementieren.
 (↔ [Broy/Siedersleben02] S. 7)

Die Abbildung 5.13 S. 123 zeigt das Klassendiagramm für ActionApplet.

Listing 5.21: ActionApplet

```

/**
2  * ActionApplet.class zeichnet geschachtelte Panels und
  * akzeptiert in der Mitte einen Text, der in der
4  * Statuszeile des Browsers und auf der Console
  * (System.out) ausgegeben wird. Grundidee: [HSS96]
6  *
  *
8  * @since      22-Jan-1997, 13-Jul-1998, 01-Jun-2007
  * @author     Hinrich E. G. Bonin
10 * @version    1.3
  */
12 package de.leuphana.ics.taste;
  
```

124 KAPITEL 5. KONSTRUKTE (BAUSTEINE ZUM PROGRAMMIEREN)

```
14 import java.awt.BorderLayout;
import java.awt.Button;
16 import java.awt.Color;
import java.awt.Event;
18 import java.awt.Font;
import java.awt.Panel;
20 import java.awt.TextField;
import java.applet.Applet;
22
23 public class ActionApplet extends Applet
24 {
    /*
26     * Textfeld zur Erfassung eines Textes, der
    * dann in der Statuszeile des Browsers angezeigt wird.
28     */
    private TextField txt;
30
    public void init()
31 {
32     Font pFont = new Font("Helvetica",
33                          Font.BOLD,
34                          24);
35     setLayout(new BorderLayout());
36     setBackground(Color.white);
37     setForeground(Color.green);
38
39     add("North", new Button("Norden"));
40     add("South", new Button("Sueden"));
41
42     /*
43     * Erzeugt ein Panel p0 mit Struktur im Zentrum
44     */
45     Panel p0 = new Panel();
46     p0.setBackground(Color.red);
47     p0.setForeground(Color.white);
48     p0.setLayout(new BorderLayout());
49     add("Center", p0);
50     p0.add("North", new Button("Oben"));
51     p0.add("South", new Button("Unten"));
52
53     /*
54     * Erzeugt ein Panel p1 mit Struktur im Zentrum von p0
55     */
56     Panel p1 = new Panel();
57     p1.setBackground(Color.blue);
58     p1.setForeground(Color.black);
59     p1.setLayout(new BorderLayout());
60     add("Center", p1);
61     p1.add("North", new Button("Hamburg"));
62 }
```

```

        p1.add("South", new Button("Hannover"));
64
        /*
66         * Setzt das Textfeld in die Mitte des inneren Panels
        */
68        txt = new TextField(10);
        txt.setFont(pFont);
70        p1.add("Center", txt);

72        p1.add("East", new Button("Bleckede"));
        p1.add("West", new Button("Salzhausen"));
74

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

78        add("West", new Button("Westen"));
        add("East", new Button("Osten"));
80    }

82    public void start()
    {
84    }

86    public void stop()
    {
88    }

90    public void destroy()
    {
92    }

94    public boolean action(Event evt, Object arg)
    {
96        System.out.println(
97            ((Button) evt.target).getLabel()
98            + ":\u2013" +
99            txt.getText());
100        showStatus(
101            ((Button) evt.target).getLabel()
102            + ":\u2013" +
103            txt.getText());
104        return true;
    }
106 }

```

Ausführen des Applets ActionApplet:

Um das Applet ActionApplet ausführen zu können bedarf es einer HTML-Datei in der das Applet eingebunden ist. Die Einbindung eines

Applets wird in Abschnitt 5.2 S. 127 ausführlich behandelt. Hier wird die Einbindung mittels `<object>`-Konstrukt genutzt. Die Ausführung wird dann dem *Appletviewer* aus dem Java-Standardpaket übertragen (↔ S. 126).

Listing 5.22: ActionApplet.html

```

2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4 <!-- Hinrich E. G. Bonin 13-Jan-1997 -->
  <!-- Update ... 01-Jun-2007 -->
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
6 <head>
  <base href="http://ics.leuphana.de/" />
8 <title>Action Applet</title>
  <meta http-equiv="Last-Modified"
10   content="01-Jun-2007 13:00:00 GMT" />
  <meta http-equiv="Expires"
12   content="31-Dec-2010 00:00:00 GMT" />
  </head>
14 <body>
  <h1>Action Applet</h1>
16 <p>
  <object codetype="application/java"
18   classid="java:de.leuphana.ics.taste.ActionApplet"
    code="de.leuphana.ics.taste.ActionApplet"
20   width="450" height="125"
    standby="Hier_kommt_gleich_was_zum_Clicken!">
22   [Java Applet ActionApplet]
  </object>
24 </p>
  </body>
26 </html>

```

Protokolldatei ActionApplet.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac -Xlint:deprecation
  de/leuphana/ics/taste/ActionApplet.java
de/leuphana/ics/taste/ActionApplet.java:94:
  warning: [deprecation]
    action(java.awt.Event, java.lang.Object)

```



Abbildung 5.14: Beispiel: ActionApplet

```
in java.awt.Component has been deprecated
public boolean action(Event evt, Object arg)
                ^
```

1 warning

```
D:\bonin\anwd\code>appletviewer ActionApplet.html
```

```
Westen: Alles klar?
```

```
Bleckede: Alles klar?
```

```
Norden: Alles klar?
```

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

5.2 Applet-Einbindung in ein Dokument

5.2.1 Applet ⇔ Applikaton

JavaTM unterscheidet zwei Ausführungstypen:¹⁰

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 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

¹⁰auch als Programm(formen) bezeichnet

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`.

5.2.2 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>`“.

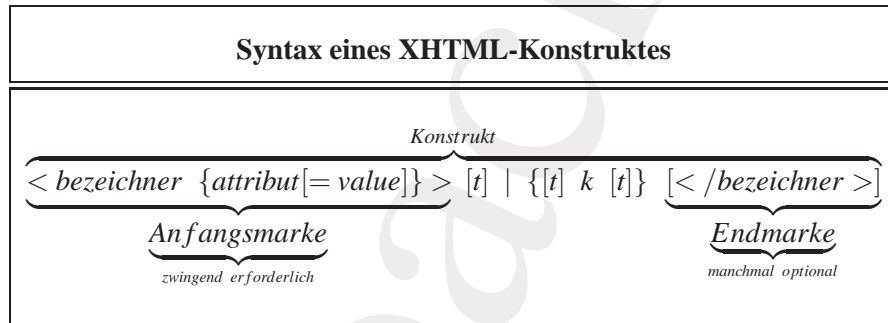
Einige Konstrukte haben keine Endemarke oder diese Marke kann entfallen. Zusätzlich zum Bezeichner können Marken Attribute (Argumente) haben, denen über ein Gleichheitszeichen ein Wert zugewiesen werden kann. Der Wert ist in doppelte Hochkommata einzuschließen.¹¹ Bei der Angabe eines Wertes wird Groß/Klein-Schreibung unterschieden. Die Syntax für ein Konstrukt in XHTML verdeutlicht Tabelle 5.2 S. 129. 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 `<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.

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">
```

¹¹Viele *Browser* benötigen jedoch die doppelten Hochkommata nicht (mehr).

**Legende:**

Notation gemäß Backus-Naur-Form (BNF)

- $\{ \dots \}$ \equiv das Eingeklammerte kann entfallen (optional)
- $\{ \dots \}$ \equiv das Eingeklammerte kann einmal, mehrmals oder gar nicht vorkommen
- $a \mid b$ \equiv Alternative, entweder *a* oder *b*
- bezeichner* \equiv aus Buchstabe(n), manchmal gefolgt von Integerzahl
- attribut* \equiv Attribut aus ASCII-Zeichen
- value* \equiv Wert aus ASCII-Zeichen
- t* \equiv Text aus ASCII-Zeichen
- k* \equiv Konstrukt

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

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

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

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

Näheres \leftrightarrow [Bonin96]

Tabelle 5.2: Syntax eines XHTML-Konstruktes

```

    Java myApplet.class: Mein Logo
</applet>
</p>
...
<body>

```

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 5.3 S. 131 beschreibt Attribute des <object>-Konstruktes.¹² Das Attribut align sollte jedoch entsprechend dem CSS-Konzept (↔ Abschnitt 8.2 S. 384) verwendet werden, das heißt nicht direkt im <object>-Konstrukt sondern im <style>-Konstrukt.

Hinweis: Nicht jeder marktübliche Browser unterstützt alle Attribute (korrekt).

¹²Umfassende, vollständige Beschreibung des <object>-Konstruktes siehe HTML4.0-Spezifikation, zum Beispiel: <http://www.w3.org/TR/REC-html40>

Attribute des <object>-Konstruktes	
<code>classid=uri</code>	Ort der Objekt-Implementation als URI-Angabe.
<code>codebase=uri</code>	Basispfad für die Auflösung der relativen URI-Angaben in <code>classid</code> , <code>data</code> und <code>archive</code> . Bei keiner Angabe wird als Ersatzwert die URI-Basis des aktuellen Dokumentes verwendet.
<code>codetype=content-t.</code>	Gibt den erwarteten Datentyp an, wenn ein Objekt, das durch <code>classid</code> spezifiziert wurde, geladen wird. Es ist ein optionales Attribut, das ein Laden eines nicht unterstützten Datentyps vermeiden soll.
<code>data=uri</code>	Gibt den Ort der Objektdaten an, zum Beispiel für Bilddaten.
<code>type=content-t.</code>	Spezifiziert den Datentyp für <code>data</code> . Es ist ein optionales Attribut, das ein Laden eines nicht unterstützten Datentyps vermeiden soll.
<code>archive=uri list</code>	Eine URI-Liste für die Angabe von Archiven, die relevante Quellen für das Objekt enthalten (Trennzeichen in der Liste ist das Leerzeichen.)
<code>standby=text</code>	Text, der angezeigt wird während das Objekt geladen wird.
<code>width=length</code>	Angabe für den <i>User Agent</i> seinen <i>Default</i> -Wert mit der angegebenen Länge (in Pixel) zu überschreiben.
<code>height=length</code>	analog zu <code>width</code>
<code>align=position</code>	Gibt die Objekt-Position bezogen auf den (Kon)Text an. <ul style="list-style-type: none"> • <code>left</code> Linke Rand ist Objekt-Position • <code>right</code> Rechte Rand ist Objekt-Position • <code>bottom</code> Unterkante fluchtet mit aktueller Basisline • <code>middle</code> Mitte fluchtet mit aktueller Basisline • <code>top</code> Oberkante fluchtet mit aktueller Basisline

Legende:

`uri` \equiv *Universal Resource Identifier* (\approx allgemeine Dokumentenadresse)

Tabelle 5.3: Applet-Einbindung: Einige Attribute des <object>-Konstruktes

5.2.3 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“¹³ `<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 8.2 S. 384).

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

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

Listing 5.23: `PruefeApplet.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Testbett fuer Applets -->
4 <!-- Hinrich E. G. Bonin 13-Jan-1997 -->
<!-- Update ... 01-Jun-2007 -->
6 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
8 <base href="http://ics.leuphana.de/" />
<title>JAVA-COACH's Applet-Testbett </title>
10 <meta http-equiv="Last-Modified"
```

¹³Das `<applet>`-Konstrukt wird im HTML4.0-Standard mißbilligt (≡ *a deprecated element*).

```

12 <meta http-equiv="Expires"
    content="31-Dec-2010 00:00:00 GMT" />
14 <meta name="KEYWORDS"
    content="Applet-Beispiele , Arbeiten von H. Bonin" />
16 <meta name="DESCRIPTION"
    content="Bonin , JAVA-COACH" />
18 <link rev=owns
    title="Hinrich E.G. Bonin"
20 href="mailto:bonin@uni-lueneburg.de" />
    <link href="/myStyle.css" rel="stylesheet" type="text/css" />
22 </head>
    <body>
24 <h1>JAVA-COACH's Applet-Testbett</h1>
    <p class="links">
26 <object
        codetype="application/java"
28 codebase="MyApplet"
        classid="java:MyApplet.class"
30 width="400" height="200">
        [Java Applet MyApplet]
32 </object>
        Das nebenstehende Beispiel ist mit
34 <em>MyApplet.class</em> konstruiert
        (<a href="http://as.uni-lueneburg.de/MyApplet/MyApplet.java">
36 Java Quellcode</a>)
        </p>
38 <p class="links">
        <applet code="ImageLoopItem.class"
40 width="179" height="175" align="left"
        alt="Der schnelle Powerman!">
42 <param name="NIMGS" value="5" />
        <param name="IMG" value="IrinaRad" />
44 <param name="PAUSE" value="0" />
        Java Applet ImageLoopItem:
46 Schneller Powerman auf der Radstrecke!
        </applet>
48 </p>
        <p>Copyright Bonin 16-Jan-1997... 01-Jun-2007
50 all rights reserved.</p>
        <address>
52 <a href="mailto:bonin@uni-lueneburg.de">
        bonin@uni-lueneburg.de</a>
54 </address>
    </body>
56 <!-- Ende der Datei PruefeApplet.html -->
    </html>

```

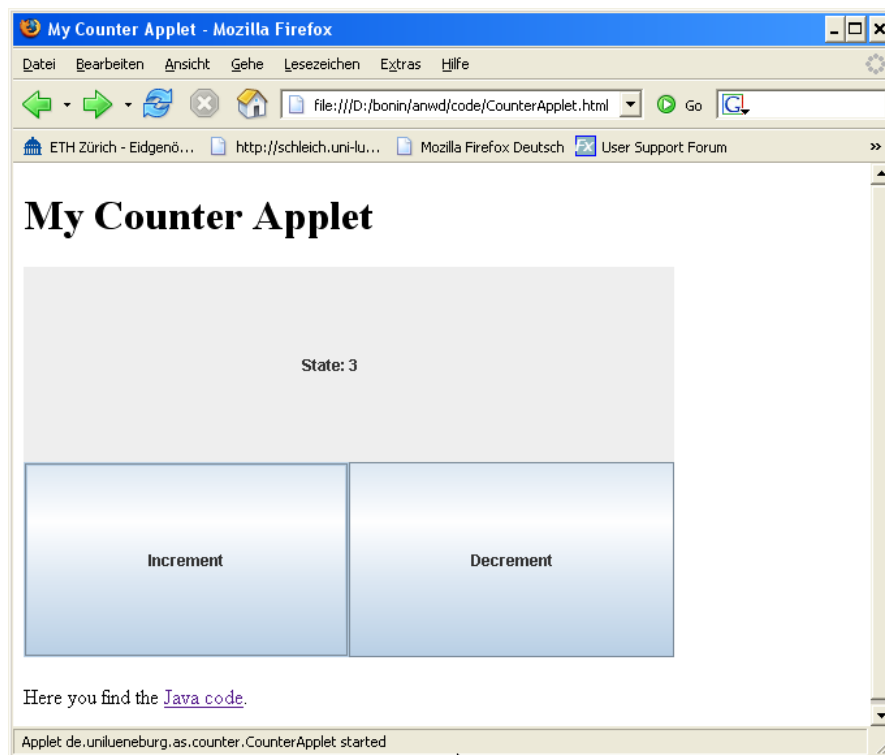


Abbildung 5.15: Beispiel: CounterApplet

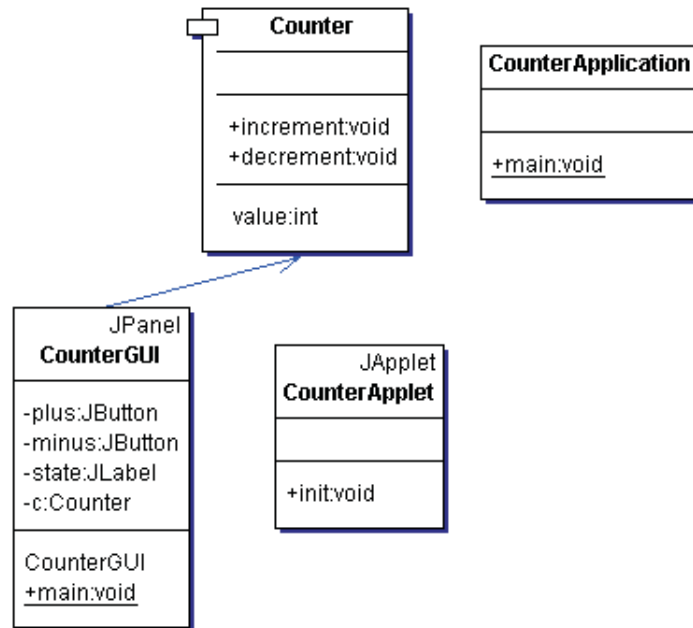
5.2.4 Beispiel CounterApplet.html

Dieses Beispiel entwickelt aus dem als Java-Kostprobe dargestellten kommandozeilengesteuerten Zähler (↔ Abschnitt 5.1.4 Seite 95) einen Zähler mit einer graphischen Benutzungsoberfläche. Diese Klasse CounterGUI (↔ S. 137) wird dann in der Klasse CounterApplet (↔ S. 138) genutzt. Über ein `<object>`-Konstrukt ist das Applet in die XHTML-Datei CounterApplet.html (↔ S. 134) eingebunden. Das Klassendiagramm zeigt Abbildung 5.16 S. 135. Die Abbildung 5.15 Seite 134 zeigt das Ergebnis im Browser *Mozilla Firefox*¹⁴.

Listing 5.24: CounterApplet.html

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

¹⁴Browser *Mozilla Firefox* Version 1.5.0.8 (Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.8.0.8) Gecko/20061025 Firefox/1.5.0.8)



Legende:

Notation in *Unified Modeling Language* (UML) Class Diagram.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung 5.16: Klassendiagramm für CounterGUI

```

    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4 <!-- Bonin Version 1.0 -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
6 <head>
<meta http-equiv="Content-Type"
8 content="text/html; charset=utf-8" />
<title>My Counter Applet</title>
10 </head>
<body>
12 <h1>My Counter Applet</h1>
<p>
14 <object codetype="application/java"
classid="java:de.leuphana.ics.counter.CounterApplet"
16 code="de.leuphana.ics.counter"
width="500" height="300">
18 [Counter Applet --- enable Java to see this applet]
</object>
20 </p>
<p>Here you find the
22 <a href="file://localhost/D:/bonin/anwd/code/\
de/leuphana/ics/counter/CounterApplet.java">
24 Java code</a>.
</p>
26 </body>
</html>

```

Listing 5.25: Counter

```

/**
2 * Beispiel: Counter
*
4 * @since 29-Oct-2003, 24-Nov-2006, 28-May-2007
* @author Hinrich E. G. Bonin
6 * @version 1.1
*/
8 package de.leuphana.ics.counter;

10 public class Counter
{
12     private int value;

14     public int getValue()
    {
16         return value;
    }
18     public void setValue(int i)
    {
20         value = i;
    }
}

```



```

22     public void increment ()
23     {
24         ++value;
25     }
26     public void decrement ()
27     {
28         --value;
29     }
30 }

```

Die Klasse CounterGUI 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.

swing

Listing 5.26: CounterGUI

```

/**
 2  * Beispiel: Counter
 3  * Idea: Schrader / Schmidt-Thieme 2003 p.6
 4  *
 5  * @since 29-Oct-2003, 24-Nov-2006, 28-May-2007
 6  * @author Hinrich E. G. Bonin
 7  * @version 1.3
 8  */
 9  package de.leuphana.ics.counter;
10
11  import java.awt.event.ActionEvent;
12  import java.awt.event.ActionListener;
13  import java.awt.GridLayout;
14  import javax.swing.JButton;
15  import javax.swing.JFrame;
16  import javax.swing.JLabel;
17  import javax.swing.JPanel;
18
19  public class CounterGUI extends JPanel
20  {
21      private JButton plus;
22      private JButton minus;
23      private JLabel state;
24      private Counter c;
25      CounterGUI ()
26      {
27          c = new Counter ();
28          setLayout(new GridLayout (2, 2));
29          add(new JLabel ("State:␣", JLabel.RIGHT));
30
31          state = new JLabel (String.valueOf(c.getValue ());
32          plus = new JButton ("Increment");

```

```

34     minus = new JButton("Decrement");
36     add(state);
36     add(plus);
38     add(minus);
38
40     plus.addActionListener(new ActionListener()
41     {
42         public void actionPerformed(ActionEvent e)
43         {
44             c.increment();
45             state.setText
46             (
47                 String.valueOf(c.getValue())
48             );
49         }
50     });
50     minus.addActionListener(new ActionListener()
51     {
52         public void actionPerformed(ActionEvent e)
53         {
54             c.decrement();
55             state.setText
56             (
57                 String.valueOf(c.getValue())
58             );
59         }
60     });
62 }
62
64 public static void main(String [] args)
65 {
66     JFrame f = new JFrame("Counter_GUI");
67     f.getContentPane().add(new CounterGUI());
68     f.pack();
69     f.setDefaultCloseOperation
70     (
71         JFrame.DISPOSE_ON_CLOSE
72     );
73     f.setVisible(true);
74 }

```

Listing 5.27: CounterApplet

```

/**
2  * Beispiel: Counter
3  * Idea: Schrader / Schmidt-Thieme 2003 p.6
4  *

```

```

    * @since      29-Oct-2003, 24-Nov-2006, 28-May-2007
    * @author     Hinrich E. G. Bonin
    * @version    1.3
    */
    package de.leuphana.ics.counter;

    import javax.swing.*;

    public class CounterApplet extends JApplet
    {
        public void init()
        {
            getContentPane().add(new CounterGUI());
        }
    }

```

5.2.5 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™-Swing-API (Application Programming Interface) ist dafür die besondere Klasse `JProgressBar` vorgesehen. Dieses einfache Beispiel verdeutlicht Möglichkeiten dieser Klasse. Dazu simuliert das Drücken des Buttons `Working!` einen Programmfortschritt. Die Abbildung 5.20 S. 143 zeigt das Klassendiagramm. Die Abbildung 5.18 S. 142 zeigt das Ergebnis im Browser *Mozilla Firefox*¹⁵ und die Abbildung 5.19 S. 142 im Java Appletviewer.

Listing 5.28: MyProgressBar

```

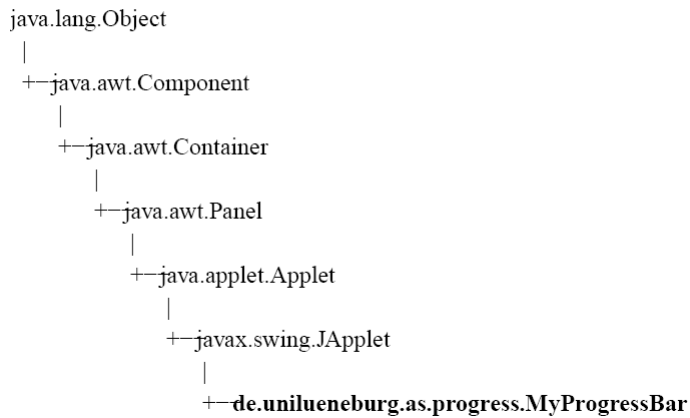
/**
 * Example: Progress Bar
 *
 * @author     Hinrich Bonin
 * @version    1.1 16-Mar2004 24-Nov-2006
 */
    package de.unilueneburg.as.progress;

    import java.awt.Container;
    import java.awt.Color;
    import java.awt.FlowLayout;
    import javax.swing.BorderFactory;
    import javax.swing.JApplet;
    import javax.swing.JProgressBar;

```

¹⁵Browser *Mozilla Firefox* Version 1.5.0.8 (Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.8.0.8) Gecko/20061025 Firefox/1.5.0.8)

de.unilueneburg.as.progress
Class MyProgressBar



Legende:

Dokument erzeugt mittels *Borland Together Control Center*TM 6.2.

Abbildung 5.17: Beispiel: MyProgressBar — Klassenhierarchie

```

import javax.swing.JButton;
16 import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
18 import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
20
public class MyProgressBar extends JApplet
22 {
    JProgressBar jpb;
24     JButton jb;

    public void init()
26     {
28         Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

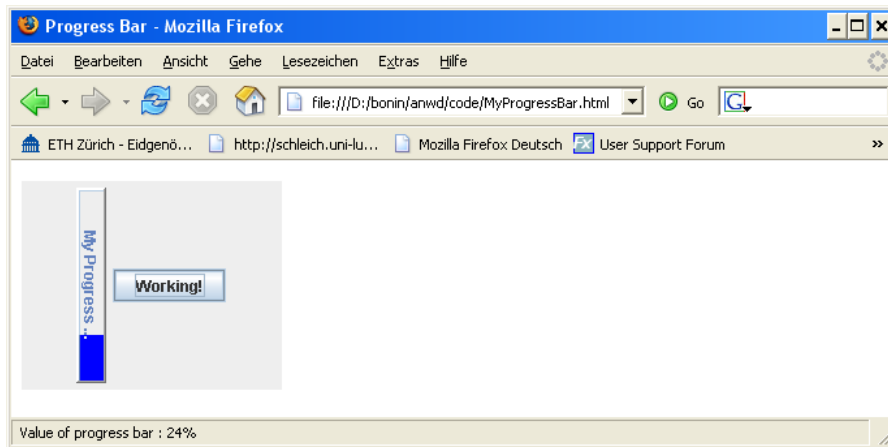
30         jpb = new JProgressBar();

32         // Note the default orientation is horizontal
34         jpb.setOrientation(JProgressBar.VERTICAL);
        jpb.setValue(2);
36         jpb.setForeground(Color.blue);
        jpb.setStringPainted(true);

```

```
38     jpb.setBorder
39         (
40             BorderFactory.createRaisedBevelBorder()
41         );
42     jpb.setString("My Progress Bar");
43     contentPane.add(jpb);
44
45     jb = new JButton("Working!");
46     contentPane.add(jb);
47     jb.addActionListener
48         (
49             new ActionListener()
50             {
51                 public void actionPerformed
52                     (
53                         ActionEvent e
54                     )
55                 {
56                     jpb.setValue
57                         (
58                             jpb.getValue() + 2
59                         );
60                 }
61             });
62     jpb.addChangeListener
63         (
64             new ChangeListener()
65             {
66                 public void stateChanged
67                     (
68                         ChangeEvent e
69                     )
70                 {
71                     showStatus
72                         (
73                             "Value of progress bar: " +
74                             jpb.getValue() +
75                             "%"
76                         );
77                 }
78             }
79         );
80     }
81 }
82 }
```

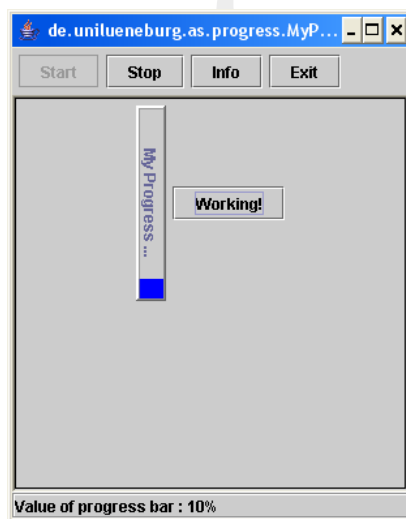
Protokolldatei MyProgressBar.log



Legende:

Java Applet MyProgressBar dargestellt mit *Mozilla Firefox Version 1.5.0.8*

Abbildung 5.18: Beispiel: MyProgressBar — im Browser



Legende:

Java Appletviewer aufgerufen mittels *Borland Together Control Center™ 6.2*.

Abbildung 5.19: Beispiel: MyProgressBar — im Appletviewer

Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center™ 6.2*.

Abbildung 5.20: Klassendiagramm für MyProgressBar

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode)

D:\bonin\anwd\code>javac
de\unilueneburg\as\progress\MyProgressBar.java

D:\bonin\anwd\code>dir
de\unilueneburg\as\progress\*.class

    762 MyProgressBar$1.class
    1.025 MyProgressBar$2.class
    1.634 MyProgressBar.class

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

Listing 5.29: MyProgressBar.html

```
<?xml version="1.0" encoding="utf-8" ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4 <!-- My progress bar example -->
  <!-- Bonin 16-Mar-2004 -->
6 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
  <head>
8   <title>Progress Bar</title>
  </head>
10 <body>
  <p>
```

```

12 <object codetype="application/java"
    classid="java:de.unilueneburg.as.progress.MyProgressBar"
14   code="de.unilueneburg.as.progress"
    width="200" height="160">
16   [Java Applet MyProgressBar]
    </object>
18 </p>
    </body>
20 </html>

```

5.3 Syntax & Semantik & Pragmatik

Die Tabelle 5.4 S. 145 nennt die in Java™ 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™ werden „zusammengesetzte“ Objekte (*ReferenzType*) von einfachen Datentypen (*PrimitiveType*) unterschieden. Die Tabelle 5.5 S. 146 zeigt anhand einer rekursiven Beschreibung die unterschiedlichen Typen.

5.3.1 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 Zugriffsrechte¹⁶ (Sichtbarkeit) sind es folgende Attribute:

- `static`
 - *Variable*: Die Variable ist eine Klassenvariable (hat „Speicherplatz“ nur in der Klasse) und wird durch den Klassennamen angesprochen.

¹⁶↔ Tabelle 5.7 S. 150

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-)Implementation
new	Erzeugt neues Objekt / Array	null	„kein Objekt“-Referenz
operator	— nicht genutzt —	outer	— nicht genutzt —
package	Paket; erste Anweisung	private	Zugriffsrecht
protected	Zugriffsrecht	public	Zugriffsrecht
rest	— nicht genutzt —	return	Kontrollkonstrukt; Rückgabe
short	einfacher Datentyp (16 Bit Zahl)	static	Klassen-Variable / -Methode
super	Zugriff auf Super-Klasse	switch	Kontrollk.; Fallunterscheidung
synchronized	Sperremechanismus	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 5.4: Reservierte Java™ -Schlüsselwörter

– *Methode*: Die Methode ist eine Klassenmethode und ist implizit `final`. Sie wird durch den Klassennamen aufgerufen.

- `abstract`

- *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 (Signatur¹⁷) 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.

¹⁷Signatur besteht aus Namen, Anzahl und Typ der Parameter

<pre> 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[] </pre>
--

Legende:

- `terminal` ≡ definiert als
 - `nonterminal` ≡ ist noch weiter zu definieren
 - `a | b` ≡ a oder b
- Weitere Definitionen ↔ [Arnold/Gosling96]

Tabelle 5.5: Datentypbeschreibung anhand von Produktionsregeln

Typ	Enthält	Standard	Größe in Bit	Minimal	Werte 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	$\pm 3.40282347E+38$	$\pm 1.40239846E-45$
double	Fließkomma IEEE 754	0.0	32	$\pm 1.79769313486231570E+308$	$\pm 4.94065645841246544E-324$

Quelle: [Flanagan96] Seite 183

Hinweis: Die Größe in Bit ist nicht implementationsabhängig!

Tabelle 5.6: Javas einfache Datentypen

- *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

- *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.

5.3.2 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 5.7 S. 150 beschreibt ihre Wirkungen. Praxisrelevante Zugriffssituationen zeigt die Tabelle 5.8 S. 151.

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. <ul style="list-style-type: none"> • Eine Klasse kann <u>nicht</u> 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. <ul style="list-style-type: none"> • Eine Klasse kann <u>nicht</u> 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. <ul style="list-style-type: none"> • Eine Klasse kann <u>nicht</u> als private protected gekennzeichnet werden.

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

Tabelle 5.7: Java-Zugriffsrechte für Klasse, Schnittstelle, Variable und Methode

Lfd. Nr.	Situation	Gekennzeichnet 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 5.8: Zugriffssituationen

5.3.3 Operator — Priorität und Assoziativität

Die Tabelle 5.9 S. 153 fasst die Operatoren mit ihrer Priorität und Assoziativität zusammen (Quelle: ↔ [Schader+03] S. 588)

Java-Coach

Operator	Funktion	Pri- ori- tät	Asso- ziati- vität
.	Zugriff auf Variable oder Methode	14	l
[]	Indexoperator	14	l
()	Funktionsaufruf	14	l
+, -	Vorzeichen	13	r
++, --	Inkrement, Dekrement	13	r
~	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 ≡ Links-assoziativ

r ≡ Rechts-assoziativ

Ein weiter oben stehender Operator hat höhere Priorität.

Tabelle 5.9: Operator — Priorität und Assoziativität

Java-Coach

Kapitel 6

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 157 ...
- die Behandlung von Ereignissen (Delegationsmodell),
hookrightarrow Seite 167 ...
- die Realisierung von persistenten Objekten,
↪ Seite 182 ...
- das Schachteln von Klassen ineinander (*Inner Classes*),
↪ Seite 195 ...
- die Möglichkeit auf die innere Struktur einer Klasse zuzugreifen
(*Reflection*),
↪ Seite 216 ...
- das Arbeiten mit einem objekt-orientierten Datenbanksystem am
Beispiel `FastObjects` t7 der Firma Poet Software
(↪ <http://www.fastobjects.de> Zugriff 18.12.2001),
↪ Seite 233 ...
- die Zusicherung eines bestimmten Wertes für eine Variable
↪ Seite 249 ...
- das Zusammenarbeiten verteilter Objekte (*Stub, Skeleton, RMI*),
↪ Seite 257 ...
- die Verarbeitung von XML-Daten (JDOM),
↪ Seite 282 ...
- die Abbildung einer Komposition mittels Interface und
↪ Seite 56 ...
- die Modelle Komponenten zu spezifizieren und zu verteilen (*Java-BeansTM & EJB*).
↪ Seite 312 ...

6.1 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:

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

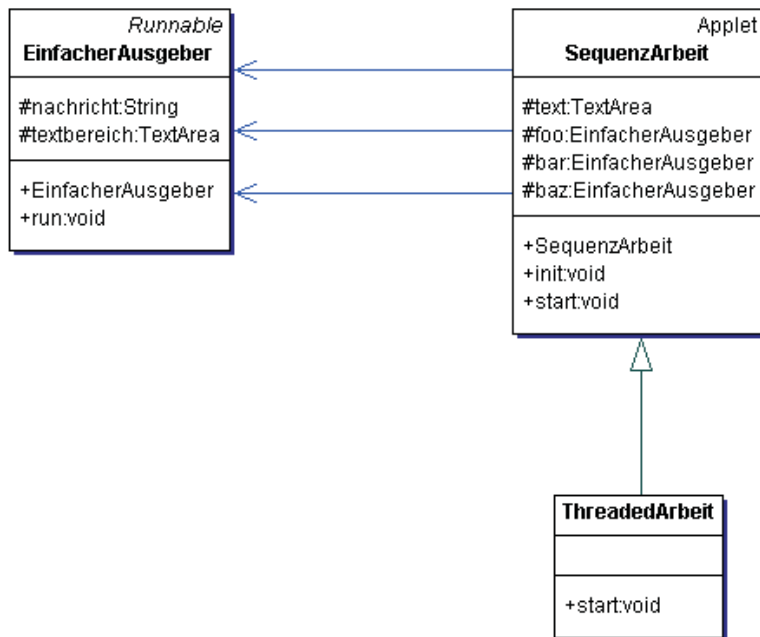
- Objekt einer Klasse vom interface `Runnable` **Runnable**
Ein Objekt einer Klasse, die das Interface `Runnable` implementiert, bildet das nebenläufig abarbeitbare Objekt. Ein solches Objekt `myThread` wird folgendermaßen erzeugt und gestartet:

```
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

Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center™ 6.2*.

Abbildung 6.1: Klassendiagramm für das Multithreading-Beispiel „Textausgabe“

Runnable-Objektes kann das zugehörige **Thread**-Objekt mit Hilfe der Klassenmethode `currentThread()` 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 159). Danach erfolgt die Ausgabe mit Hilfe von drei *Threads*, für jede Zeile ein eigener *Thread* (↔ Seite 163). In beiden Fällen wird dazu die Klasse **EinfacherAusgeber** benutzt (↔ Seite 160). Dieses einfache Beispiel verdeutlicht, daß eine sequentiell konzipierte Lösung durchaus noch in eine nebenläufige verwandelt werden kann. Die Abbildung 6.1 S. 158 zeigt das Klassendiagramm für diese Multithreading-Beispiel.

Listing 6.1: Teil des Dokumentes `SequenzArbeit.html`

```

<object
2   codetype="application/java"
   classid="java:de.unilueneburg.as.parallel.SequenzArbeit.class"
4   name="SeqArbeit"
   width="500"
6   height="600"
   alt="Kleiner Scherzber SAP.">
8   Java SequenzArbeit.class
</object>

```

Listing 6.2: SequenzArbeit

```

2  /**
   * Sequentielles Abarbeiten in einem Applet! Idee bzw. Code
4  * ähnlich: Doug Lea; Concurrent Programming in Java, ISBN
   * 0-201-63455-4
6  *
   * @author      Hinrich E. G. Bonin
8  * @version    1.1
   * @since      13-Jan-1998 20-Nov-2006
10 * /

12 package de.unilueneburg.as.parallel;

14 import java.applet.Applet;
   import java.awt.TextArea;

16
   public class SequenzArbeit extends Applet
18 {
   protected TextArea text;
20 protected EinfacherAusgeber foo;
   protected EinfacherAusgeber bar;
22 protected EinfacherAusgeber baz;

24
   public SequenzArbeit ()
26 {
   /*
28  * Textbereich von 5 Zeile mit 20 Spalten
   */
30 text = new TextArea(5, 20);

32 foo = new EinfacherAusgeber(
   "SAP ist ... \n", text);
34 bar = new EinfacherAusgeber(
   "SAP go go ... \n", text);

```

```

36     baz = new EinfacherAusgeber(
37         "üßTsch... \n", text);
38     }
39
40     public void init()
41     {
42         add(text);
43     }
44
45     public void start()
46     {
47         foo.run();
48         bar.run();
49         baz.run();
50     }
51 }

```

Listing 6.3: EinfacherAusgeber

```

/**
2  * Einfache Ausgabe in einem Fenster! Idee bzw. Code
3  * ä hnlich: Doug Lea; Concurrent Programming in Java, ISBN
4  * 0-201-63455-4
5  *
6  * @author    Hinrich E. G. Bonin
7  * @version   1.1
8  * @since    13-Jan-1998 20-Nov-2006
9  */
10
11 package de.unilueneburg.as.parallel;
12
13 import java.awt.TextArea;
14
15 public class EinfacherAusgeber implements Runnable
16 {
17     /*
18     * Variable üfr die auszugebende Nachricht
19     */
20     protected String nachricht;
21     /*
22     * Variable üfr den Textbereich in den ausgegeben wird
23     */
24     protected TextArea textbereich;
25
26

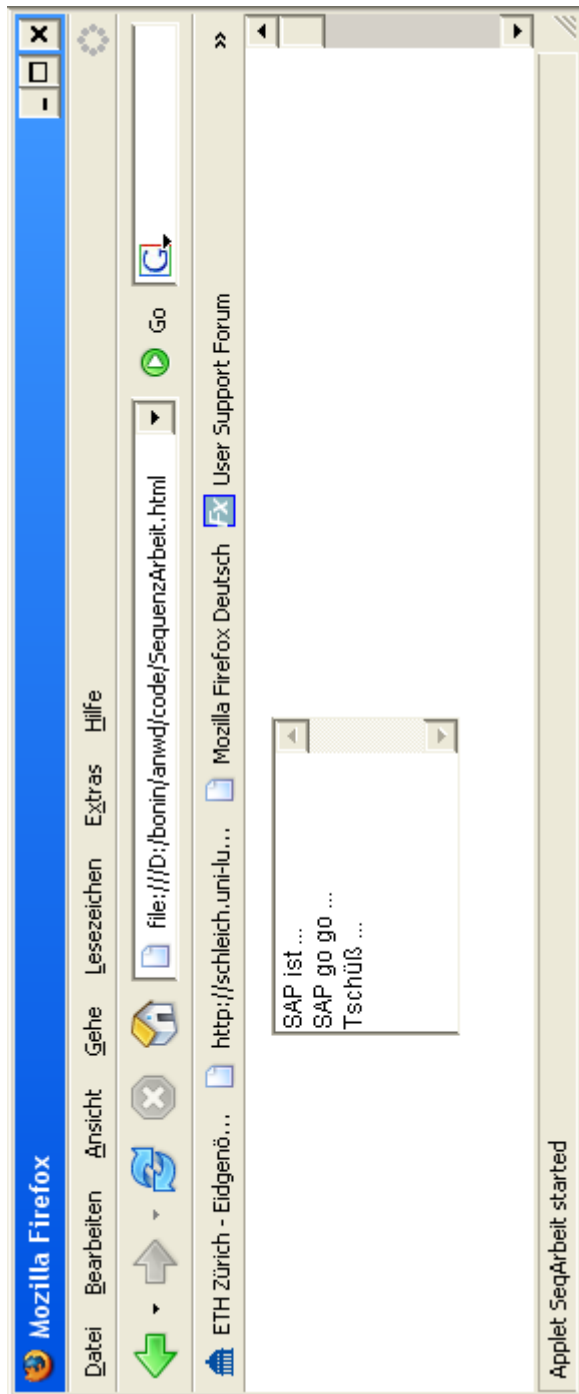
```



```
28     public EinfacherAusgeber(  
        String nachricht, TextArea textbereich)  
30     {  
        this.nachricht = nachricht;  
        this.textbereich = textbereich;  
32     }  
  
34     public void run()  
36     {  
        textbereich.appendText(nachricht);  
38     }  
}
```

Protokolldatei SequenzArbeit.log

```
D:\bonin\anwd\code>java -version  
java version "1.5.0_08"  
Java(TM) 2 Runtime Environment,  
Standard Edition (build 1.5.0_08-b03)  
Java HotSpot(TM) Client VM  
(build 1.5.0_08-b03, mixed mode)  
  
D:\bonin\anwd\code>javac -Xlint de/unilueneburg/as/parallel/*.java  
de/unilueneburg/as/parallel/EinfacherAusgeber.java:37:  
warning: [deprecation] appendText(java.lang.String)  
in java.awt.TextArea has been deprecated  
textbereich.appendText(nachricht);  
^  
  
de/unilueneburg/as/parallel/SequenzArbeit.java:17:  
warning: [serial] serializable class  
de.unilueneburg.as.parallel.SequenzArbeit  
has no definition of serialVersionUID  
public class SequenzArbeit extends Applet  
^  
  
de/unilueneburg/as/parallel/ThreadedArbeit.java:13:  
warning: [serial] serializable class  
de.unilueneburg.as.parallel.ThreadedArbeit  
has no definition of serialVersionUID  
public class ThreadedArbeit extends  
^  
  
3 warnings  
  
D:\bonin\anwd\code>
```



Legende:

Java-Code ↔ Abschnitt 6.2 S. 159

Abbildung 6.2: Ergebnis von SequenzArbeit

Listing 6.4: Teil des Dokumentes ThreadedArbeit

```

<object
2  codetype="application/java"
   classid="java:de.unilueneburg.as.parallel.ThreadedArbeit.class"
4  name="ThreadedArbeit"
   width="500"
6  height="600"
   alt="Kleiner Scherz; ber SAP.">
8  Java ThreadedArbeit.class
</object>

```

Listing 6.5: ThreadedArbeit

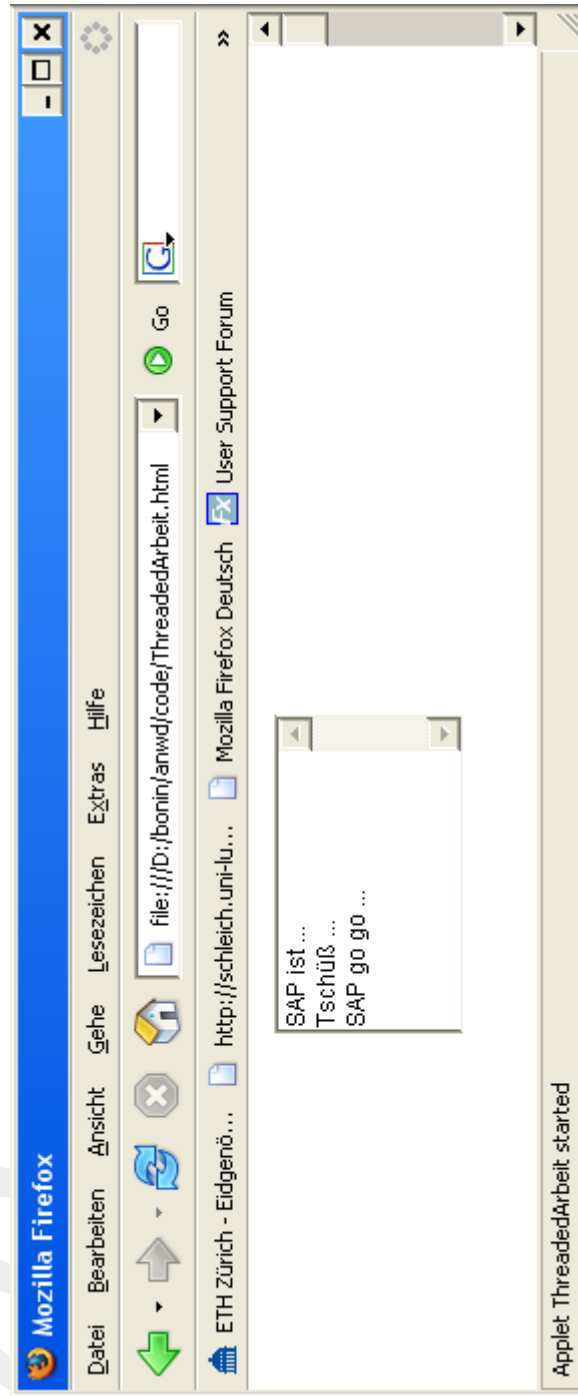
```

/**
2  * ä Nebenlufiges Abarbeiten in einem Applet! Idee bzw. Code
   * ä hnlich: Doug Lea; Concurrent Programming in Java ISBN
4  * 0-201-63455-4
   *
6  * @author      Hinrich E. G. Bonin
   * @version     1.1
8  * @since      13-Jan-1998 20-Nov-2006
   */
10
package de.unilueneburg.as.parallel;
12
public class ThreadedArbeit extends
14     SequenzArbeit
16     {
18         /* Applying implizit:
   *
20         * public ThreadedArbeit ()
   * {
22         *     super ();
   * }
24         */

26     public void start ()
       {
28         new Thread (foo). start ();
         new Thread (bar). start ();
30         new Thread (baz). start ();
       }

```

Java-Coach



Legende:

Java-Code ↔ Abschnitt 6.5 S. 163

Abbildung 6.3: Ergebnis von ThreadedArbeit

6.1.1 Unterbrechung (sleep)

Bei mehreren *Threads* spielt die Aufteilung der Rechenzeit eine große Rolle. Durch das direkte Setzen 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
            }
        }
    }
}
```

6.1.2 Synchronisation (wait(), notify(), synchronized, join)

Das `Thread`-Konzept ist in Java™ 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, daß 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.

6.2 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 Mausklick 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 6.4 S. 168 skizziert als Klassendiagramm dieses Delegationsmodell.

Listener

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

¹Andere Beispiele sind Microsoft's Windows und Unix's Motif.

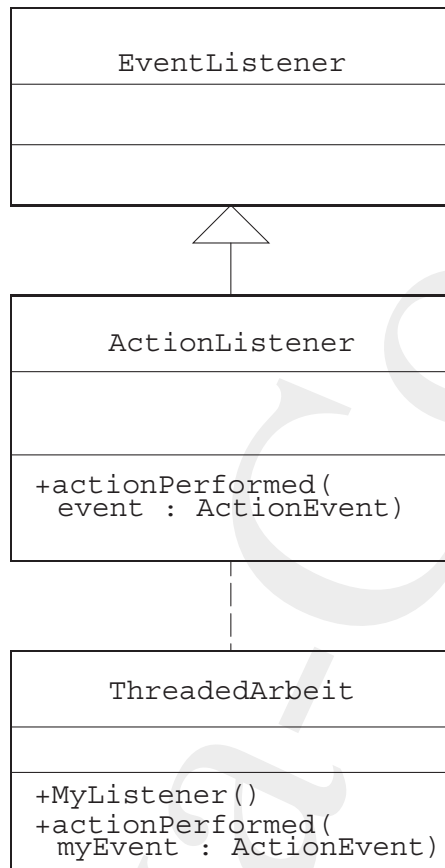


Abbildung 6.4: Java™ AWT: Konstruktion des Delegationsmodells

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 `addActionListener()` wie folgt zugeordnet:

```
public class MyButton extends Button {
    Button anmelden;
    public MyButton()
        // irgendeinen Button definieren, zum Beispiel
        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};
}
}

```

6.2.1 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 (\Leftrightarrow Seite 172). Im zweiten Fall kann durch einen Doppelklick auf eine Liste mit Farbnamen die gewünschte Farbe ausgewählt werden (\Leftrightarrow Seite 173). Für beide Fälle wird nur eine Klasse `SetFarbe`, die das Interface `ActionListener` implementiert, benötigt (\Leftrightarrow Seite 170), 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 6.5 S. 171 zeigt das Klassendiagramm für beide Fälle, also der textlichen Eingabe der gewünschten Farbe beziehungsweise des Doppelklicks auf die gewünschte Farbe.

Listing 6.6: SetFarbe

```

/**
2  * Kleines Beispiel fuer die ActionListener
3  * Idee aus Glenn Vanderburg, et al.;
4  * MAXIMUM JAVA 1.1, pp.230--234
5  * Quellcode modifiziert. Teil 1:
6  * Listener SetFarbe
7  *
8  * @since      27-Apr-1998 21-Nov-2006
9  * @author     Hinrich Bonin
10 * @version    1.2
11 */
12

```



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung 6.5: Klassendiagramm SetFarbe

```

14 package de.unilueneburg.as.farbe;
15
16 import java.awt.event.ActionListener;
17 import java.awt.event.ActionEvent;
18 import java.awt.Color;
19 import java.awt.Label;
20
21 public class SetFarbe implements ActionListener
22 {
23     final String farbname[]
24         = {"rot", "gruen", "blau"};
25
26     final Color farbe[]
27         = {Color.red, Color.green, Color.blue};
28
29     private Label beschriftung;
30
31     public SetFarbe(Label beschriftung)
32     {
33         this.beschriftung = beschriftung;
34     }
35
36     public void actionPerformed(ActionEvent myE)
37     {
38         String name = myE.getActionCommand();
39
40         for (int n = 0; n < farbname.length; n++)
41         {
42             if (farbname[n].equals(name))
43             {
44                 beschriftung.setForeground(farbe[n]);
45                 return;
46             }
47         }
48         System.out.println(
49             "Unbekannter Farbwunsch: " + name);
50     }
51 }

```

Listing 6.7: TextEingabeFarbe

```

/**
2  * Kleines Beispiel fuer
3  * ActionListener Teil IIa:
4  * Farbwahl per Texteingabe
5  *
6  * @since 27-Apr-1998 21-Nov-2006

```

```

    * @author    Hinrich Bonin
8   * @version   1.2
    */
10  package de.unilueneburg.as.farbe;
12
13  import java.awt.Frame;
14  import java.awt.Label;
15  import java.awt.TextField;
16
17  public class TextEingabeFarbe extends Frame
18  {
19      public TextEingabeFarbe ()
20      {
21          Label myL = new Label("Hello World");
22          TextField text = new TextField(10);
23
24          this.add("North", text);
25          this.add("Center", myL);
26          this.pack();
27
28          SetFarbe sf = new SetFarbe(myL);
29          /*
30           * Aktion ist Enter-Taste druecken
31           */
32          text.addActionListener(sf);
33      }
34
35      public static void main(String [] args)
36      {
37          new TextEingabeFarbe().show();
38      }
39  }

```

Listing 6.8: ListWahlFarbe

```

/**
2  * Kleines Beispiel fuer
3  * ActionListener Teil IIb:
4  * Farbliste
5  *
6  * @since    27-Apr-1998 21-Nov-2006
7  * @author   Hinrich Bonin
8  * @version  1.2
9  */
10 package de.unilueneburg.as.farbe;

```



Abbildung 6.6: Ergebnis: java.de.unilueneburg.as.farbe.TextEingabeFarbe

```

12
13 import java.awt.Frame;
14 import java.awt.Label;
15 import java.awt.List;
16
17 public class ListWahlFarbe extends Frame
18 {
19     public ListWahlFarbe ()
20     {
21         Label myL = new Label("Hello World");
22         List myFarbList = new List(3);
23
24         myFarbList.add("rot");
25         myFarbList.add("gruen");
26         myFarbList.add("blau");
27
28         this.add("West", myFarbList);
29         this.add("Center", myL);
30         this.pack();
31
32         SetFarbe sf = new SetFarbe(myL);
33         /*
34          * Aktion besteht im Doppelklick
35          * auf List-Eintragung
36          */
37         myFarbList.addActionListener(sf);
38     }
39
40     public static void main(String argv[])
41     {
42         new ListWahlFarbe().show();

```

```

    }
44 }

```

Protokolldatei SetFarbe.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode)

D:\bonin\anwd\code>javac
  -Xlint de/unilueneburg/as/farbe/*.java
de/unilueneburg/as/farbe/ListWahlFarbe.java:40:
warning: [deprecation] show() in
java.awt.Window has been deprecated
  new ListWahlFarbe().show();
  ^
de/unilueneburg/as/farbe/ListWahlFarbe.java:15:
warning: [serial] serializable class
de.unilueneburg.as.farbe.ListWahlFarbe
has no definition of serialVersionUID
public class ListWahlFarbe extends Frame
  ^
de/unilueneburg/as/farbe/TextEingabeFarbe.java:36:
warning: [deprecation] show() in
java.awt.Window has been deprecated
  new TextEingabeFarbe().show();
  ^
de/unilueneburg/as/farbe/TextEingabeFarbe.java:16:
warning: [serial] serializable class
de.unilueneburg.as.farbe.TextEingabeFarbe
has no definition of serialVersionUID
public class TextEingabeFarbe extends Frame
  ^
4 warnings

D:\bonin\anwd\code>java
  de.unilueneburg.as.farbe.TextEingabeFarbe

D:\bonin\anwd\code>java
  de.unilueneburg.as.farbe.ListWahlFarbe

```

6.2.2 Event→Listener→Method

Für verschiedene Ereignisse gibt es unterschiedliche *Listener* mit fest vorgegebenen Methoden (↔ Tabelle 6.1 S. 177). Das Beispiel:

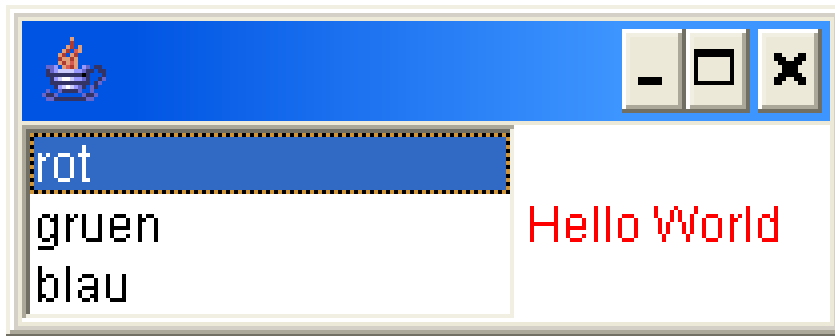


Abbildung 6.7: Ergebnis: `java de.unilueneburg.as.farbe.ListWahlFarbe`

`ZeigeTastenWert` nutzt den `KeyListener` (↔ Seite 176).

Verständlicherweise kann nicht jedes Objekt jedem *Listener* zugeordnet werden; beispielsweise setzt der `WindowListener` ein Objekt der Klasse `Frame` voraus. Tabelle 6.2 S. 178 zeigt die Zuordnung. Die Auslösung des Ereignisses ist ebenfalls abhängig vom jeweiligen Objekt. Zum Beispiel ist es ein Mausklick beim `Button` und ein Mausklick beim `List`-Objekt. Tabelle 6.3 S. 178 zeigt die jeweiligen Aktionen. Darüber hinaus ist bedeutsam, welcher Wert zur Identifizierung des jeweiligen Objekts von `getActionCommand()` zurückgegeben wird. Tabelle 6.4 S. 178 nennt die Werte.

6.2.3 `KeyListener` — Beispiel `ZeigeTastenWert`

Die Abbildung 6.8 S. 179 zeigt das Klassendiagramm.

Listing 6.9: `ZeigeTastenWert`

```

/**
2  * Kleines Beispiel fuer die
   * Tasten-Rueckgabewerte der
4  * Tastatur Idee aus
   * Glenn Vanderburg, et al.; MAXIMUM JAVA
6  * 1.1, pp.239 Quellcode modifiziert.
   *
8  * @author      Hinrich Bonin
   * @since       27-Apr-1998 21-Nov-2006
10 * @version     1.2

```


<i>event</i>	<i>listener</i>	<i>method</i>
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 6.1: Listener-Typen: event→listener→method

GUI <i>object</i>	<i>listener</i>
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 6.1 S. 177

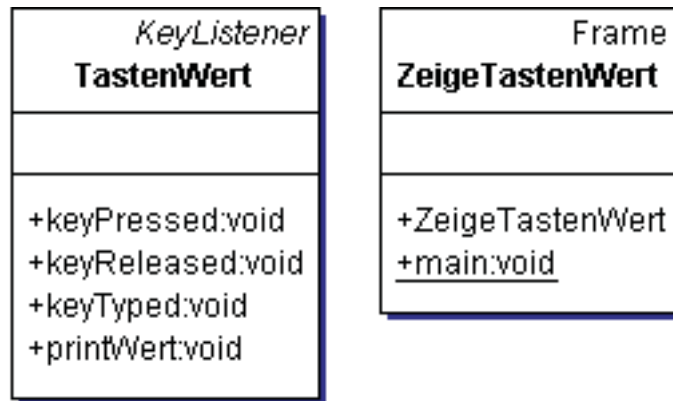
Tabelle 6.2: Object→listener

GUI <i>object</i>	Benutzeraktion
Button	Click auf den Button
List	Doppelclick auf das gewählte Item
MenuItem	Loslassen auf dem MenuItem
TextField	Auslösen der Entertaste

Tabelle 6.3: Benutzeraktion auf das GUI *object*

GUI <i>object</i>	Rückgabewert von <code>getActionCommand()</code>
Button	Text von Button
List	Text vom gewählten Item
MenuItem	Text vom gewählten MenuItem
TextField	Gesamte Text des Feldes

Tabelle 6.4: Rückgabewert von `getActionCommand()`



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center™ 6.2*.

Abbildung 6.8: Klassendiagramm ZeigeTastenWert

```

12 */
13
14 package de.unilueneburg.as.keyvalue ;
15
16 import java.awt.Frame ;
17 import java.awt.Label ;
18 import java.awt.TextField ;
19
20 public class ZeigeTastenWert extends Frame
21 {
22     public ZeigeTastenWert ()
23     {
24         Label myL = new Label("Hello World");
25         TextField text = new TextField(10);
26
27         this.add("North" , text);
28         this.add("Center" , myL);
29         this.pack ();
30
31         TastenWert tW = new TastenWert ();
32         text.addKeyListener (tW);
33     }
34
35     public static void main(String [] args)
36     {
37         new ZeigeTastenWert ().show ();
38     }
39 }
  
```

```

38 }
    }

```

Listing 6.10: TastenWert

```

/**
 2  * Kleines Beispiel fuer die
 3  * Tasten-Rueckgabewerte der
 4  * Tastatur Idee aus
 5  * Glenn Vanderburg, et al.; MAXIMUM JAVA
 6  * 1.1, pp.239 Quellcode modifiziert.
 7  *
 8  * @author      Hinrich Bonin
 9  * @since       27-Apr-1998 21-Nov-2006
10  * @version     1.2
11  */
12
13 package de.unilueneburg.as.keyvalue;
14
15 import java.awt.event.KeyListener;
16 import java.awt.event.KeyEvent;
17 import java.awt.Toolkit;
18
19 public class TastenWert implements KeyListener
20 {
21     public void keyPressed(KeyEvent kE)
22     {
23         if (!Character.isLetter(kE.getKeyChar()))
24         {
25             Toolkit.getDefaultToolkit().beep();
26             kE.consume();
27         }
28         printWert(kE);
29     }
30
31     public void keyReleased(KeyEvent kE)
32     {
33         printWert(kE);
34     }
35
36     public void keyTyped(KeyEvent kE)
37     {
38         printWert(kE);
39     }
40
41     public void printWert(KeyEvent kE)
42     {
43         System.out.println(kE.toString());
44     }

```



Abbildung 6.9: Ergebnis: `java de.unilueneburg.as.keyvalue.ZeigeTastenWert`

```
}
```

Protokolldatei `ZeigeTastenWert.log`

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode)

D:\bonin\anwd\code>javac -Xlint
de/unilueneburg/as/keyvalue/ZeigeTastenWert.java
de/unilueneburg/as/keyvalue/ZeigeTastenWert.java:35:
warning: [deprecation] show() in
java.awt.Window has been deprecated
new ZeigeTastenWert().show();
^
de/unilueneburg/as/keyvalue/ZeigeTastenWert.java:21:
warning: [serial] serializable class
de.unilueneburg.as.keyvalue.ZeigeTastenWert
has no definition of serialVersionUID
public class ZeigeTastenWert extends Frame
^
2 warnings

D:\bonin\anwd\code>java de.unilueneburg.as.keyvalue.ZeigeTastenWert
java.awt.event.KeyEvent [KEY_PRESSED, keyCode=79, keyText=0, keyChar='o',
keyLocation=KEY_LOCATION_STANDARD] on textfield0
java.awt.event.KeyEvent [KEY_TYPED, keyCode=0, keyText=Unknown keyCode:
0x0, keyChar='o', keyLocation=KEY_LOCATION_UNKNOWN] on textfield0
```

```

java.awt.event.KeyEvent [KEY_RELEASED, keyCode=79, keyText=0, keyChar='o',
keyLocation=KEY_LOCATION_STANDARD] on textField0
java.awt.event.KeyEvent [KEY_PRESSED, keyCode=75, keyText=K, keyChar='k',
keyLocation=KEY_LOCATION_STANDARD] on textField0
java.awt.event.KeyEvent [KEY_TYPED, keyCode=0, keyText=Unknown keyCode:
0x0, keyChar='k', keyLocation=KEY_LOCATION_UNKNOWN] on textField0
java.awt.event.KeyEvent [KEY_RELEASED, keyCode=75, keyText=K, keyChar='k',
keyLocation=KEY_LOCATION_STANDARD] on textField0

```

6.3 Persistente Objekte

```

( ) ( )
( ) ( ) +-----+
( % % ) | Gestern |
( @ _ ) | Heute   |
(   )   | Morgen  |
//(   ) \\+-----+
//(   ) \\
vv (   ) vv
(   )
_//~~\\_
( ) ( )

```

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:

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

²Damit ist Objekt-Persistenz zu unterscheiden von einem einfachen Speichern einer Zeichenkette (string) wie es beispielsweise durch die Methoden `save()` und `load`

**Seriali-
zation**

Casting

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, 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
```

Diese Persistenz ist daher nicht hinreichend für Objekte, die sich wie **!Agent** 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 \leftrightarrow [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 (\leftrightarrow 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

der Klasse `java.util.Properties` erfolgt. Dort wird nur der Inhalt der Zeichenkette gespeichert und die zugehörigen Methoden der Klasse `String` werden nicht berücksichtigt.

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 186). Das Schreiben in die Datei `PButton.ser` erfolgt 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

```
ObjectOutputStream out
    = new ObjectOutputStream(
        new GZIPOutputStream(
            new FileOutputStream(
                "./PButton.ser")));
out.writeObject(new PersButton());
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 `readObject()`. 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

```
ObjectInputStream in
    = new ObjectInputStream(
```



```

        new GZIPInputStream(
            new FileInputStream(
                "./PButton.ser"));
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 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™ nach dieser Methode und wendet sie für das Rekonstruieren der Klasse an (⇨ Abbildung 6.11 S. 188).

6.3.1 Serialization — Beispiel PersButton

Listing 6.11: PersButton

```

/**
2  * Kleines Beispiel fuer die ,,Serializing a button'',
  * Grundidee ,,Button'' aus: Glenn Vanderburg, et al.;
4  * MAXIMUM JAVA 1.1, pp.543 jedoch eigene
  * Quellcodestruktur. Teil I: Button-Beschreibng
6  *
  * @since      30-Apr-1998, 29-Oct-2006, 28-May-2007
8  * @author     Hinrich E. G. Bonin
  * @version    1.3
10 */
package de.leuphana.ics.button;

12
import java.io.Serializable;
14 import java.awt.Button;
import java.awt.event.ActionListener;
16 import java.awt.event.ActionEvent;

18 public class PersButton extends Button
    implements ActionListener, Serializable
20 {
    Button myButton;
22
    public PersButton()
24     {
        myButton = new Button("Anmelden");
26         System.out.println("Button_Lerzeugt");
        this.myButton.addActionListener(this);
28     }

30     public void actionPerformed(ActionEvent e)
    {
32         System.out.println("Button_Lgedrueckt");
    }
34 }

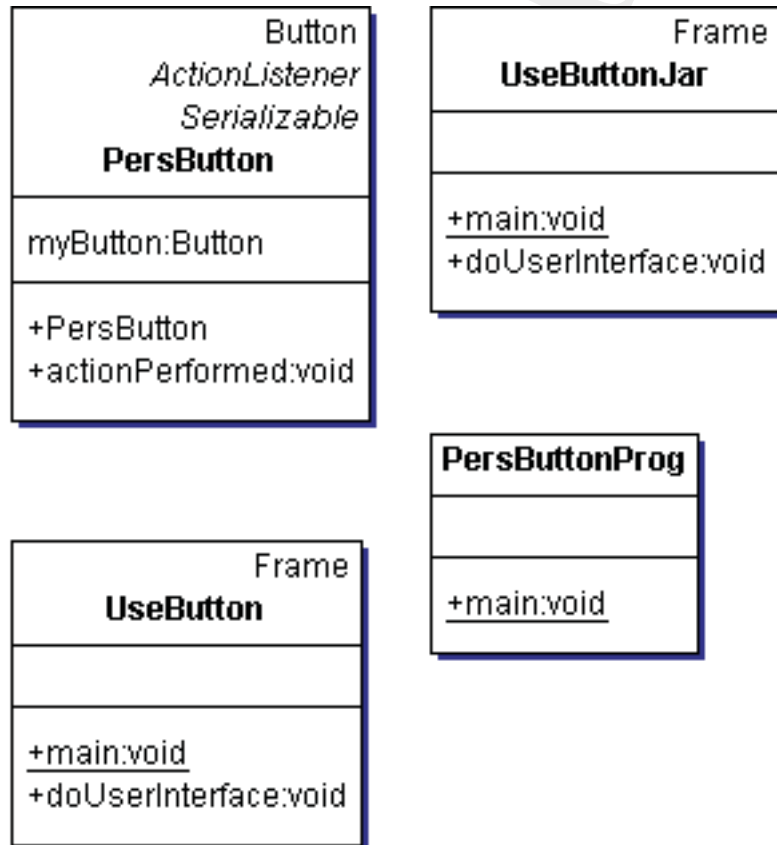
```

Listing 6.12: PersButtonProg

```

/**
2  * Kleines Beispiel fuer die ,,Serializing a
  * button'' Teil II: Schreiben eines Button in
4  * PButton.dat
  *
6  *

```



Legende:

Notation in *Unified Modeling Language* (UML) Class Diagram.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung 6.10: Beispiel PButton

```

8  * @since      30-Apr-1998, 29-Oct-2006, 28-May-2007
9  * @author     Hinrich E. G. Bonin
10 * @version    1.3
11 */
12 package de.leuphana.ics.button;
13
14 import java.io.FileOutputStream;
15 import java.io.ObjectOutputStream;
16 import java.util.zip.GZIPOutputStream;
17
18 public class PersButtonProg
19 {
20     public static void main(String [] args)
21     {
22         try
23         {
24             ObjectOutputStream out
25                 = new ObjectOutputStream (
26                     new GZIPOutputStream (
27                         new FileOutputStream (
28                             ". / PButton . ser ")));
29             out.writeObject(new PersButton());
30             out.close();
31             System.exit(0);
32         } catch (Exception e)
33         {
34             e.printStackTrace(System.out);
35         }
36     }
37 }

```

6.3.2 Rekonstruktion — Beispiel UseButton

Listing 6.13: UseButton

```

1  /**
2   * Kleines Beispiel fuer die
3   * "Use a persistent button object"
4   *
5   * @since      30-Apr-1998, 29-Oct-2006, 28-May-2007
6   * @author     Hinrich E. G. Bonin
7   * @version    1.3
8   */
9  package de.leuphana.ics.button;
10

```

```
import java.io.ObjectInputStream ;
12 import java.io.FileInputStream ;
import java.awt.BorderLayout ;
14 import java.awt.Frame ;
import java.awt.Panel ;
16 import java.util.zip.GZIPInputStream ;

18 public class UseButton extends Frame
{
20     public static void main(String [] args)
    {
22         Frame myFrame =
            new Frame("Willi will ...?");
24         UseButton myUseButton = new UseButton();
        myUseButton.doUserInterface (
26             myFrame, new Panel());
        myFrame.pack();
28         myFrame.setVisible(true);
    }
30
    public void doUserInterface (
32         Frame frame, Panel panel)
    {
34         try
        {
36             ObjectInputStream in
                = new ObjectInputStream (
38                 new GZIPInputStream (
                    new FileInputStream (
40                     "./PButton.ser")));
            PersButton bar
42                 = (PersButton) in.readObject();

44             in.close();
            frame.setLayout(new BorderLayout());
46             panel.add(bar.myButton);
            frame.add("Center", panel);
48         } catch (Exception e)
            {
50             e.printStackTrace(System.out);
            }
52     }
}
```

Protokolldatei PersButton.log

Legende:

Beispiel *Serialization & Rekonstruktion*

Quelcode PersButton ↔ S. 186; PersButtonProg ↔ S. 186
und UseButton ↔ S. 188.

Abbildung 6.11: Beispiel UseButton

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac
  de/leuphana/ics/button/PersButtonProg.java
```

```
D:\bonin\anwd\code>java
  de.leuphana.ics.button.PersButtonProg
Button erzeugt
```

```
D:\bonin\anwd\code>dir PButton.ser
  696 PButton.ser
```

```
D:\bonin\anwd\code>javac
  de/leuphana/ics/button/UseButton.java
```

```
D:\bonin\anwd\code>java
  de.leuphana.ics.button.UseButton
Button gedrueckt
Button gedrueckt
```

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

6.3.3 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*.
. Üblicherweise werden in einer solchen Archivdatei mit dem Suffix

.jar folgende Dateitypen zusammengefaßt:

- <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:

```
D:\bonin\anwd\code>jar
Syntax: jar {ctxu}[vfmOMi] [JAR-Datei]
      [Manifest-Datei] [-C dir] Dateien ...
Optionen:
-c neues Archiv erstellen
-t Inhaltsverzeichnis für Archiv auflisten
-x benannte (oder alle) Dateien
  aus dem Archiv extrahieren
-u vorhandenes Archiv aktualisieren
-v ausführliche Ausgabe für
  Standardausgabe generieren
-f Namen der Archivdatei angeben
-m Manifestinformationen aus angegebener
  Manifest-Datei einbeziehen
-0 nur speichern; keine
  ZIP-Komprimierung verwenden
-M keine Manifest-Datei für die
  Einträge erstellen
-i Indexinformationen für die
  angegebenen JAR-Dateien generieren
-C ins angegebene Verzeichnis wechseln
  und folgende Datei einbeziehen
```

Falls eine Datei ein Verzeichnis ist, wird sie rekursiv verarbeitet.
Der Name der Manifest-Datei und der Name der Archivdatei müssen
in der gleichen Reihenfolge wie die Flags 'm' und 'f' angegeben werden.

Beispiel 1: Archivieren von zwei Klassendateien
in einem Archiv mit dem Namen classes.jar:

```
jar cvf classes.jar Foo.class Bar.class
```

Beispiel 2: Verwenden der vorhandenen Manifest-Datei
'meinmanifest' und Archivieren aller
Dateien im Verzeichnis foo/ in 'classes.jar':

```
jar cvfm classes.jar meinmanifest -C foo/ .
```

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

In unserem Beispiel bietet sich folgendes jar-Kommando an:

```
D:\bonin\anwd\code>jar -cvfm PButton.jar PButton.mf
PButton.ser de/leuphana/ics/button/PersButton.class
de/leuphana/ics/button/UseButton.class
```

Manifest wurde hinzugefügt.

```
Hinzufügen von: PButton.ser (ein = 694)
```

```
(aus = 699) (komprimiert 0 %)
```

```
Hinzufügen von: de/leuphana/ics/button/PersButton.class
```

```
(ein = 745) (aus = 463) (komprimiert 37 %)
```

```
Hinzufügen von: de/leuphana/ics/button/UseButton.class
```

```
(ein = 1384) (aus = 797) (komprimiert 42 %)
```

```
D:\bonin\anwd\code>erase de\leuphana\ics\button\*.class
```

```
D:\bonin\anwd\code>java -jar PButton.jar
```

```
java -jar PButton.jar
```

```
Button gedrueckt
```

```
Button gedrueckt
```

```
D:\bonin\anwd\code>PButton.jar
```

```
PButton.jar
```

```
REM: Keine Ausgabe auf die Console
```

```
D:\bonin\anwd\code>erase PButton.ser
```

```
D:\bonin\anwd\code>java -jar PButton.jar
```

```
java.io.FileNotFoundException:
```

```
.\PButton.ser (Das System kann die angegebene Datei nicht finden)
```

```
at java.io.FileInputStream.open(Native Method)
```

```
at java.io.FileInputStream.<init>(Unknown Source)
```

```
at java.io.FileInputStream.<init>(Unknown Source)
```

```
at de.unilueneburg.as.button.UseButton.doUserInterface(UseButton.java:36)
```

```
at de.unilueneburg.as.button.UseButton.main(UseButton.java:26)
```

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

```
PButton.ser wird also nicht wie hier die Klassen
```

```
aus dem JAR-File genommen. Man muss ihn erst aus dem
```

```
Archiv extrahieren.
```

Dabei bedeuten die Parameter `cfm`, daß ein neues Archiv mit dem Dateinamen des ersten Argumentes erzeugt wird wobei mit dem zweiten Argument der Name der Manifestdatei angegeben wird. Die Manifestdatei hat üblicherweise einen Namen mit dem Suffix `mf`, hier `PButton.mf`.

Manifest-Datei `PButton.mf`

Main-Class: de.leuphana.ics.button.UseButton

Möchte man neben den Klassen auch eine Datei aus dem Archiv direkt nutzen, dann bedarf es der Feststellung der entsprechenden URL mittels Aufruf der Methode `getResource(filename)`. Das folgende Beispiel `FileApp` zeigt die entsprechenden Java-Zeilen sowie die übliche Kurzform dafür.

Listing 6.14: FileApp

```

/**
2  * Kleines Beispiel fuer die
  * Textdatei aus JAR
4  *
  * @since      03-Nov-2006, 28-May-2007
6  * @author     Hinrich E. G. Bonin
  * @version    1.1
8  */
package de.leuphana.ics.archiv;

10
import java.net.URL;
12 import java.io.InputStream;
import java.io.IOException;
14 import java.util.Scanner;

16 public class FileApp
  {
18     public static void main(String [] args)
        throws IOException
20     {
        /*
22         URL url =
           FileApp.class.getResource("mytext.txt");
24         InputStream stream = url.openStream();
        /*
26         bliche Kurzform üfr diese beiden Zeilen:
        */
28         InputStream stream =
30         FileApp.class.getResourceAsStream(
           "mytext.txt");
32         Scanner in = new Scanner(stream);
           while (in.hasNext())
34             System.out.println(in.nextLine());
        }
36 }

```

Text-Datei mytext.txt

Input aus

der Archivdatei!

...

Sollte im gleichen Verzeichnis sein!

Manifest-Datei FileApp.mf

Main-Class: de.leuphana.ics.archiv.FileApp

Protokolldatei FileApp.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/archiv/FileApp.java

D:\bonin\anwd\code>jar -cvfm FileApp.jar FileApp.mf
  de/leuphana/ics/archiv/mytext.txt
  de/leuphana/ics/archiv/FileApp.class
Manifest wurde hinzugefügt.
Hinzufügen von:
  de/leuphana/ics/archiv/mytext.txt
  (ein = 71) (aus = 70) (komprimiert 1 %)
Hinzufügen von:
  de/leuphana/ics/archiv/FileApp.class
  (ein = 758) (aus = 483) (komprimiert 36 %)

D:\bonin\anwd\code>java -jar FileApp.jar
Input aus
der Archivdatei!
...
Sollte im gleichen Verzeichnis sein!

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

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 *JavaBeansTM* (↔ Abschnitt 6.13.1 S. 313) und *Java EnterpriseBeansTM* (↔ Abschnitt 6.13.2 S. 319).

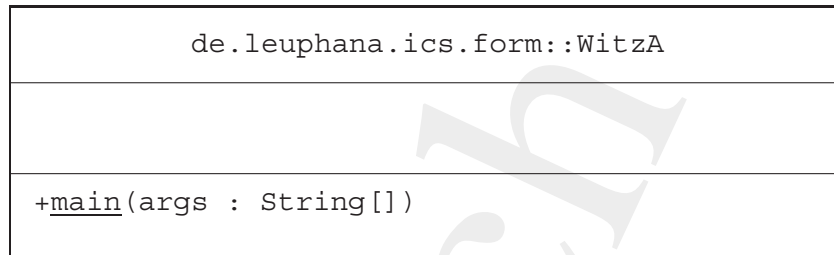


Abbildung 6.12: Klassendiagramm für WitzA

6.4 Geschachtelte Klassen (*Inner Classes*)

In Java™ können Klassen innerhalb von Klassen definiert werden. Um diesen Mechanismus der sogenannten *Inner Classes* zu erläutern, werden einige Konstruktionsalternativen anhand eines sehr einfachen Beispiels gezeigt. Dieses Beispiel gibt den Witztext³ `Piep, piep ... lieb!` als String auf der Systemconsole aus.

Witztext als lokale Variable Zunächst wird eine einfache Klassendefinition `WitzA` (↔ Seite 195) mit einer lokalen Variable betrachtet. In `WitzA` ist innerhalb der Klassenmethode `main()` die lokale Variable `text` initialisiert und anschließend wird sie ausgegeben. Die Abbildung 6.12 S. 195 zeigt das Klassendiagramm.

Listing 6.15: WitzA

```

/**
2  * Kleines Beispiel fuer „Konstruktionsalternativen“
  * hier: Witztext als lokale Variable
4  *
  * @since      26-Nov-2002, 02-Jun-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.2
8  */
package de.leuphana.ics.form;
10
public class WitzA
12 {
    public static void main(String args[])
14 {

```

³Schlagerkurztext von Guildo Horn, Mai 1998

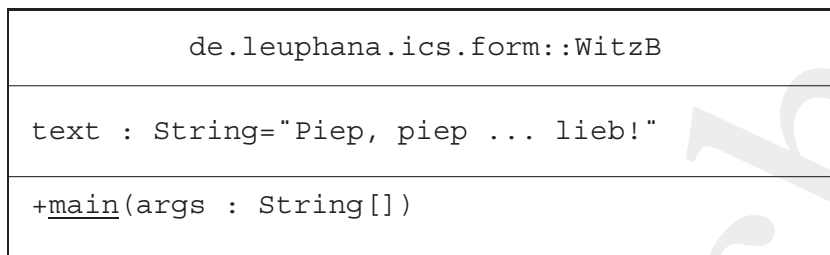


Abbildung 6.13: Klassendiagramm für WitzB

```

16     final String text = "Piep, piep ... lieb!";
17     System.out.println(text);
18 }

```

Witztext als Instanzvariable In WitzB (\leftrightarrow Seite 196) 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 Konstruktor der Klasse, also `WitzB()`, angewendet. Die Abbildung 6.13 S. 196 zeigt das Klassendiagramm.

Listing 6.16: WitzB

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen''
3  * hier: Witztext als Instanzvariable
4  *
5  * @since      26-Nov-2002, 02-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.2
8  */
9  package de.leuphana.ics.form;
10
11 public class WitzB
12 {
13     final String text = "Piep, piep ... lieb!";
14
15     public static void main(String [] args)
16     {
17         System.out.println((new WitzB()).text);
18     }
19 }

```

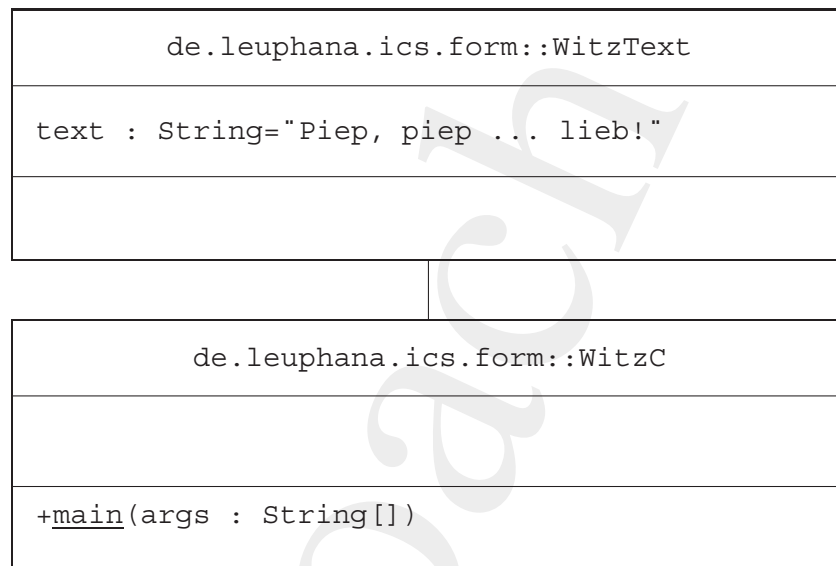


Abbildung 6.14: Klassendiagramm für WitzC

Witztext als Instanzvariablen einer anderen Klasse In WitzC (↔ Seite 197) 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 Datei⁴ `WitzC`. Sie sind im gemeinsamen Paket. Die Abbildung 6.14 S. 197 zeigt das Klassendiagramm.

Listing 6.17: WitzC

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen''
3  * hier: Witztext als Instanzvariable einer
4  * anderen Klasse
5  *
6  * @since 26-Nov-2002, 02-Jun-2007
7  * @author Hinrich E. G. Bonin
8  * @version 1.2
9  */
10 package de.leuphana.ics.form;
  
```

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

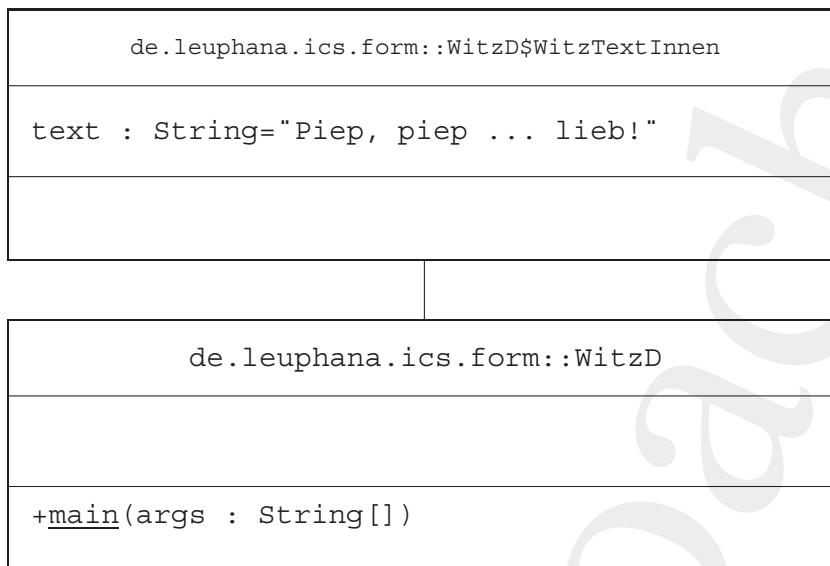


Abbildung 6.15: Klassendiagramm für WitzD

```

12 public class WitzC
13 {
14     public static void main(String [] args)
15     {
16         System.out.println((new WitzText()).text);
17     }
18 }

20 class WitzText
21 {
22     final String text = "Piep, piep ... lieb!";
23 }
  
```

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 199) zeigt diesen Mechanismus der *Inner classes*. Zur Hervorhebung dieser Schachtelung wird die „innere Klasse“ in `WitzTextInnen` umgetauft. Die Abbildung 6.15 S. 198 zeigt das Klassendiagramm.

Listing 6.18: WitzD

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen''
  * hier: Witztext als Member Class
4  *
  * @since      26-Nov-2002, 18-Apr-2006 02-Jun-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.form;

10
public class WitzD
12 {
    public static void main(String [] args)
14 {
        System.out.println (
16         new WitzD ().new WitzTextInnen ().text );
    }
18
    class WitzTextInnen
20 {
        final String text = "Piep, ,piep ,... ,lieb!";
22    }
}

```

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 199). Der Konstruktor der Subklasse `WitzE()` erzeugt eine Instanz, die auch die Variable `text` enthält. Die Abbildung 6.16 S. 200 zeigt das Klassendiagramm.

Listing 6.19: WitzE

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen''
  * hier: Witztext als Instanzvariable einer
4  * Superklasse
  *
6  * @since      26-Nov-2002, 02-Jun-2007
  * @author    Hinrich E. G. Bonin
8  * @version    1.2
  */
10 package de.leuphana.ics.form;

12 public class WitzE extends WitzText
    {

```

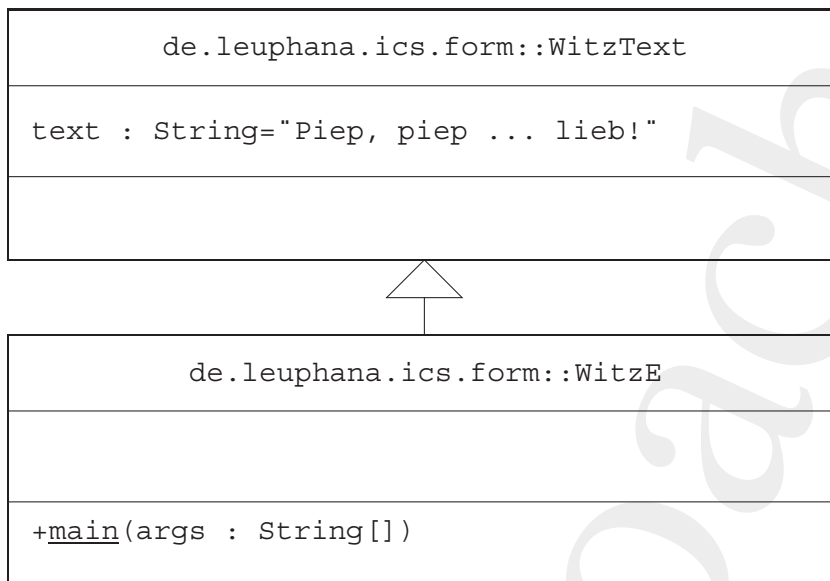


Abbildung 6.16: Klassendiagramm für WitzE

```

14  public static void main(String [] args)
    {
16      System.out.println((new WitzE()).text);
    }
18  }

20  class WitzText
    {
22      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 ihrem Konstruktor eine Instanz erzeugt werden (↔ Seite 200). Außerhalb des Blockes oder der Methode ist keine Instanz erzeugbar. Die Abbildung 6.17 S. 201 zeigt das Klassendiagramm.

Listing 6.20: WitzF

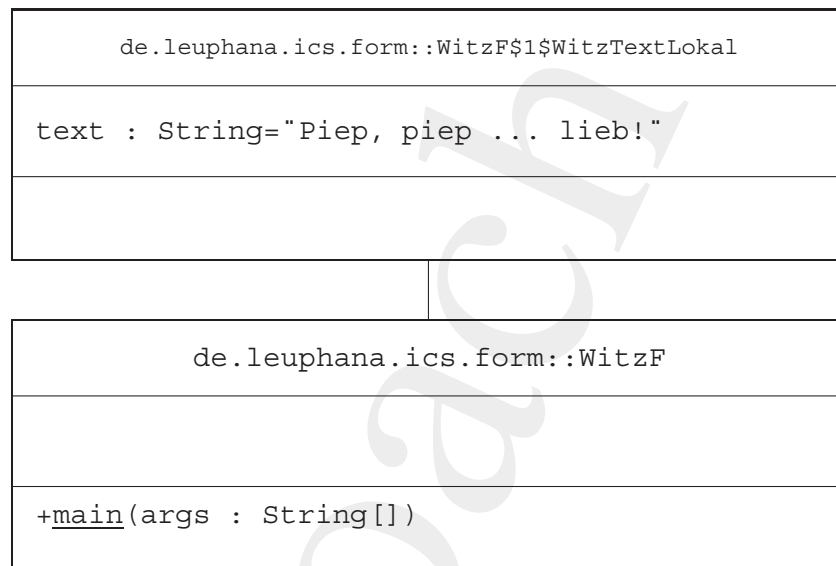


Abbildung 6.17: Klassendiagramm für WitzF

```

/**
2  * Kleines Beispiel fuer „Konstruktionsalternativen“
  * hier: Witztext als lokale Klasse
4  *
  * @since      26-Nov-2002, 18-Apr-2006, 02-Jun-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.form;
10
public class WitzF
12 {
    public static void main(String [] args)
14     {
        class WitzTextLokal
16         {
            final String text =
18             "Piep, piep, ... lieb";
20         }
        System.out.println((new WitzTextLokal()).text);
22     }
}

```

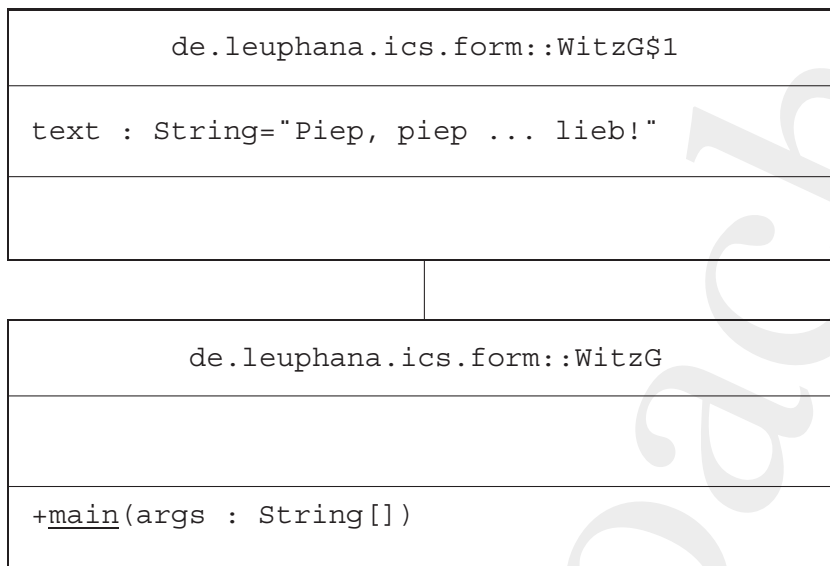


Abbildung 6.18: Klassendiagramm für WitzG

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 (↔ Seite 202). Mit der Ausführung von:

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

wird eine anonyme Klasse erzeugt und instanziiert, die die Klasse `WitzG` spezialisiert, also hier die Methode `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 203). Die Abbildung 6.18 S. 202 zeigt das Klassendiagramm.

Listing 6.21: WitzG

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen''
3  * hier: Witztext als anonyme Klasse
4  *
   * @since      26-Nov-2002, 02-Jun-2007
  
```

```

6  * @author    Hinrich E. G. Bonin
   * @version  1.2
8  */
package de.leuphana.ics.form;

10
public class WitzG
12 {
   public void bar() { }
14
   public static void main(String [] args)
16   {
       WitzG foo =
18       new WitzG ()
       {
20           public void bar ()
           {
22               final String text =
                   "Piep, _piep _... _lieb!";
24               System.out.println (text);
           }
26       };
       foo.bar ();
28   }
}

```

Die Abbildung 6.19 S. 204 zeigt das Klassendiagramm.

Listing 6.22: WitzGa

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen''
   * hier: Witztext als anonyme Klasse – Alternative
4  *
   * @since    26-Nov-2002, 02-Jun-2007
6  * @author   Hinrich E. G. Bonin
   * @version  1.2
8  */
package de.leuphana.ics.form;

10
public class WitzGa
12 {
   public String bar()
14   {
       final String text = "Piep, _piep _... _lieb!";
16       return text;
   }
18
   public static void main(String [] args)
20   {

```

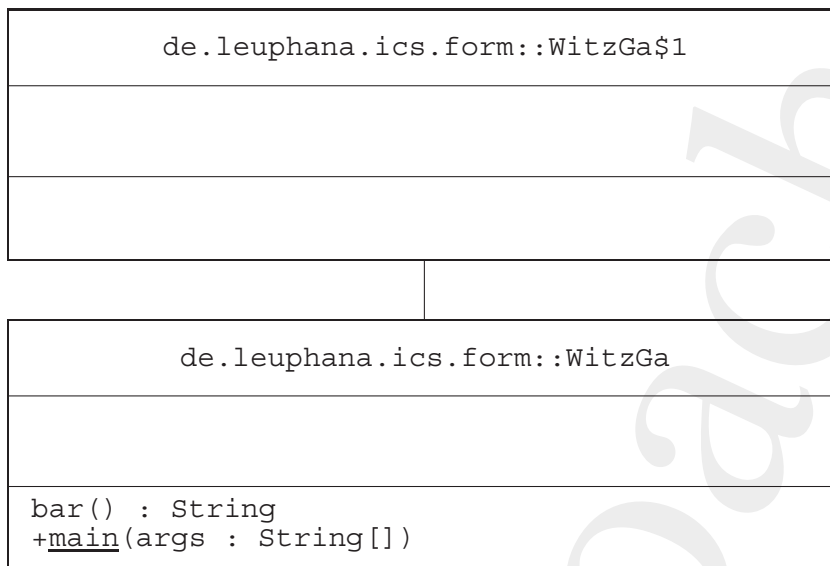


Abbildung 6.19: Klassendiagramm für WitzGa

```

22     WitzGa foo =
        new WitzGa ()
24     {
        public String bar ()
26     {
        System.out.println(
        super.bar ());
28     return "";
        }
30     };
        foo.bar ();
32     }
}

```

Witztext als Klassenvariable derselben Klasse Eine einfache Konstruktion definiert den Witztext als eine Klassenvariable `text` der eigenen Klasse (↔ Seite 204). Die Abbildung 6.20 S. 205 zeigt das Klassendiagramm.

Listing 6.23: WitzH

```
/**
```

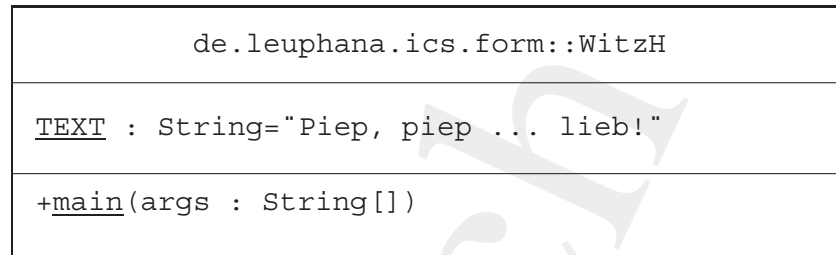


Abbildung 6.20: Klassendiagramm für WitzH

```

2  * Kleines Beispiel fuer ,,Konstruktionsalternativen ''
3  * hier: Witztext als Klassenvariable derselben
4  * Klasse
5  *
6  * @since      26-Nov-2002, 02-Jun-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    1.2
9  */
10 package de.leuphana.ics.form;
11
12 public class WitzH
13 {
14     final static String TEXT
15         = "Piep , piep ... lieb!";
16
17     public static void main(String [] args)
18     {
19         System.out.println(TEXT);
20     }
21 }

```

Witztext als eigene Klasse mit Klassenvariable Etwas aufwendiger ist eine Klassenvariable `text` in einer eigenen Klasse `WitzText` (↔ Seite 205). Die Abbildung 6.21 S. 206 zeigt das Klassendiagramm.

Listing 6.24: WitzJ

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen ''
3  * hier: Witztext als eigene Klasse mit
4  * Klassenvariable
5  *
6  * @since      26-Nov-2002, 02-Jun-2007

```

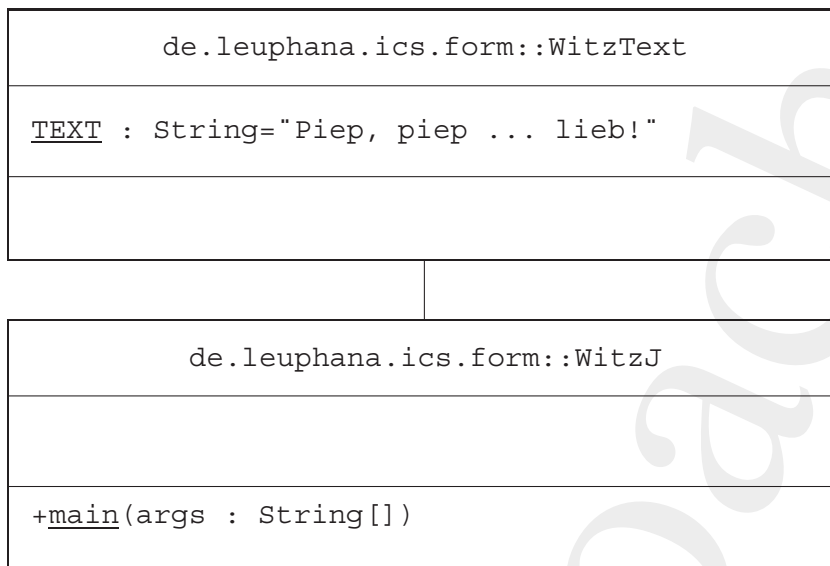


Abbildung 6.21: Klassendiagramm für WitzJ

```

8  * @author      Hinrich E. G. Bonin
   * @version    1.2
   */
10 package de.leuphana.ics.form;

12 public class WitzJ
   {
14     public static void main(String[] args)
       {
16         System.out.println(WitzText.TEXT);
       }
18 }

20 class WitzText
   {
22     final static String TEXT
       = "Piep , piep , ... , lieb!";
24 }

```

Witztext als Klassenvariable der Superklasse Als Klassenvariable TEXT der Superklasse kann aufgrund der Vererbung die eigene Klasse WitzK referenziert werden (↔ Seite 206).

Listing 6.25: WitzK

```

/**
2  * Kleines Beispiel fuer ,,Konstruktionsalternativen''
  * hier: Witztext als Klassenvariable der
4  * Superklasse
  *
6  * @since      26-Nov-2002, 02-Jun-2007
  * @author     Hinrich E. G. Bonin
8  * @version    1.2
  */
10 package de.leuphana.ics.form;

12 public class WitzK extends WitzText
  {
14     public static void main(String [] argv)
        {
16         System.out.println(WitzK.TEXT);
        }
18 }

20 class WitzText
  {
22     final static String TEXT = "Piep, _piep_..._lieb!";
  }

```

Zum Nachweis der Lauffähigkeit der obigen *Witz-Text*-Alternativen ist hier eine Protokolldatei dokumentiert.

Protokolldatei Witz.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzA.java

D:\bonin\anwd\code>java de.leuphana.ics.form.WitzA
Piep, piep ... lieb!

D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzB.java

D:\bonin\anwd\code>java de.leuphana.ics.form.WitzB
Piep, piep ... lieb!

```

208KAPITEL 6. KONSTRUKTIONEN (ANALYSE UND SYNTHESE)

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzC.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzC  
Piep, piep ... lieb!
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzD.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzD  
Piep, piep ... lieb!
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzE.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzE  
Piep, piep ... lieb!
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzF.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzF  
Piep, piep, ... lieb
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzG.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzG  
Piep, piep ... lieb!
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzGa.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzGa  
Piep, piep ... lieb!
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzH.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzH  
Piep, piep ... lieb!
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzJ.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzJ  
Piep, piep ... lieb!
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/form/WitzK.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.form.WitzK  
Piep, piep ... lieb!
```


D:\bonin\anwd\code>

6.4.1 Beispiel Aussen

Nachdem die verschiedenen 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();
```

Listing 6.26: Aussen

```
/**
2  * Beispiel fuer „inner classes“
3  *
4  * @since    26-Nov-2002, 02-Jun-2007
5  * @author   Hinrich E. G. Bonin
6  * @version  1.1
7  */
8  package de.leuphana.ics.deep;

9
10 public class Aussen
11 {
12     public String name = "Aussen";
13
14     public class Innen
15     {
16         public String name = "Innen";
17
18         public class GanzInnen
19         {
20             public String name = "GanzInnen";
21
22             public void printSituation ()
23             {
24                 System.out.println (
25                     name + "\n" +
```

⁵Die Idee zum folgenden Beispiel für „inner classes“ stammt von [Flanagan97] S. 109–110.

```

26         this.name + "\n" +
           GanzInnen.this.name + "\n" +
28         Innen.this.name + "\n" +
           Aussen.this.name);
30     }
31 }
32 }
34 public static void main(String [] args)
35 {
36     Aussen.Innen i = new Aussen().new Innen();
37     Aussen.Innen.GanzInnen g = i.new GanzInnen();
38     g.printSituation();
39 }
40 }

```

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 "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

```

```
D:\bonin\anwd\code>javac de/leuphana/ics/deep/Aussen.java
```

```

D:\bonin\anwd\code>dir de\leuphana\ics\deep\*
1.008 Aussen$Innen$GanzInnen.class
 516 Aussen$Innen.class
  718 Aussen.class
  888 Aussen.java

```

```

D:\bonin\anwd\code>java de.leuphana.ics.deep.Aussen
GanzInnen
GanzInnen
GanzInnen
Innen
Aussen

```

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

6.4.2 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.

Listing 6.27: BlinkLicht.html

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Testbett fuer Applet BlinkLicht.class -->
4  <!-- Bonin 12-May-1998 -->
<!-- Update ... 02-Jun-2007 -->
6  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
8  <title>Blinklicht</title>
<style type="text/css">
10  p.links {
    text-align: left;
12  }
    em {
14  font-size: 28pt;
    font-style: italic;
16  color: #FFFFFF;
    background-color: #000000;
18  }
    body {
20  color: white;
    background-color: #000000
22  }
</style>
24 </head>
<body>
26 <p class="links">
<applet
28  name="Blinker"
    code="de.leuphana.ics.flash.BlinkLicht.class"
30  width="100"
    height="260"
32  standby="Hier_kommt_gleich_ein_Blinklicht!"
    alt="Java Applet BlinkLicht.class">
34  Java Applet BlinkLicht.class
</applet>
36 </p>
<p><em>Gelbes Blinklicht</em> mittels Thread</p>
38 <p>Copyright H.E.G. Bonin 1998 all rights reserved</p>
<address>
40 <a href="mailto:bonin@uni-lueneburg.de">
    bonin@uni-lueneburg.de</a>
42 </address>
</body>
44 <!-- End of File BlinkLicht.html -->
</html>

```

Listing 6.28: BlinkLicht

```
/**
 2  * Kleines Beispiel fuer einen „Thread“, Idee
 3  * aus: Hubert Partl; Java-Einfuehrung, Version
 4  * April 1998, Musterloesungen, S. 33
 5  * http://www.boku.ac.at/javaeinf/
 6  * Quellcode leicht modifiziert
 7  *
 8  * @since      26-Nov-2002, 02-Jun-2007
 9  * @author     Hinrich E. G. Bonin
10  * @version    1.1
11  */
12  package de.leuphana.ics.flash;

13
14  import java.applet.*;
15  import java.awt.*;
16
17  public class BlinkLicht extends Applet
18  implements Runnable
19  {
20      Graphics grafik;
21      Image bild;
22
23      MyCanvas theCanvas;
24
25      Thread myT = null;
26
27      public void init()
28      {
29          setLayout(new FlowLayout());
30          theCanvas = new MyCanvas();
31          theCanvas.setSize(100, 260);
32          add(theCanvas);
33          setVisible(true);
34
35          Dimension d = theCanvas.getSize();
36          bild = theCanvas.createImage(
37              d.width,
38              d.height);
39          grafik = bild.getGraphics();
40      }
41
42      public void start()
43      {
44          if (myT == null)
45          {
46              myT = new Thread(this);
47              myT.start();
48          }
49          System.out.println("start() - appliziert");
50      }
51  }
```

```

50     }
52     public void stop ()
53     {
54         if (myT != null)
55             {
56                 myT.stop ();
57                 myT = null;
58             }
59         System.out.println ("stop () - appliziert");
60     }
62     public void run ()
63     {
64         boolean onOff = false;
65         while (true)
66             {
67                 grafik.setColor (
68                     Color.black);
69                 grafik.fillRect (10,
70                                 10,
71                                 80,
72                                 240);
73                 if (onOff)
74                     {
75                         grafik.setColor (
76                             Color.yellow);
77                         grafik.fillOval (20,
78                                          100,
79                                          60,
80                                          60);
81                     }
82                 onOff = !onOff;
83                 theCanvas.repaint ();
84                 try
85                     {
86                         Thread.sleep (1000);
87                     } catch (InterruptedException e)
88                     {
89                         System.exit (1);
90                     }
91             }
92     }
94     private class MyCanvas extends Canvas
95     {
96         public Dimension getMinimumSize ()
97         {
98             return new Dimension (100, 260);

```

```

    }
100
    public Dimension getPreferredSize()
102    {
        return getMinimumSize();
104    }

    public void paint(Graphics g)
106    {
        update(g);
108    }

    public void update(Graphics g)
110    {
        if ( bild != null)
112        {
            g.drawImage( bild ,
114                        0,
116                        0,
118                        this);
        }
120    }
    }
122 }

```

Protokolldatei BlinkLicht.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

```

```

D:\bonin\anwd\code>javac -Xlint:deprecation
de/leuphana/ics/flash/BlinkLicht.java
de/leuphana/ics/flash/BlinkLicht.java:56:
warning: [deprecation] stop() in
java.lang.Thread has been deprecated
        myT.stop();
        ^

```

1 warning

```

D:\bonin\anwd\code>dir de\leuphana\ics\flash\*
225 BlinkLicht$1.class
1.108 BlinkLicht$MyCanvas.class
2.230 BlinkLicht.class
2.899 BlinkLicht.java

```



Legende:

Java Applet BlinkLicht dargestellt mit *Microsoft Internet Explorer Version: 6.0*

Abbildung 6.22: *Inner class* — Beispiel BlinkLicht

D: \bonin\anwd\code>

↔ Abbildung 6.22 Seite 216

6.5 Interna einer Klasse (*Reflection*)

Das Paket `java.lang.reflect` ermöglicht zusammen mit der Klasse `java.lang.Class` auf die Interna⁶ einer Klasse oder eines Inter-

⁶Interna \approx nur die inneren, eigenen Verhältnisse angehenden Angelegenheiten; vorbehaltenes eigenes Gebiet

faces zuzugreifen. Einige Möglichkeiten dieser sogenannten *Reflection* zeigen die beiden folgenden Beispiele. Die Java-Klasse `ZeigeKlasse` (\leftrightarrow Seite 218) lädt eine Java-Bytecode-Datei (`.class`-Datei) und zeigt die Bestandteile dieser Klasse an. Die Java-Klasse `Aufruf` (\leftrightarrow Seite 225) zeigt das Anwenden einer Klassenmethode (*static method*) und von Instanzmethoden.

6.5.1 `.class`-Datei laden und analysieren

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:

```
>java de.leuphana.ics.reflection.ZeigeKlasse OttoPlan
```

Die Klasse `OttoPlan` erhält man innerhalb von `ZeigeKlasse` als eine Klasse (\equiv Instanz von `Class`) mit folgendem Statement:

```
Class c = Class.forName(args[0]);
```

Das Paket `java.lang.reflect` hat die Klassen `Field`, `Constructor` und `Method` für die Abbildung von Feldern (= Variablen), Constructoren 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.

```
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.

Listing 6.29: `ZeigeKlasse`

```
/**
2  * Kleines Beispiel fuer die „Reflection
3  * API“-Moeglichkeiten Idee von David Flanagan;
4  * Java Examples in a Nutshell, 1997, p. 257
5  * Quellcode modifiziert.
6  *
7  * @since      26-Nov-2002, 03-Jun-2007
8  * @author     Hinrich E. G. Bonin
9  * @version    1.1
10 */
11
12 package de.leuphana.ics.reflection;
13
14 import java.lang.reflect.Constructor;
15 import java.lang.reflect.Field;
16 import java.lang.reflect.Member;
17 import java.lang.reflect.Method;
18 import java.lang.reflect.Modifier;
19
20 public class ZeigeKlasse
21 {
22     String [] myWitz
```

```

22         = new String []{ "Piep",
23                           "piep",
24                           "...",
25                           "lieb!" };
26
27     public static void main(String args [])
28         throws ClassNotFoundException
29     {
30
31         Class c = Class.forName(args [0]);
32         printClass (c);
33     }
34
35     /**
36     *  Gibt von der Klasse die Modifikatoren , den
37     *  Namen, die Superklasse und das Interface
38     *  aus.
39     *
40     * @param c  Description of the Parameter
41     */
42     public static void printClass (Class c)
43     {
44         if (c.isInterface ())
45         {
46             /*
47             *  Modifikatoren enthalten
48             *  das Wort "interface"
49             */
50             System.out.print(
51                 Modifier.toString(
52                     c.getModifiers ())
53                 + c.getName ());
54         }
55         /*
56         *  es gibt kein c.isClass () daher else
57         */
58         else if (c.getModifiers () != 0)
59         {
60             System.out.print(
61                 Modifier.toString(
62                     c.getModifiers ()) +
63                 "┆class┆" +
64                 c.getName () +
65                 "┆extends┆" +
66                 c.getSuperclass ().getName ());
67         } else
68         {
69             /*
70             *  Modifier.toString(0)

```

```

72         * gibt "" zurueck
73         */
74         System.out.print("class_" +
75                             c.getName());
76     }
77
78     /*
79     * Interfaces oder Super-Interfaces
80     * einer Klasse oder eines Interface
81     */
82     Class[] myInterfaces = c.getInterfaces();
83     if ((myInterfaces != null) &&
84         (myInterfaces.length > 0))
85     {
86         if (c.isInterface())
87         {
88             System.out.println(
89                 "_extends_");
90         } else
91         {
92             System.out.print(
93                 "_implements_");
94         }
95         for (int i = 0;
96             i < myInterfaces.length;
97             i++)
98         {
99             if (i > 0)
100            {
101                System.out.print(",");
102            }
103            System.out.print(
104                myInterfaces[i].getName());
105        }
106    }
107
108     /*
109     * Beginnklammer fuer Klassenbody
110     */
111     System.out.println("_{");
112
113     /*
114     * Ausgabe der Felder
115     */
116     System.out.println("_//_Feld(er)");
117     Field[] myFields = c.getDeclaredFields();
118     for (int i = 0; i < myFields.length; i++)
119     {
120         printField(myFields[i]);
121     }

```

```

120
121     /*
122     *  Ausgabe der Konstruktoren
123     */
124     System.out.println("  _/_/_Konstruktor(en)");
125     Constructor [] myConstructors =
126         c.getDeclaredConstructors ();
127     for (int i = 0;
128         i < myConstructors.length;
129         i++)
130     {
131         printMethodOrConstructor (
132             myConstructors[i]);
133     }
134
135     /*
136     *  Ausgabe der Methoden
137     */
138     System.out.println("  _/_/_Methode(n)");
139     Method [] myMethods = c.getDeclaredMethods ();
140     for (int i = 0;
141         i < myMethods.length;
142         i++)
143     {
144         printMethodOrConstructor (
145             myMethods[i]);
146     }
147
148     /*
149     *  Endecklammer fuer Klassenbody
150     */
151     System.out.println("}");
152 }
153
154 /**
155  *  Drucken Methoden und Konstruktoren
156  *
157  *  @param member Description of the Parameter
158  */
159 public static void printMethodOrConstructor
160     (Member member)
161 {
162     Class returnType = null;
163     Class myParameters [];
164     Class myExceptions [];
165     if (member instanceof Method)
166     {
167         Method m = (Method) member;
168         returnType = m.getReturnType ();

```

```

170         myParameters = m.getParameterTypes();
171         myExceptions = m.getExceptionTypes();
172     } else
173     {
174         Constructor c =
175             (Constructor) member;
176         myParameters =
177             c.getParameterTypes();
178         myExceptions =
179             c.getExceptionTypes();
180     }
181     System.out.print("  " +
182         modifiersSpaces(
183             member.getModifiers()) +
184         ((returnType != null) ?
185             typeName(returnType)
186             + " : ") +
187         member.getName() + "(");
188     for (int i = 0;
189         i < myParameters.length;
190         i++)
191     {
192         if (i > 0)
193         {
194             System.out.print(",");
195         }
196         System.out.print(
197             typeName(myParameters[i]));
198     }
199     System.out.print(")");
200     if (myExceptions.length > 0)
201     {
202         System.out.print(" throws");
203     }
204     for (int i = 0;
205         i < myExceptions.length;
206         i++)
207     {
208         if (i > 0)
209         {
210             System.out.print(",");
211         }
212         System.out.print(
213             typeName(myExceptions[i]));
214     }
215     System.out.println(";");
216 }
/**

```

```
218     * Aufbereitung der Modifiers mit Zwischenraeumen
219     *
220     * @param m Description of the Parameter
221     * @return Description of the Return Value
222     */
223     public static String modifiersSpaces (int m)
224     {
225         if (m == 0)
226         {
227             return "";
228         } else
229         {
230             return
231                 Modifier.toString(m) +
232                 " ";
233         }
234     }
235
236     /**
237     * Feld-Ausgabe mit Modifiers und Type
238     *
239     * @param f Description of the Parameter
240     */
241     public static void printField(Field f)
242     {
243         System.out.println (
244             " " +
245             modifiersSpaces (f.getModifiers ()) +
246             typeName(f.getType ()) + " " +
247             f.getName () + ";" );
248     }
249
250     /**
251     * Aufbereitung des Namens mit Array-Klammern
252     *
253     * @param t Description of the Parameter
254     * @return Description of the Return Value
255     */
256     public static String typeName(Class t)
257     {
258         String myBrackets = "";
259         while (t.isArray ())
260         {
261             myBrackets += "[]";
262             t = t.getComponentType ();
263         }
264         return t.getName () + myBrackets;
265     }
266 }
```

Protokolldatei ZeigeKlasse.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/reflection/ZeigeKlasse.java

D:\bonin\anwd\code>java
  de.leuphana.ics.reflection.ZeigeKlasse
  de.leuphana.ics.reflection.ZeigeKlasse
public class de.leuphana.ics.reflection.ZeigeKlasse
  extends java.lang.Object {
  // Feld(er)
  java.lang.String[] myWitz;
  // Konstruktor(en)
  public de.leuphana.ics.reflection.ZeigeKlasse();
  // Methode(n)
  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 void main(java.lang.String[]) throws
    java.lang.ClassNotFoundException;
  public static java.lang.String typeName(java.lang.Class);
}

D:\bonin\anwd\code>

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>

6.5.2 Methode aufrufen

In diesem Beispiel wird eine Methode anhand der Indexangabe ausgewählt.

```
Method m = myMethods[i];
```

Die Referenz `m` verweist auf ein Objekt. Dieses Objekt `m` repräsentiert eine Methode; in diesem Fall wäre es mit $i = 0$ die Instanzmethode `getStart()`.

Um eine so referenzierte Methode auszuführen wird die Methode `invoke()` der Klasse `java.lang.reflect.Method` genutzt. Sie hat folgende Parameter:

```
invoke(java.lang.Object, java.lang.Object[])
```

Der erste Parameter bindet das Objekt auf das die Methode `m` angewendet wird. Der zweite Parameter bindet die Argumente für die Methode `m`. Weil `m` mehr als ein Argument haben kann, wird als „Container“ für den `invoke`-Parameter der Typ *Object Array* verwendet. `invoke()`

Die Methode `invoke()` throws zwei Ausnahmen: `IllegalAccessException` und `InvocationTargetException`. Daher sind entsprechende `try/catch`-Konstrukte zu notieren.

Listing 6.30: Aufruf

```
/**
2  * Example invoke(object, arguments)
  *
4  * @since 23-May-2007
  * @author Hinrich E.G. Bonin
6  * @version 1.0
  */
8
package de.leuphana.ics.reflection;
```

```

10
11 import java.lang.reflect.InvocationTargetException;
12 import java.lang.reflect.Method;

14 public class Aufruf
15 {
16     static int anzahl = 0;

18     String getStart()
19     {
20         return "Start_...";
21     }

22     static String anzahlPlus(String text)
23     {
24         anzahl++;
25         return text;
26     }

28     public static void main(String [] args)
29     {

32         Aufruf myA = new Aufruf ();
33         Class myClass = myA.getClass ();

34         System.out.println ("Class:_ " +
35             myClass.getName ());

38         Method [] myMethods =
39             myClass.getDeclaredMethods ();

40         for (int i = 0; i < myMethods.length; i++)
41         {
42             Method m = myMethods[i];
43             System.out.println ("Method:_ " + m);
44         }

46         // Invocation of static method anzahlPlus ()
47         try
48         {
49             Object [] arguments = new Object []
50                 {"_Working ... "};
51             Object [] noArguments = new Object []{};

54             // Match myA.getStart ()
55             System.out.println ("First_call:_ " +
56                 myMethods[0].invoke (myA, noArguments) +
57                 anzahl);
58             // Match Aufruf.anzahlPlus (" Working ... ")

```

```

        System.out.println("Second call:" +
60         myMethods[1].invoke(null, arguments) +
            anzahl);
62         // Match Aufruf.anzahlPlus(" Working ...")
        System.out.println("Third call:" +
64         myMethods[1].invoke(null, arguments) +
            anzahl);
66         // Match myA.getEnd()
        System.out.println("Fourth call:" +
68         myMethods[2].invoke(myA, noArguments) +
            anzahl);
70     } catch (IllegalAccessException e)
        {
72         System.err.println
            ("Illegal access:" + e);
74     } catch (InvocationTargetException e)
        {
76         System.err.println
            ("Invocation exception:" + e);
78     }
    }
80
    String getEnd()
82     {
        return "End...";
84     }
}

```

Protokolldatei Aufruf.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/reflection/Aufruf.java

D:\bonin\anwd\code>java
  de.leuphana.ics.reflection.Aufruf
Class: de.leuphana.ics.reflection.Aufruf
Method: java.lang.String
  de.leuphana.ics.reflection.Aufruf.getStart()
Method: static java.lang.String
  de.leuphana.ics.reflection.Aufruf.anzahlPlus(java.lang.String)
Method: java.lang.String

```

```

    de.leuphana.ics.reflection.Aufruf.getEnd()
Method: public static void
    de.leuphana.ics.reflection.Aufruf.main(java.lang.String[])
First call : Start ...0
Second call: Working...1
Third call : Working...2
Fourth call: End ...2

D:\bonin\anwd\code>

```

6.6 Referenzen & Cloning

Wenn man eine „Kopie“ eines Objektes mittels einer Zuweisung anlegt, dann verweist die Referenz der Kopie auf dasselbe Originalobjekt. 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) original.clone();
// OK! original bleibt unverändert
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 Struktur 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 Objekte dar. Da nun diese Bits genauso in der Kopie enthalten sind, verweist die Kopie auf dieselben Objekte wie das Original. Es gilt daher drei Fälle beim *Cloning* zu unterscheiden:

Bit → **Bit**

Cloning

1. Problemloses *Cloning*

Die Default-Methode `clone()` reicht aus, weil das Original nur aus primitiven 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 angewendet.

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 Zugriffsrecht `public`. Dabei verhält sich das Interface `Cloneable` anders als ein übliches Interface. Man kann es sich mehr als einen Erinnerungsposten für den Programmierer vorstellen. Er verweist darauf, daß *Cloning* nicht ohne Kenntnis des Kopierprozesses angewendet 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.

Listing 6.31: `Person`

```

/**
2  * Beispiel einer Rekursion innerhalb der Klasse:
3  * Der Ehegatte und die Kinder sind wieder eine
4  * Person
5  *
6  * @since      26-Nov-2002, 03-Jun-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    1.1
9  */
10 package de.leuphana.ics.duplicate;
11
12 import java.util.*;
13
14 class Person implements Cloneable

```

⁷Hinweis: In dem ursprünglichen JDK 1.1.n gab es keine Methode `add(int index, Object element)` in der Klasse `Vector`.

```
{
16  private String name = "";
17  private Person ehEGatte;
18  private Vector kinder = new Vector();

20  public Person (String name)
    {
22      this.name = name;
        System.out.println(name + " lebt!");
24    }

26  public String getName()
    {
28      return name;
    }

30  public Person getEhEGatte ()
    {
32      return ehEGatte;
34    }

36  public Person getKind (int i)
    {
38      return (Person) kinder.get(i - 1);
40    }

42  /*
    *  Achtung!
    *  Set-Metode mit Return-Wert.
44  */
    public Person setName (String name)
    {
46      this.name = name;
48      return this;
    }

50  public void setEhEGatte (Person ehEGatte)
    {
52      this.ehEGatte = ehEGatte;
54    }

56  public void setKind (Person kind, int i)
    {
58      kinder.add(i - 1, (Object) kind);
60    }

62  public Object clone ()
    {
        try
```

```

64         {
65             return super.clone ();
66         } catch ( CloneNotSupportedException e)
67         {
68             System.out.println (
69                 "Objekt_nicht_geklont");
70             return null;
71         }
72     }
73
74     public static void main(String [] args)
75     {
76         Person oE = new Person ("Otto_Eiderente");
77         /*
78          * Applikation der set-Methode mit Reeturn-Wert
79          */
80         Person eE =
81             (new Person (
82                 "Musterperson" )).setName ("Emma_Eiderente");
83
84         oE.setEhegatte (eE);
85         eE.setEhegatte (oE);
86
87         Person madel =
88             new Person ("Emmchen_Eiderente");
89         eE.setKind (madel, 1);
90         oE.setKind (eE.getKind (1), 1);
91
92         Person junge =
93             new Person ("Ottochen_Eiderente");
94         junge.setEhegatte (madel);
95         eE.setKind (junge, 2);
96         oE.setKind (eE.getKind (2), 2);
97
98         System.out.println (
99             "Person_eE:_Ehegatte_von_Ehegatte_ist_" +
100             eE.getEhegatte ().getEhegatte ().getName () +
101             "\n" +
102             "Ehegatte_vom_zweiten_Kind\n" +
103             "_____vom_Ehegatten_ist_" +
104             eE.getEhegatte ().getKind (2).getEhegatte ().getName ());
105
106         /*
107          * Simple Loesung fuer das Problem der
108          * mehrfachen Applikation der
109          * Methode getEhegatte () auf das
110          * jeweilige Resultatobjekt.
111          */
112     }

```

```

114     * Das Clonen (= Bitstromkopierung)
115     * sichert hier nur
116     * die urspruengliche Referenz fuer eE
117     */
118     Person eEclone = (Person) eE.clone();
119     int i;
120     for (i = 1; (i < 4); i++)
121     {
122         eEclone = eEclone.getEhegatte();
123     }
124     System.out.println(
125         "Von_" +
126         eE.getName() +
127         "_ausgehend_immer_wieder_Ehegatte\n" +
128         "_____von_Ehegatte_ist_" +
129         eEclone.getName());
130 }

```

Protokolldatei Person.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

```

```

D:\bonin\anwd\code>javac -Xlint:unchecked
de/leuphana/ics/duplicate/Person.java
de/leuphana/ics/duplicate/Person.java:58:
warning: [unchecked] unchecked call
to add(int,E) as a member of
the raw type java.util.Vector
    kinder.add(i - 1, (Object) kind);
                ^
1 warning

```

```

D:\bonin\anwd\code>java
de.leuphana.ics.duplicate.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>
```

6.7 Integration eines ODBMS — Beispiel Fast-Objects

Als charakteristisches Beispielprodukt für ein objekt-orientiertes Daten- **POET** BankManagementSystem dient im Folgenden das Produkt `FastObjects` der Firma POET Software GmbH, Hamburg⁸. Das Modell der ODMG⁹ **ODMG** (*Object Data Management Group*) für persistente Objekte in einer Datenbank spezifiziert das Erzeugen eines solchen Objektes im Rahmen einer Transaktion.

6.7.1 Transaktions-Modell

Man kreiert ein solches „Transaktionsobjekt“¹⁰ zwischen den Methode `begin()` und `commit()`.

```

import com.poet.odmg.*;
... // Deklaration von Klassen
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`.

⁸Beziehungsweise: POET Software Corporation, San Mateo, USA

⁹ODMG ≡ vormalis: Object Database Management Group,

↪ <http://www.odmg.org/frrbottom.htm> (Zugriff: 27-May-1998)

¹⁰Bei 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.

```

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);
    }
}

```

6.7.2 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:

```
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();

```

lookup (\wiedergewonnen und anschließend mittels einer *Casting*-Operation rekonstruiert.

```
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 6.7.5 S. 237).

6.7.3 Referenzierung & Persistenz

In Java™ 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());
```

ODMG-Collections		
Typ	Sortierung	Duplikate
Bag	vom System bestimmt	Ja
List	selbstgewählt	Ja
Set	vom System bestimmt	Nein
Array	selbstgewählt	Ja

Tabelle 6.5: FastObjects Java Binding Collection Interfaces

Im Regelfall ist beim Speichern eines Objektes ein umfangreiches Netzwerk von referenzierten Objekten betroffen, damit das Originalverhalten des Objektes nach dem Selektieren und Rekonstruieren wieder herstellbar ist.

6.7.4 Collections

Eine *Collection* ermöglicht einem Objekt, eine Sammlung mit mehreren Objekten¹¹ 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());
}
```

Entsprechend der ODMG-Spezifikation unterstützt *FastObjects Java Binding* die *Collections* Bag, List Set und Array. Die Tabelle 6.5 S. 236 zeigt die Unterschiede in Bezug auf die Sortierung der Objekte und auf das mehrfache Vorkommen des gleichen Objektes.

¹¹oder eine Sammlung von Referenzen auf mehrere Objekte

6.7.5 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 Konstruktor `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

```
Extent allePersonen = new Extent(database, "Person");
```

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());
}
```

6.7.6 Transientes Objekt & Constraints

In JavaTM 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;
```

¹²Derzeit definiert weder Java noch das *ODMG Java Binding* ein Konstrukt `Extent`. Es handelt sich dabei (noch?) um eine spezifische `FastObjects`-Leistung.

```

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() {
        // ...
    }
}

```

6.7.7 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 ermöglicht `FastObjects` neben dem automatischen (impliziten) *Resolving* auch ein explizites¹³. Dazu dienen die Methoden `resolve()` und `resolveALL()`.

¹³Ein *explizites Resolving* benötigt eine entsprechende Eintragung in der Konfigurationsdatei.

6.7.8 Abfragesprache (OQL)

Für das Arbeiten mit einer objektorientierten Datenbanken hat die ODMG als Abfragesprache OQL (*Object Query Language*) standardisiert. OQL **OQL** basiert wie SQL¹⁴ auf dem Konstrukt

```
select ... from ... where ...
```

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.*;
```

¹⁴Standard Query Language für eine relationale Datenbank.

```

.
.
.
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());
    }
}
.
.
.

```

6.7.9 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 Dictionary-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

¹⁵ODMG Standard Release 2.0

persistent ist. Für eine persistente Klasse `Foo` ist eine Eintragung in folgender Form notwendig:

```
[classes\Foo]
persistent = true
```

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 Aufspaltung geschieht bei der Eintragung `oneFile = false`.

```
[schemata\myDict]
oneFile = true
```

```
[databases\myBase]
oneFile = true
```

6.7.10 Beispiel: Bind, Lookup und Delete

Das Einführungsbeispiel¹⁶ verdeutlicht 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.

¹⁶Ursprüngliche Quelle: Inhalt des POET-Paketes 5.0
 /POET50/Examples/JavaODMG/First/
 — jedoch leicht modifiziert.

```

set CLASSPATH=%CLASSPATH%;C:\Programme\POET50\Lib\POETClasses.zip;.
ptjavac *.java
java Bind poet://LOCAL/my_base bonin
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

```

Listing 6.32: MyClass

```

/**
2  * Persistence capable class MyClass
 *
4  * @author    Hinrich Bonin
 * @version    1.1
6  */
import com.poet.odmg.*;
8  import com.poet.odmg.util.*;
import java.util.*;
10
11 class MyClass
12 {
13     short s;
14     int i;
15     float f;
16     double d;
17     Object obj;
18     Date aDate;
19     String str;
20     SetOfObject set;
21     BagOfObject bag;
22     ArrayOfObject varray;
23     ListOfObject list;
24     MyClass myObj;

```

```

26  boolean aBool;
    long l;

28  // Objekt e aus java.util.Enumeration
    // wird nicht gespeichert

30

32  transient Enumeration e;

34  public MyClass ()
    {
36      set = new SetOfObject ();
        bag = new BagOfObject ();
38      varray = new ArrayOfObject ();
        list = new ListOfObject ();
40      i = 11;
        s = 12;
42      d = 3.1415926;
        str = "Mein_erstes_POET_Objekt";
44  }

46

48  public String toString ()
    {
50      return str + "\(" +
        Integer.toString(i) + ", " +
        Integer.toString(s) + ", " +
52      Double.toString(d) + ", " +
        aDate.toString() + ")";
54  }
}

```

Listing 6.33: Bind

```

/**
2  * Persistence aware class Bind
   *
4  * @author    Hinrich Bonin
   * @version   1.1
6  */
import com.poet.odmg.*;

8  /*
   * Exception-Import nicht durch org.odmg.* ersetzen,
10 * da dann 2 mal Klasse Database
   */
12 import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;
14 import java.util.*;

```

```

16 public class Bind
17 {
18     Bind(Database db, String name)
19         throws ODMGException
20     {
21         Transaction txn = new Transaction ();
22         txn.begin ();
23         try
24         {
25             MyClass myObject = new MyClass ();
26             db.bind (myObject, name);
27         } catch (ObjectNameNotUniqueException exc)
28         {
29             txn.abort ();
30             throw exc;
31         } catch (ODMGRuntimeException exc)
32         {
33             txn.abort ();
34             throw exc;
35         }
36
37         txn.commit ();
38     }
39
40     public static void main(String [] argv)
41         throws ODMGException
42     {
43         if (argv.length < 2)
44         {
45             System.out.println (
46                 "Bitte _Datenbank_ und _Objektname_ nennen!");
47             System.exit (1);
48         }
49         Database db = new Database ();
50         db.open (argv [0], Database.OPEN_READ_WRITE);
51         try
52         {
53             new Bind (db, argv [1]);
54         } finally
55         {
56             db.close ();
57         }
58     }
59 }
60 }

```

Listing 6.34: Lookup

/**

6.7. INTEGRATION EINES ODBMS — BEISPIEL FASTOBJECTS247

```
2  * Selektieren und Rekonstruieren eines POET-Objektes
3  *
4  * @author    Hinrich Bonin
5  * @version  1.1
6  */
7  import com.poet.odmg.*;
8  // Exception-Import nicht durch org.odmg.* ersetzen,
9  // da dann 2 mal Klasse Database
10 import org.odmg.ODMGException;
11 import org.odmg.ODMGRuntimeException;
12
13 public class Lookup
14 {
15     Lookup(Database db, String name)
16         throws ODMGException
17     {
18         Transaction txn = new Transaction ();
19         txn.begin ();
20         try
21         {
22             MyClass myObject = (MyClass) db.lookup(name);
23             System.out.println(myObject);
24         }
25         // Fuer den Fehlerfall
26         catch (ObjectNameNotFoundException exc)
27         {
28             txn.abort ();
29             throw exc;
30         } catch (ODMGRuntimeException exc)
31         {
32             txn.abort ();
33             throw exc;
34         }
35         txn.commit ();
36     }
37
38     public static void main(String [] argv)
39         throws ODMGException
40     {
41         if (argv.length < 2)
42         {
43             System.out.println (
44                 "Bitte Datenbank und Objektname nennen!");
45             System.exit(1);
46         }
47         Database db = new Database ();
48         db.open(argv[0], Database.OPENREAD_WRITE);
49         try
```

```

52     {
        new Lookup(db, argv[1]);
54     } finally
        {
56         db.close();
        }
58 }

```

Listing 6.35: Delete

```

/**
2  * Selektieren und ölschen eines POET-Objektes
3  *
4  * @author      Hinrich Bonin
5  * @version     1.1
6  */
import com.poet.odmg.*;
8  /**
   * Exception-Import nicht durch org.odmg.* ersetzen,
10 * da dann 2 mal Klasse Database
   */
12 import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;
14
public class Delete
16 {
    Delete(Database db, String name)
18         throws ODMGException
    {
20         Transaction txn = new Transaction();
        txn.begin();
22         try
        {
24             ObjectServices os =
                ObjectServices.of(myDB);
26             os.delete(db.lookup(name));
                db.unbind(name);
28         }
        // Fuer den Fehlerfall
30         catch (ObjectNameNotFoundException exc)
        {
32             txn.abort();
                throw exc;
34         } catch (ODMGRuntimeException exc)
        {
36             txn.abort();
                throw exc;
38         }
    }

```



```

    txn.commit();
40 }
42
43 public static void main(String [] args)
44     throws ODMGException
45 {
46     if (args.length < 2)
47     {
48         System.out.println(
49             "Bitte_Datenbank_und_Objektname_nennen!");
50         System.exit(1);
51     }
52     Database db = new Database();
53     db.open(args[0], Database.OPEN_READ_WRITE);
54     try
55     {
56         new Delete(db, args[1]);
57     } finally
58     {
59         db.close();
60     }
61 }
62 }

```

6.8 Zusicherung über Werte

Wenn man sicherzustellen möchte, dass eine Variable einen bestimmten Wert hat oder eine Wertgrenze einhält, dann kann statt eines `if`-Konstruktes dazu das `assert`-Konstrukt¹⁷ genutzt werden. Zur Laufzeit `assert` wird dann überprüft, ob die *Assertion* erfüllt ist, wenn nicht, dann wird eine Ausnahme ausgelöst. Soll beispielsweise die Instanzvariable `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

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.)

auf das `assert`-Konstrukt wie folgt zu verweisen:

```
> javac -source 1.4 file.java
```

```
> java -enableassertions file
```

Die Parameter `enableassertions` bzw. `disableassertions` können auch in Kurzschreibweise `ea` bzw. `da` angegeben werden.

Die folgende Klasse `Foo` enthält als Beispiel in ihrem *Setter* die Zusicherung, dass der Wert ihrer Instanzvariablen `slot` stets größer null ist.

Listing 6.36: `Foo`

```

2  /**
3   * Zusicherung einer Wertgrenze
4   *
5   * @since      9-Jan-2003, 25-May-2007
6   * @author     Hinrich Bonin
7   * @version    1.2
8   */
9
10 package de.leuphana.ics.value;
11
12 public class Foo
13 {
14     private int slot;
15
16     public int getSlot()
17     {
18         return slot;
19     }
20
21     public void setSlot(int slot)
22     {
23         try
24         {
25             assert slot >= 0;
26             this.slot = slot;
27         }
28         catch (AssertionError e)
29         {
30             System.out.println(
31                 "Error:␣" + e);
32             this.slot = 5;
33         }
34     }
35 }

```

```
public static void main(String[] args)
36 {
    Foo myObject = new Foo();
38
    myObject.setSlot(7);
40
    System.out.println(
42         "Slot value: " +
            myObject.getSlot());
44
    myObject.setSlot(-1);
46
    System.out.println(
48         "Slot value: " +
            myObject.getSlot());
50 }
}
```

Protokolldatei Foo.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac -source 1.4
de.leuphana.ics.value.Foo.java
```

```
D:\bonin\anwd\code>java -enableassertions
de.leuphana.ics.value.Foo
Slot value: 7
Error: java.lang.AssertionError
Slot value: 5
```

```
D:\bonin\anwd\code>java -disableassertions
de.leuphana.ics.value.Foo
Slot value: 7
Slot value: -1
```

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

6.9 Applikation mit großem Speicherbedarf

Als Beispiel für eine Applikation mit einem großen Arbeitsspeicherbedarf dient die Erzeugung eines Bildes im Format JPEG¹⁸ (*Joint Photo-graphics 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:

```
>java -Xms512m -Xmx512m
    de.leuphana.ics.graphic.MyImgStore

-XmsSize setzt den initialen Java™ -Heap-Wert und -XmxSize den
maximalen Java™ -Heap-Wert beim Start der Java Virtual Maschine
(JVM).
```

Listing 6.37: MyImgStore

```
/**
2  * Erzeugung einer JPEG-Graphik;
3  *
4  * @since      16-Jan-2003 25-May-2007
5  * @version    1.2
6  * @author     Hinrich E. G. Bonin
7  */
8  package de.leuphana.ics.graphic;

10 import java.awt.Color;
11 import java.awt.Font;
12 import java.awt.FontMetrics;
13 import java.awt.Graphics2D;
14
15 import java.util.Random;
16
17 public class MyImgStore extends ImgJpegStore
18 {
19     public static void main(String [] args)
20     {
21         try
22         {
23             MyImgStore mis = new MyImgStore ();
24             mis.store (
25                 4096,
26                 4096,
27                 " ./de/leuphana/ics/graphic/myPicture.jpg ");
28         } catch (Exception ex)
```

¹⁸Web-Site ↔ <http://www.jpeg.org/>, Zugriff 16-Jan-2003

```
30     {
31         System.out.println(ex.getMessage());
32         System.exit(1);
33     }
34     System.out.println("Image stored.");
35     System.exit(0);
36 }
37
38 public void myPaintFunction(
39     Graphics2D g,
40     int width,
41     int height,
42     String imgFilename)
43 {
44     Random generator = new Random();
45     String mySymbol = "+";
46
47     g.setFont(new Font("Courier", Font.PLAIN, 10));
48     FontMetrics fm =
49         g.getFontMetrics(g.getFont());
50
51     /*
52      * Damit zwei Zeilen s
53      * dichter zusammenruecken
54      */
55     int heightStep = fm.getHeight() / 5;
56
57     int widthStep = fm.stringWidth(mySymbol);
58
59     g.setColor(Color.red);
60
61     for (int i = 0; i < height;
62         i = i + heightStep)
63     {
64         String s = "";
65
66         for (int j = 0; j < width;
67             j = j + widthStep)
68         {
69             if (generator.nextInt(2) == 0)
70             {
71                 s = s + "␣";
72             } else
73             {
74                 s = s + mySymbol;
75             }
76         }
77         g.drawString(s, 0, i);
78     }
79 }
```

```

78     }
80 }

```

Listing 6.38: ImgJpegStore

```

/**
2  * Erzeugung einer JPEG-Graphik;
3  *
4  * @since      16-Jan-2003 25-May-2007
5  * @version    1.2
6  * @author     Hinrich E. G. Bonin
7  */
8
9  package de.leuphana.ics.graphic;
10
11 import java.io.File;
12 import java.io.FileOutputStream;
13 import java.awt.Graphics2D;
14 import java.awt.image.BufferedImage;
15 import com.sun.image.codec.jpeg.JPEGCodec;
16 import com.sun.image.codec.jpeg.JPEGEncodeParam;
17 import com.sun.image.codec.jpeg.JPEGImageEncoder;
18
19 public abstract class ImgJpegStore
20 {
21     public abstract void myPaintFunction(
22         Graphics2D g,
23         int width,
24         int height,
25         String imgFilename);
26
27
28     public void store(
29         int width,
30         int height,
31         String imgFilename)
32         throws Exception
33     {
34         BufferedImage img =
35             new BufferedImage(width, height,
36                 BufferedImage.TYPE_INT_RGB);
37         myPaintFunction(
38             img.createGraphics(),
39             width, height,
40             imgFilename);
41         try
42         {
43             FileOutputStream out =

```

```
44         new FileOutputStream (
45             new File (imgFilename));
46     JPEGImageEncoder enc =
47         JPEGCodec.createJPEGEncoder (out);
48     JPEGEncodeParam prm =
49         enc.getDefaultJPEGEncodeParam (img);
50     prm.setQuality (1.0f, false);
51     enc.setJPEGEncodeParam (prm);
52     enc.encode (img);
53 } catch (Exception e)
54 {
55     throw new Exception (
56         "\nError: Image storing to " +
57         imgFilename + " failed: " +
58         e.getMessage ());
59 }
60 }
```

Protokolldatei MyImgStore.log

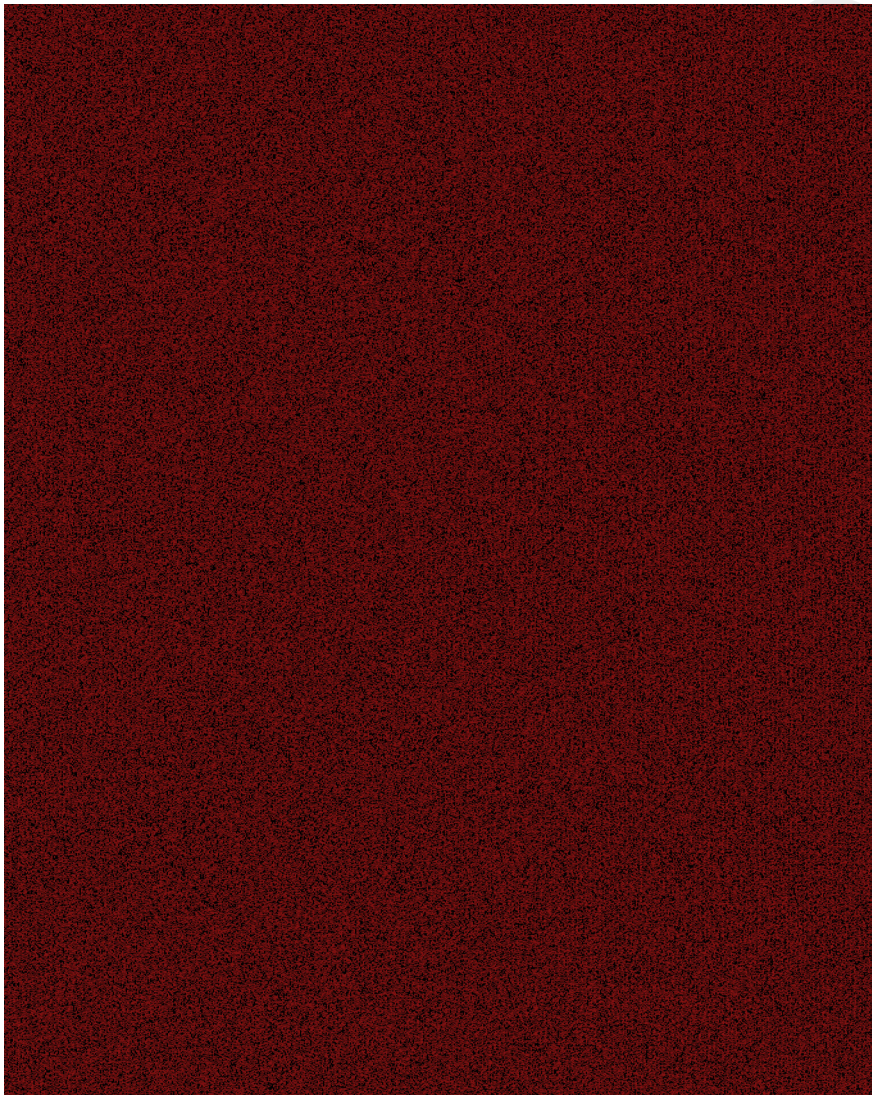
```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de/leuphana/ics/graphic/MyImgStore.java

D:\bonin\anwd\code>java
de.leuphana.ics.graphic.MyImgStore
Exception in thread "main"
java.lang.OutOfMemoryError: Java heap space

D:\bonin\anwd\code>java -Xms512m -Xmx512m
de.leuphana.ics.graphic.MyImgStore
Image stored.

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



Legende:

Quellcode der Klasse `MyImgStore` ↔ S. 252 und der Klasse `ImgJpegStore` ↔ S. 254

Abbildung 6.23: JPEG-Bildbeispiel

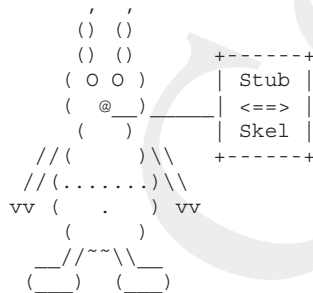
6.10 Verteilte Objekte

Wie funktioniert ein *Distributed Object System*? Mit der Erläuterung dieser Frage befassen sich die beiden Abschnitte:

1. Beispiel Stub¹⁹ & Skeleton²⁰
 ↪ Abschnitt 6.10.1 S. 257
2. Remote Method Invocation Protocol (RMI)
 ↪ Abschnitt 6.10.2 S. 265

Im ersten Schritt wird das grundlegende Prinzip mit einem eigen, primitiven System erläutert. Im zweiten Schritt wird das Prinzip anhand eines RMI-Beispiels vertieft.

6.10.1 Beispiel Stub & Skeleton



Das *Business Object*, das sich auf einem anderen Rechner befindet (Server), also im Hinblick auf den Objektnutzer (Client) verteilt ist, ist ein einfaches Konto. Das Interface `Konto` (↪ S. 258) spezifiziert die Eigenschaften mit den beiden Methoden `getID()` und `getUmsatz()` für beide Seiten, also für Server und Client. Die Klasse `KontoServer` (↪ S. 261) 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 `KontoStub` (↪ S. 258) auf der Client-Seite und die Klasse `KontoSkeleton` (↪ S. 262) auf der Server-Seite. Diese beiden Klassen implementieren das Interface `Konto` und „kennen“ damit die Eigenschaften des Kontos.

Auf der Client-Seite ist die Klasse `KontoStub` quasi ein Ersatz für eine Klasse `Konto`, die ja durch die serverseitige Klasse `KontoServer` abgebildet ist. Die Klasse `KontoStub` spezifiziert eine Netzwerkverbindung auf `Socket`-Basis; hier mit der frei gewählten Portnummer 4711. Wird beispielsweise die Methode `getUmsatz()` für eine In- `Socket`

¹⁹Deutsch ≈ Kontrollabschnitt, (Baum-)Stumpf

²⁰Deutsch ≈ Skelett, Rohbau, Rahmen

stanz 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.

Wrapping

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 myServer)` 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 6.24 S. 259 und auch die Tabelle 6.6 S. 273.

Listing 6.39: Konto

```

/**
2  * Beispiel "Own Distributed Object Protocol"
  *
4  * @since      16-Dec-2002, 12-Jun-2007
  * @author      Hinrich E. G. Bonin
6  * @version     1.1
  */
8  package de.leuphana.ics.distributed;
  /*
10 * Konto-Interface zeigt den Ansatz eines
  * "Business Object"
12 */
14 public interface Konto
  {
16     public String getID() throws Throwable;

18     public int getUmsatz() throws Throwable;
  }

```

Listing 6.40: Konto_Stub

```

/**
2  * Beispiel "Own Distributed Object Protocol"
  *
4  * @since      16-Dec-2002, 12-Jun-2007
  * @author      Hinrich E. G. Bonin
6  * @version     1.1
  */
8  package de.leuphana.ics.distributed;

```

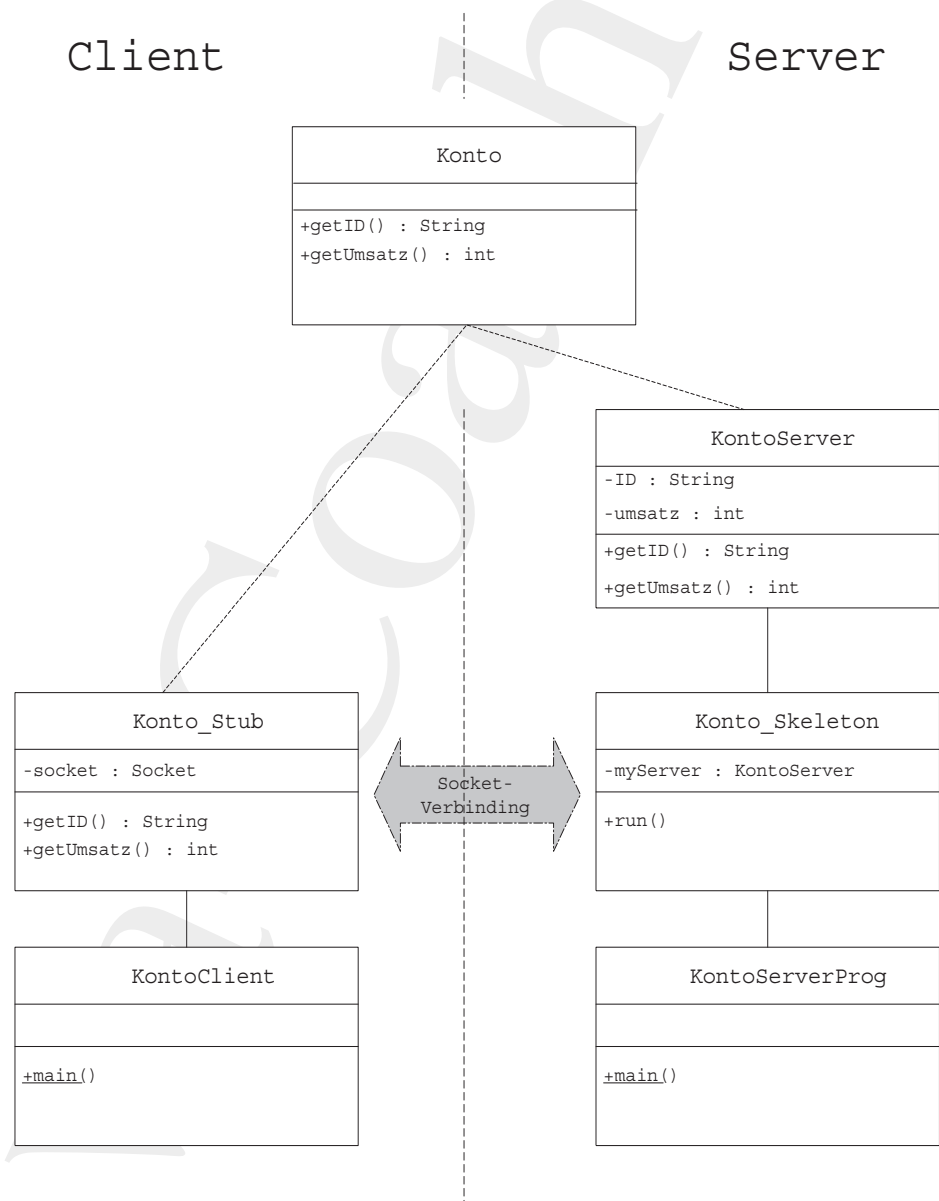


Abbildung 6.24: Own Distributed Object Protocol — Klassendiagramm

```

10 import java.io.ObjectOutputStream ;
11 import java.io.ObjectInputStream ;
12 import java.net.Socket ;
13 /*
14  * Konto_Stub implementiert das Interface Konto
15  * damit es wie ein "Business Object" auf den
16  * Client aussieht.
17  * Es leitet ein Anfrage zum Skeleton auf dem Server.
18  * Das Skeleton sendet diese an das "Business Object"
19  * auf dem Server.
20 */
21 public class Konto_Stub implements Konto
22 {
23     private Socket socket ;
24
25     public Konto_Stub () throws Throwable
26     {
27         /*
28          * Erzeugt eine Netzverbindung zum Skeleton.
29          * Hier "localhost" um auf einem
30          * Rechner zu testen ;
31          * sonst IP-Adresse.
32          */
33         socket = new Socket("localhost" , 4711);
34     }
35
36     /*
37     * Diese Methode schickt einen "Stream"
38     * mit dem Methodennamen zum Skeleton
39     */
40     public String getID () throws Throwable
41     {
42         ObjectOutputStream outputStream =
43             new ObjectOutputStream (
44                 socket.getOutputStream ());
45         outputStream.writeObject ("ID");
46         outputStream.flush ();
47         ObjectInputStream inputStream =
48             new ObjectInputStream (
49                 socket.getInputStream ());
50         return
51             (String) inputStream.readObject ();
52     }
53
54     /*
55     * Diese Methode schickt einen "Stream"
56     * mit dem Methodennamen zum Skeleton
57     */
58     public int getUmsatz () throws Throwable

```

```

60     {
        ObjectOutputStream outStream =
            new ObjectOutputStream (
62         socket.getOutputStream ());
        outStream.writeObject ("umsatz");
64         outStream.flush ();
        ObjectInputStream inStream =
66         new ObjectInputStream (
            socket.getInputStream ());
68         return
            inStream.readInt ();
70     }
    }

```

Listing 6.41: KontoServer

```

/**
2  * Beispiel "Own Distributed Object Protocol"
  *
4  * @since 16-Dec-2002, 12-Jun-2007
  * @author Hinrich E. G. Bonin
6  * @version 1.1
  */
8  package de.leuphana.ics.distributed;
  /*
10  * KontoServer implementiert die "Business Logic"
  * und den "State" üfr ein Konto.
12  */
  public class KontoServer implements Konto
14  {
        private String ID;
16         private int umsatz;

18         public KontoServer (String ID, int umsatz)
        {
20             this.ID = ID;
                this.umsatz = umsatz;
22         }

24         public String getID ()
        {
26             return ID;
        }

28         public int getUmsatz ()
30         {
                return umsatz;
32         }
    }

```

Listing 6.42: Konto_Skeleton

```

/**
2  * Beispiel "Omn Distributed Object Protocol"
  *
4  * @since      16-Dec-2002, 12-Jun-2007
  * @author     Hinrich E. G. Bonin
6  * @version    1.1
  */
8  package de.leuphana.ics.distributed;

10 import java.io.ObjectOutputStream;
  import java.io.ObjectInputStream;
12 import java.net.Socket;
  import java.net.ServerSocket;
14
15 public class Konto_Skeleton extends Thread
16 {
17     private KontoServer myServer;
18
19     public Konto_Skeleton(KontoServer myServer)
20     {
21         /*
22          * Erzeugt eine Referenz zum
23          * "Business Object" das vom Skeleton
24          * umhüllt wird
25          */
26         this.myServer = myServer;
27     }
28
29     public void run()
30     {
31         try
32         {
33             ServerSocket serverSocket =
34                 new ServerSocket(4711);
35             /*
36              * Wartet auf eine Socket-Verbindung
37              */
38             Socket socket = serverSocket.accept();
39
40             while (socket != null)
41             {
42                 ObjectInputStream inStream =
43                     new ObjectInputStream(
44                         socket.getInputStream());
45                 String method =
46                     (String) inStream.readObject();
47                 if (method.equals("ID"))

```

```

48         {
50             String id = myServer.getID();
51             ObjectOutputStream outputStream =
52                 new ObjectOutputStream (
53                     socket.getOutputStream());
54             outputStream.writeObject(id);
55             outputStream.flush();
56         } else if (method.equals("umsatz"))
57         {
58             int umsatz =
59                 myServer.getUmsatz();
60             ObjectOutputStream outputStream =
61                 new ObjectOutputStream (
62                     socket.getOutputStream());
63             outputStream.writeInt(umsatz);
64             outputStream.flush();
65         }
66     } catch (Throwable t)
67     {
68         t.printStackTrace();
69         System.exit(1);
70     }
71 }
72 }

```

Listing 6.43: KontoClient

```

/**
2  * Beispiel "Own Distributed Object Protocol"
3  *
4  * @since 16-Dec-2002, 12-Jun-2007
5  * @author Hinrich E. G. Bonin
6  * @version 1.1
7  */
8  package de.leuphana.ics.distributed;
9
10 public class KontoClient
11 {
12     public static void main(String [] args)
13     {
14         try
15         {
16             /*
17              * Erzeugt als Typ Konto eine
18              * Instanz vom Stellvertreter
19              */
20             Konto konto = new Konto_Stub();
21             System.out.println(

```

```

22         konto.getID () +
23         " = " +
24         konto.getUmsatz () +
25         " EUR");
26     } catch (Throwable t)
27     {
28         t.printStackTrace ();
29     }
30 }

```

Listing 6.44: KontoServerProg

```

/**
2  * Beispiel "Own Distributed Object Protocol"
3  *
4  * @since      16-Dec-2002, 12-Jun-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8  package de.leuphana.ics.distributed;
9
10 public class KontoServerProg
11 {
12     public static void main(String [] args)
13     {
14         Konto_Skeleton skel =
15             new Konto_Skeleton (
16                 new KontoServer ("Giro777", 1500));
17         skel.run ();
18     }
19 }

```

Protokolldatei KontoServer.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

```

```

D:\bonin\anwd\code>javac
de/leuphana/ics/distributed/KontoServerProg.java

```

```

D:\bonin\anwd\code>java

```



```

C:\WINDOWS\system32\cmd.exe

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/distributed/KontoClient.java

D:\bonin\anwd\code>java de.leuphana.ics.distributed.KontoClient
Giro777 = 1500 EUR

D:\bonin\anwd\code>

```

Legende:

Beispiel *Own Distributed Object Protocol*

Quellecode KontoClient ↔ S. 263

Abbildung 6.25: Applikation KontoClient

```

de.leuphana.ics.distributed.KontoServerProg
java.net.SocketException: Connection reset
...
D:\bonin\anwd\code>

```

6.10.2 Beispiel RMI

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

RPC

CORBA

²¹Präziser formuliert: RMI verbindet Systeme, die das *Standard Java Native Method Interface* (JNI) benutzen. Dies könnten prinzipiell auch Systeme in einer anderen Sprache sein.

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.

RMI ist ein Modell der verteilten Objekte, das allgemein bekannte Lösungen in eine durchgängige JavaTM 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-URL²³-Angabe. Diese hat folgende Form:

```
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.

```
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 6.45 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.

²²↔ [Vanderburg97] p. 525

²³*Uniform Resource Locator*

Tabelle 6.6 S. 273 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 6.3 S. 182).

Für die Entwicklung einer RMI-Anwendung sind folgende Schritte erforderlich:

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 implementiert und der Client aufrufen kann.

Client

2. Definieren einer Subklasse von `java.rmi.server.UnicastRemoteObject`
↔ Sie implementiert das `Remote`-Interface.

Server

3. Schreiben der Server-Applikation — Erzeugen einer Instanz des Remote-Objekts und „Exportieren“ dieser Instanz, das heißt, Verfügbar machen für die Nutzung durch den Client.
↔ Registrieren des Objektes anhand seines Namens mit einem Registrierungsservice. Üblicherweise erfolgt diese Registrierung

Server

mittels der Klasse `java.rmi.Naming` und dem Programm `rmiregistry` (↔ übernächsten Punkt).

Server

4. Erzeugen von Stub und Skeleton mit dem Programm `rmic` aus der compilierten Server-Klasse.

Server

5. Registrierung

Windows-NT-Plattform: `start rmiregistry [port]`

UNIX-Plattform: `rmiregistry [port] &`

Client

6. Schreiben der Client-Applikation
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`. 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!
...

```

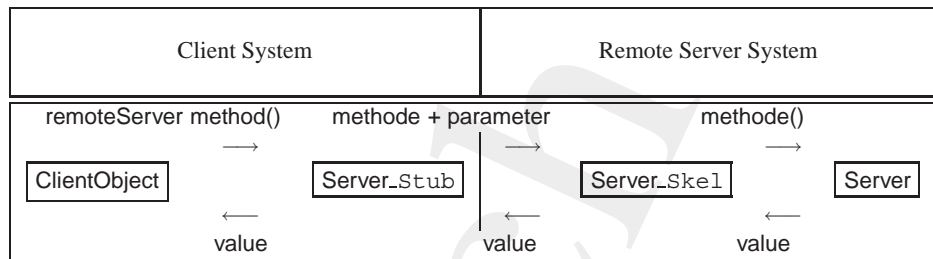
Listing 6.45: Bank

```

/**
2  * RMI-Client-Beispiel Idee von David Flanagan;
   * Java Examples in a Nutshell, 1997,
4  * p. 294 Quellcode stark modifiziert.
   *
6  * @since      13-Jun-1998, 26-Nov-2002, 14-Jun-2007
   * @author     Hinrich E. G. Bonin

```

²⁴Diese sind mit dem Modifikator `static` versehen, um als Toplevel-Interface bzw. Toplevel-Klassen nutzbar zu sein.

Legende:

Stub ≡ lokaler Stellvertreter

Skel ≡ Skeleton, RahmenDas RMI-System gliedert sich in die *Layer* — Näheres ⇔ [SunRMI98]:

1. *Stub/Skeleton Layer* — eine Proxy-Funktion auf der Client-Seite (Stub)
2. *Remote Reference Layer* — Verhalten bei einem einzelnen Objekt oder bei replizierten Objekten
3. *Transport Layer* — Verbindungsmanagement und Remote Objekt Verfolgung

Tabelle 6.6: RMI: *Stub/Skeleton, Remote-Reference* und *Transport*

```

8  * @version      1.1
   */
10 package de.leuphana.ics.bank;

12 import java.rmi.Naming;
   import java.rmi.RMISecurityManager;
14 import java.rmi.Remote;
   import java.rmi.RemoteException;

16
   import java.util.Vector;
18 /*
   * Bank enthaelt Interfaces und Klassen (top-level)
   */
20 */
   public class Bank
22 {
       /*
24      * Methoden auf dem Bankserver einzahlen abbheben
       * getStand getKontoBewegungen eroeffnenKonto
26      * aufolesenKonto
       */
28      public interface RemoteBank extends Remote
       {
30          public void einzahlen(String name,
                               String passwort,
32                               Euro geld)

```

```

        throws RemoteException , BankingException ;
34
    public Euro abheben( String name,
36                      String passwort,
                        int betrag)
38        throws RemoteException , BankingException ;

    public int getStand( String name,
40                      String passwort)
42        throws RemoteException , BankingException ;

    public Vector getKontoBewegungen(
44                      String name,
46                      String passwort)
48        throws RemoteException , BankingException ;

    public void eroeffnenKonto( String name,
50                              String passwort)
52        throws RemoteException , BankingException ;

    public Euro aufloesenKonto( String name,
54                              String passwort)
56        throws RemoteException , BankingException ;
    }
    /*
58     * Einfache Klasse die Geld repraesentiert
60     */
    public static class Euro implements
62                          java.io.Serializable
    {
        public int betrag ;
64
        public Euro( int betrag)
66        {
            this.betrag = betrag ;
68        }
    }

    /*
70     * Bankspezifische Ausnahmen
72     */
    public static class BankingException
74        extends Exception
    {
76        public BankingException( String nachricht)
78        {
            super( nachricht);
80        }
    }

```

```
82
83  /*
84  * Bank$Client kommuniziert mit dem RMI-Server
85  */
86  public static class Client
87  {
88      public static void main(String args[])
89      {
90          try
91          {
92              /*
93               * Sicherheit gegen
94               * untrusted stub code
95               * ueber das Netz
96               */
97              //System.setSecurityManager (
98              //    new RMISecurityManager ());
99              /*
100             * Default-Wert BoninRemote
101             */
102             String url = System.getProperty (
103             "bankname", "rmi:///BoninRemote");
104             /*
105             * Naming Objekt
106             * kontaktet rmiregistry
107             */
108             RemoteBank bank =
109             (RemoteBank)
110             Naming.lookup(url);
111
112             String aktion = args[0].toLowerCase();
113
114             if (aktion.equals("einzahlen"))
115             {
116                 Euro geld =
117                 new Euro(
118                 Integer.parseInt(args[3]));
119                 bank.einzahlen(
120                 args[1], args[2], geld);
121                 System.out.println (
122                 "Eingezahlt:_" +
123                 geld.betrag +
124                 "_Euro");
125             } else if (aktion.equals("abheben"))
126             {
127                 Euro geld =
128                 bank.abheben(
129                 args[1], args[2],
130                 Integer.parseInt(args[3]));
```

```

132         System.out.println (
133             "Abgehoben: " +
134             geld.betrag +
135             " Euro");
136     } else if
137     (aktion.equals("stand"))
138     {
139         System.out.println (
140             "Kontostand: " +
141             bank.getStand (
142                 args[1], args[2]) +
143             " Euro");
144     } else if
145     (aktion.equals("bewegungen"))
146     {
147         Vector bewegungen =
148             bank.getKontoBewegungen(
149                 args[1], args[2]);
150         for (int i = 0;
151             i < bewegungen.size();
152             i++)
153         {
154             System.out.println (
155                 bewegungen.elementAt
156                 (i));
157         }
158     } else if
159     (aktion.equals("eroeffnen"))
160     {
161         bank.eroeffnenKonto (
162             args[1], args[2]);
163         System.out.println (
164             "Konto_eroeffnet!");
165     } else if
166     (aktion.equals("aufloesen"))
167     {
168         Euro geld =
169             bank.aufloesenKonto
170             (args[1], args[2]);
171         System.out.println (
172             geld.betrag +
173             " Euro erhalten_Sie_ +
174             " noch ausgezahlt!");
175     } else
176     {
177         System.out.println (
178             "Unbekannte_Aktion!");
179     }
180 } catch (RemoteException e)

```

```

180         {
181             System.err.println(e);
182         } catch (BankingException e)
183         {
184             System.err.println(e.getMessage());
185         } catch (Exception e)
186         {
187             System.err.println(e);
188             System.err.println(
189                 "Usage: java [-Dbank=<url>] Bank$Client " +
190                 "<aktion> <name> <passwort> [<betrag>]");
191             System.err.println(
192                 "wobei <aktion> einer der folgenden Wert ist: " +
193                 "\nEinzahlen, Abheben, Stand, " +
194                 "\nBewegungen, Eroeffnen, Aufloesen");
195         }
196     }
197 }
198 }

```

Listing 6.46: RemoteBankServer

```

/**
2  * RMI-Client-Beispiel Idee von David Flanagan;
3  * Java Examples in a Nutshell, 1997,
4  * p. 294 Quellcode stark modifiziert.
5  *
6  * @since      13-Jun-1998, 26-Nov-2002, 14-Jun-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    1.1
9  */
10 package de.leuphana.ics.bank;

12 import java.rmi.Naming;
13 import java.rmi.RemoteException;
14 import java.rmi.server.UnicastRemoteObject;

16 import java.util.Date;
17 import java.util.Hashtable;
18 import java.util.Vector;

20 public class RemoteBankServer
21     extends UnicastRemoteObject
22     implements RemoteBank
23 {
24     class Konto
25     {
26         int stand;
27         String passwort;

```

```

28     Vector bewegungen = new Vector ();
30     Konto(String passwort)
31     {
32         this.passwort = passwort;
33         bewegungen.addElement (
34             "Kontoeroeffnung am: " +
35             new Date ());
36     }
37 }
38
39 Hashtable kontos = new Hashtable ();
40
41 public RemoteBankServer()
42     throws RemoteException
43 {
44     super ();
45 }
46
47 public void einzahlen (String name,
48                       String passwort,
49                       Euro geld)
50     throws RemoteException, BankingException
51 {
52     Konto myK = pruefen (name, passwort);
53     synchronized (myK)
54     {
55         myK.stand += geld.betrag;
56         myK.bewegungen.addElement (
57             "Eingezahlt: " +
58             geld.betrag +
59             " am " +
60             new Date ());
61     }
62 }
63
64 public Euro abheben (String name,
65                    String passwort,
66                    int betrag)
67     throws RemoteException, BankingException
68 {
69     Konto myK = pruefen (name, passwort);
70     synchronized (myK)
71     {
72         if (myK.stand < betrag)
73         {
74             throw new BankingException (
75                 "Keine Deckung!");
76         }

```

```
78         myK.stand -= betrag;
79         myK.bewegungen.addElement(
80             "Abgehoben:_" +
81             betrag +
82             "_am_" +
83             new Date());
84         return new Euro(betrag);
85     }
86 }
87
88 public int getStand(String name,
89                     String password)
90     throws RemoteException, BankingException
91 {
92     Konto myK = pruefen(name, password);
93     synchronized (myK)
94     {
95         return myK.stand;
96     }
97 }
98
99 public Vector getKontoBewegungen(
100     String name,
101     String password)
102     throws RemoteException, BankingException
103 {
104     Konto myK = pruefen(name, password);
105     synchronized (myK)
106     {
107         return myK.bewegungen;
108     }
109 }
110
111 public synchronized void eroeffnenKonto(
112     String name,
113     String password)
114     throws RemoteException, BankingException
115 {
116     if (kontos.get(name) != null)
117     {
118         throw new BankingException(
119             "Konto_gibt_es_schon!");
120     }
121     Konto myK = new Konto(password);
122     kontos.put(name, myK);
123 }
124
125 public Konto pruefen(String name,
126                     String password)
```

```

126     throws BankingException
127     {
128         synchronized (kontos)
129         {
130             Konto myK = (Konto) kontos.get(name);
131             if (myK == null)
132             {
133                 throw new BankingException(
134                     "Kein solches Konto!");
135             }
136             if (!passwort.equals(myK.passwort))
137             {
138                 throw new BankingException(
139                     "Falches Passwort!");
140             }
141             return myK;
142         }
143     }
144
145     public synchronized Euro auflösenKonto(
146         String name,
147         String passwort)
148         throws RemoteException, BankingException
149     {
150         Konto myK;
151         myK = prüfen(name, passwort);
152         kontos.remove(name);
153         synchronized (myK)
154         {
155             int wert = myK.stand;
156             myK.stand = 0;
157             return new Euro(wert);
158         }
159     }
160
161     public static void main(String[] args)
162     {
163         try
164         {
165             RemoteBankServer bank
166                 = new RemoteBankServer();
167             String name
168                 = System.getProperty(
169                     "bankname", "BoninRemote");
170             Naming.rebind(name, bank);
171             System.out.println(
172                 name +
173                 "\tist eröffnet und" +
174                 "\tbereit fuer Buchungen.");

```



```

    } catch (Exception e)
176     {
        System.err.println(e);
178     System.err.println(
        "Usage: java [-Dbankname=<name>]" +
180     " RemoteBankServer");
        System.exit(1);
182     }
    }
184 }

```

Protokoll einer Session

```

--- Windows-NT-Rechner 193.174.33.66
--- Persönliche Firewall auststellen --

```

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

```

```

D:\bonin\anwd\code>javac
de/leuphana/ics/bank/RemoteBankServer.java
Note: de/leuphana/ics/bank/RemoteBankServer.java
uses unchecked or unsafe operations.

```

```

D:\bonin\anwd\code>rmic
de.leuphana.ics.bank.RemoteBankServer

```

```

D:\bonin\anwd\code>start rmiregistry
start rmiregistry

```

```

D:\bonin\anwd\code>java
de.leuphana.ics.bank.RemoteBankServer
BoninRemote ist eroeffnet und bereit fuer Buchungen.

```

```

--- UNIX-Rechner 193.174.33.106
>java -fullversion
java full version "JDK 1.1.6 IBM build a116-19980529" (JIT: jitc)
>javac de/leuphana/ics/bank/Bank.java
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\${Client doof otto geheim
Unbekannte Aktion!

```

```
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client eroeffnen otto geheim
Konto eroeffnet!
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client einzahlen otto geheim 100
Eingezahlt: 100 Euro
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client einzahlen otto geheim 200
Eingezahlt: 200 Euro
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client stand otto geheim
Kontostand : 300 Euro
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client abheben otto geheim 50
Abgehoben: 50 Euro
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client stand otto geheim
Kontostand : 250 Euro
>java -Dbank="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client bewegungen otto geheim
Kontoeroeffnung am: Sat Jun 13 14:14:13 CEST 2007
Eingezahlt: 100 am Sat Jun 13 14:15:03 CEST 2007
Eingezahlt: 200 am Sat Jun 13 14:15:14 CEST 2007
Abgehoben: 50 am Sat Jun 13 14:16:00 CEST 2007
>java -Dbankname="rmi://193.174.33.100:1099/BoninRemote" \
de.leuphana.ics.bank.Bank\$Client aufloesen otto geheim
250 Euro erhalten Sie noch ausgezahlt!
>
```

6.11 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` (\leftrightarrow S. 288) nutzt JDOM, eine *Open Source Library* 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 (\leftrightarrow Abschnitt 6.10.2 S. 265) zu CORBA (*Common Object Request Broker Architecture*).

Für die beispielhaften Lieferantendaten `lief-dat.xml` (\leftrightarrow S. 286) wird eine *Document Type Definition* entworfen. Diese DTD ist in der Datei `lief-dat.dtd` notiert (\leftrightarrow S. 285). Für die die Ergebnisdatei `jahr-tab.xml` wird ebenfalls eine *Document Type Definition* entworfen. Diese DTD ist in der Datei `jahr-tab.dtd` notiert (\leftrightarrow S. 288).

Aus JDOM nutzen wir die folgende Klasse:

²⁵Hinweis: Nach Durchführung des Batchlaufes `jdom-b10/build.bat` ist die Java-Umgebungsvariable `CLASSPATH` zu ergänzen, hier um: `c:/programme/jdom-b10/build/jdom.jar`.

```

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 (\leftrightarrow S.295). Das Einlesen der `lief-dat.xml` basiert auf folgender Konstruktion:

Listing 6.47: Detail XML-Input

```

SAXBuilder builder = new SAXBuilder();
2 builder.setValidation(true);

4 Document docInput = builder.build(new File(
    this.getInputXMLFile()));
6
Element rootInput =
8 docInput.getRootElement();

10 lieferantenListe =
    rootInput.getChildren(
12 this.getInputXMLFileChild());

```

Zur Ausgabe der XML-Datei `jahr-tab.xml` starten wir mit dem `dtdRootElement`, hier `umsatz`, und geben die dazugehörige DTD als `dtdOutputXMLFile` an, hier `jahr-tab.dtd`. Die Details zeigt der folgende Quellcodeausschnitt:

Listing 6.48: Detail XML-Output

```

Document docOutput = new Document(
2 new Element(
    this.getDtdRootElement()));
4 DocType docType = new DocType(
    this.getDtdRootElement(),
6 this.getDtdOutputXMLFile());

8 docOutput.setDocType(docType);

```

```

10 Element rootOutput = docOutput.getRootElement();
    addElementeMonate(rootOutput);
12
13 XMLOutputter out = new XMLOutputter();
14
15 Format format = Format.getPrettyFormat();
16 out.setFormat(format);
17
18 BufferedWriter bw = new BufferedWriter(
    new FileWriter(outputXMLFile));
19
20 out.output(docOutput, bw);
21
22 bw.close();

```

Listing 6.49: lief-dat.dtd

```

<?xml version="1.0" encoding="UTF-8"?>
2 <!-- DTD zur Inputvalidation von lief-dat.xml -->
   <!-- Bonin May 2004 -->
4 <!ELEMENT lief-dat (lieferant+) >
   <!ELEMENT lieferant (
6     januar ,
     februar ,
8     maerz ,
     april ,
10    mai ,
     juni ,
12    juli ,
     august ,
14    september ,
     oktober ,
16    november ,
     dezember
18    )>
   <!ATTLIST lieferant id ID #REQUIRED
20                        m ( A | B | C ) #REQUIRED>
   <!ELEMENT januar (#PCDATA)>
22 <!ELEMENT februar (#PCDATA)>
   <!ELEMENT maerz (#PCDATA)>
24 <!ELEMENT april (#PCDATA)>
   <!ELEMENT mai (#PCDATA)>
26 <!ELEMENT juni (#PCDATA)>
   <!ELEMENT juli (#PCDATA)>
28 <!ELEMENT august (#PCDATA)>
   <!ELEMENT september (#PCDATA)>
30 <!ELEMENT oktober (#PCDATA)>
   <!ELEMENT november (#PCDATA)>

```

32 <!ELEMENT dezember (#PCDATA)>

Listing 6.50: lief-dat.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE lief-dat SYSTEM "lief-dat.dtd">
<lief-dat>
4   <lieferant id="OttoAG" m="A">
      <januar>10</januar>
6     <februar>100</februar>
      <maerz>10</maerz>
8     <april>10</april>
      <mai>100</mai>
10    <juni>100</juni>
      <juli>100</juli>
12    <august>100</august>
      <september>100</september>
14    <oktober>200</oktober>
      <november>60</november>
16    <dezember>100</dezember>
    </lieferant>
18   <lieferant id="MuellerGmbH" m="C">
      <januar>100</januar>
20    <februar>100</februar>
      <maerz>30</maerz>
22    <april>100</april>
      <mai>10</mai>
24    <juni>200</juni>
      <juli>100</juli>
26    <august>45</august>
      <september>100</september>
28    <oktober>67</oktober>
      <november>10</november>
30    <dezember>500</dezember>
    </lieferant>
32   <lieferant id="KrauseOHG" m="A">
      <januar>100</januar>
34    <februar>100</februar>
      <maerz>500</maerz>
36    <april>10</april>
      <mai>100</mai>
38    <juni>10</juni>
      <juli>100</juli>
40    <august>400</august>
      <september>100</september>
42    <oktober>100</oktober>
      <november>100</november>
44    <dezember>100</dezember>
    </lieferant>

```

```
46     <lieferant id="SchulzeAG" m="C">
47         <januar>100</januar>
48         <februar>100</februar>
49         <maerz>100</maerz>
50         <april>10</april>
51         <mai>100</mai>
52         <juni>100</juni>
53         <juli>unbekannt</juli>
54         <august>100</august>
55         <september>100</september>
56         <oktober>60</oktober>
57         <november>100</november>
58         <dezember>800</dezember>
59     </lieferant>
60     <lieferant id="HausmannKG" m="B">
61         <januar>100</januar>
62         <februar>100</februar>
63         <maerz>100</maerz>
64         <april>100</april>
65         <mai>100</mai>
66         <juni>100</juni>
67         <juli>100</juli>
68         <august>100</august>
69         <september>100</september>
70         <oktober>100</oktober>
71         <november>100</november>
72         <dezember>100</dezember>
73     </lieferant>
74     <lieferant id="MeyerGmbH" m="C">
75         <januar>100</januar>
76         <februar>100</februar>
77         <maerz>100</maerz>
78         <april>100</april>
79         <mai>100</mai>
80         <juni>100</juni>
81         <juli>100</juli>
82         <august>100</august>
83         <september>100</september>
84         <oktober>100</oktober>
85         <november>100</november>
86         <dezember>100</dezember>
87     </lieferant>
88     <lieferant id="WilhelmEG" m="B">
89         <januar>100</januar>
90         <februar>100</februar>
91         <maerz>100</maerz>
92         <april>100</april>
93         <mai>100</mai>
94         <juni>100</juni>
```

```

96         <juli>100</juli>
          <august>100</august>
98         <september>100</september>
          <oktober>100</oktober>
          <november>100</november>
100        <dezember>100</dezember>
        </lieferant>
102    <lieferant id="GutknechtGmbH" m="C">
          <januar>100</januar>
104        <februar>100</februar>
          <maerz>100</maerz>
106        <april>30</april>
          <mai>100</mai>
108        <juni>100</juni>
          <juli>100</juli>
110        <august>100</august>
          <september>100</september>
112        <oktober>100</oktober>
          <november>100</november>
114        <dezember>100</dezember>
        </lieferant>
116 </lief-dat>

```

Listing 6.51: jahr-tab.dtd

```

<?xml version="1.0" encoding="UTF-8"?>
2 <!-- DTD zur XML-Ausgabe von jahr-tab -->
   <!-- Bonin May 2004 -->
4 <!ELEMENT umsatz (
   jan , feb , mar , apr ,
6   may , jun , jul , aug ,
   sep , oct , nov , dec)>
8 <!ELEMENT jan (#PCDATA)>
  <!ELEMENT feb (#PCDATA)>
10 <!ELEMENT mar (#PCDATA)>
  <!ELEMENT apr (#PCDATA)>
12 <!ELEMENT may (#PCDATA)>
  <!ELEMENT jun (#PCDATA)>
14 <!ELEMENT jul (#PCDATA)>
  <!ELEMENT aug (#PCDATA)>
16 <!ELEMENT sep (#PCDATA)>
  <!ELEMENT oct (#PCDATA)>
18 <!ELEMENT nov (#PCDATA)>
  <!ELEMENT dec (#PCDATA)>

```

Listing 6.52: Aggregation

```

/**
2 * Example "Lieferantendaten"

```



```
* selektieren und aggregieren
4 *
* @since      29-May-2007
6 * @author    Hinrich E. G. Bonin
* @version    1.2
8 */
package de.leuphana.ics.xmldata;

10
import java.io.BufferedWriter;
12 import java.io.File;
import java.io.FileWriter;
14 import java.util.ArrayList;
import java.util.List;

16
import org.jdom.DocType;
18 import org.jdom.Document;
import org.jdom.Element;
20 import org.jdom.input.JDOMParseException;
import org.jdom.input.SAXBuilder;
22 import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;

24
public class Aggregation
26 {
    int januar = 0;
28     int februar = 0;
    int maerz = 0;
30     int april = 0;
    int mai = 0;
32     int juni = 0;
    int juli = 0;
34     int august = 0;
    int september = 0;
36     int oktober = 0;
    int november = 0;
38     int dezember = 0;

40     String inputXMLFile = new String();
    String inputXMLFileChild = new String();
42
    String outputXMLFile = new String();
44
    String dtdRootElement = new String();
46     String dtdOutputXMLFile = new String();

48     String getInputXMLFile()
    {
50         return inputXMLFile;
    }
}
```

```

52     String getInputXMLFileChild ()
54     {
56         return inputXMLFileChild ;
58     }
59
60     String getOutputXMLFile ()
61     {
62         return outputXMLFile ;
63     }
64
65     String getDtdRootElement ()
66     {
67         return dtdRootElement ;
68     }
69
70     String getDtdOutputXMLFile ()
71     {
72         return dtdOutputXMLFile ;
73     }
74
75     Aggregation (
76         String inputXMLFile ,
77         String inputXMLFileChild ,
78         String outputXMLFile ,
79         String dtdRootElement ,
80         String dtdOutputXMLFile )
81     {
82         this.inputXMLFile = inputXMLFile ;
83         this.inputXMLFileChild = inputXMLFileChild ;
84         this.outputXMLFile = outputXMLFile ;
85         this.dtdRootElement = dtdRootElement ;
86         this.dtdOutputXMLFile = dtdOutputXMLFile ;
87     }
88
89     /**
90     * [R01]
91     *
92     * @return Liste der Lieferanten
93     */
94     List readXMLInput ()
95     {
96         /**
97         * zur Vermeidung von
98         * java.lang.NullPointerException
99         * nicht mit null initialisiert
100        */
101         List lieferantenListe = new ArrayList ();

```

```
102     try
103     {
104         SAXBuilder builder =
105             new SAXBuilder();
106         builder.setValidation(true);
107
108         Document docInput = builder.build(
109             new File(this.getInputXMLFile()));
110
111         Element rootInput =
112             docInput.getRootElement();
113
114         lieferantenListe =
115             rootInput.getChildren(
116                 this.getInputXMLFileChild());
117     }
118     catch (JDOMParseException e)
119     {
120         System.err.println(e);
121         System.exit(1);
122     }
123     catch (Exception e)
124     {
125         System.err.println(e);
126         System.exit(1);
127     }
128     return lieferantenListe;
129 }
130
131 /**
132  * @param lieferantenListe
133  * @return Description of the Return
134  *        Value
135  */
136 Aggregation process(List lieferantenListe)
137 {
138     /*
139     * [R03]
140     */
141     String m;
142     String id;
143
144     try
145     {
146         for (int i = 0;
147              i < lieferantenListe.size();
148              i++)
149         {
```

```

150         Element lieferant =
151             (Element) (lieferantenListe.get(i));
152         id = lieferant.getAttributeValue("id");
153         m = lieferant.getAttributeValue("m");
154         /*
155          * [R03.2]
156          */
157         if (m.equals("B"))
158             {
159                 System.out.println (
160                     "Der Datensatz des Lieferanten " +
161                     id +
162                     " ist dubious!");
163             }
164         /*
165          * [R03.3]
166          */
167         else if (m.equals("C"))
168             {
169             januar += numerisch(
170                 lieferant.getChildText("januar"));
171             februar += numerisch(
172                 lieferant.getChildText("februar"));
173             maerz += numerisch(
174                 lieferant.getChildText("maerz"));
175             april += numerisch(
176                 lieferant.getChildText("april"));
177             mai += numerisch(
178                 lieferant.getChildText("mai"));
179             juni += numerisch(
180                 lieferant.getChildText("juni"));
181             juli += numerisch(
182                 lieferant.getChildText("juli"));
183             august += numerisch(
184                 lieferant.getChildText("august"));
185             september += numerisch(
186                 lieferant.getChildText("september"));
187             oktober += numerisch(
188                 lieferant.getChildText("oktober"));
189             november += numerisch(
190                 lieferant.getChildText("november"));
191             dezember += numerisch(
192                 lieferant.getChildText("dezember"));
193             } else
194             {
195                 /*
196                  * [R03.1]
197                  * Satz wird uebersprungen
198                 */

```

```
200         }
201     }
202     } catch (Exception e)
203     {
204         e.printStackTrace();
205     }
206     return this;
207 }
208
209 /**
210  * Schreibt die XML-datei im Pretty-Print-Format.
211  */
212 void writeXMLOutput()
213 {
214     try
215     {
216         Document docOutput = new Document(
217             new Element(
218                 this.getDtdRootElement());
219         DocType docType = new DocType(
220             this.getDtdRootElement(),
221             this.getDtdOutputXMLFile());
222
223         docOutput.setDocType(docType);
224
225         Element rootOutput =
226             docOutput.getRootElement();
227         addElementeMonate(rootOutput);
228
229         XMLOutputter out = new XMLOutputter();
230
231         Format format = Format.getPrettyFormat();
232         out.setFormat(format);
233
234         BufferedWriter bw = new BufferedWriter(
235             new FileWriter(outputXMLFile));
236
237         out.output(docOutput, bw);
238
239         bw.close();
240     } catch (Exception e)
241     {
242         {
243             e.printStackTrace();
244         }
245     }
246 }
```

```

248  /**
249   * @param rootOutput Addiert die Monatelement zum
250   *   Root-Element
251   */
252  private void addElementeMonate(Element rootOutput)
253  {
254      rootOutput.addContent(new Element("jan").
255          setText(String.valueOf(januar)));
256      rootOutput.addContent(new Element("feb").
257          setText(String.valueOf(februar)));
258      rootOutput.addContent(new Element("mar").
259          setText(String.valueOf(maerz)));
260      rootOutput.addContent(new Element("apr").
261          setText(String.valueOf(april)));
262      rootOutput.addContent(new Element("may").
263          setText(String.valueOf(mai)));
264      rootOutput.addContent(new Element("jun").
265          setText(String.valueOf(juni)));
266      rootOutput.addContent(new Element("jul").
267          setText(String.valueOf(juli)));
268      rootOutput.addContent(new Element("aug").
269          setText(String.valueOf(august)));
270      rootOutput.addContent(new Element("sep").
271          setText(String.valueOf(september)));
272      rootOutput.addContent(new Element("oct").
273          setText(String.valueOf(oktober)));
274      rootOutput.addContent(new Element("nov").
275          setText(String.valueOf(november)));
276      rootOutput.addContent(new Element("dec").
277          setText(String.valueOf(dezember)));
278  }
279  /**
280   * R[03.3.2]
281   *
282   * @param s String der eine int-Zahl repraesentiert
283   * @return int-Zahl; im Fehlerfall int-Null.
284   */
285  private int numerisch(String s)
286  {
287      int x = 0;
288      try
289      {
290          x = Integer.parseInt(s);
291      } catch (NumberFormatException e)
292      {
293          System.err.println(e);
294          x = 0;
295      }
296  }

```

```

    return x;
298 }
}

```

Listing 6.53: AggregationProg

```

/**
2  * Example "Lieferantendaten"
  * selektieren und aggregieren
4  *
  * @since 29-May-2007
6  * @author Hinrich E. G. Bonin
  * @version 1.2
8  */
package de.leuphana.ics.xmldata;
10
public class AggregationProg
12 {
    public static void main(String [] args)
14 {
        final String inputXMLFile =
16         "de/leuphana/ics/xmldata/lief-dat.xml";
        final String inputXMLFileChild =
18         "lieferant";

20         final String outputXMLFile =
           "de/leuphana/ics/xmldata/jahr-tab.xml";
22
        final String dtdRootElement = "umsatz";
24         final String dtdOutputXMLFile =
           "jahr-tab.dtd";
26

        Aggregation foo = new Aggregation (
28             inputXMLFile ,
                inputXMLFileChild ,
30             outputXMLFile ,
                dtdRootElement ,
32             dtdOutputXMLFile );

34         foo.process (foo.readXMLInput ()).writeXMLOutput ();
        /*
36         * [R02]
        */
38         System.out.println ("Alles verarbeitet!");
40     }
}

```

Protokolldatei Aggregation.log

296 KAPITEL 6. KONSTRUKTIONEN (ANALYSE UND SYNTHESE)

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac
  de/leuphana/ics/xmldata/AggregationProg.java
```

```
D:\bonin\anwd\code>java
  de.leuphana.ics.xmldata.AggregationProg
java.lang.NumberFormatException: For input string: "unbekannt"
Der Datensatz des Lieferanten HausmannKG ist dubios!
Der Datensatz des Lieferanten WilhelmEG ist dubios!
Alles verarbeitet!
```

```
D:\bonin\anwd\code>
  REM Update 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.leuphana.ics.xmldata.AggregationProg
org.jdom.input.JDOMParseException:
  Error on line 5 of document
  file:/D:/bonin/anwd/code/de/leuphana/ics/xmldata/lief-dat.xml:
  Element type "jaenner" must be declared.
```

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

Listing 6.54: jahr-tab.xml

```
<?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE umsatz SYSTEM "jahr-tab.dtd">
4 <umsatz>
  <jan>400</jan>
6   <feb>400</feb>
  <mar>330</mar>
8   <apr>240</apr>
  <may>310</may>
10  <jun>500</jun>
  <jul>300</jul>
12  <aug>345</aug>
```



```
14     <sep>400 </sep>
16     <oct>327 </oct>
    <nov>310 </nov>
    <dec>1500 </dec>
</umsatz>
```

Einige Hinweise zum Arbeiten mit JDOM

JDOM erleichtert das Navigieren im Baum der Elemente. Wie in der **JDOM** 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-Element 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:

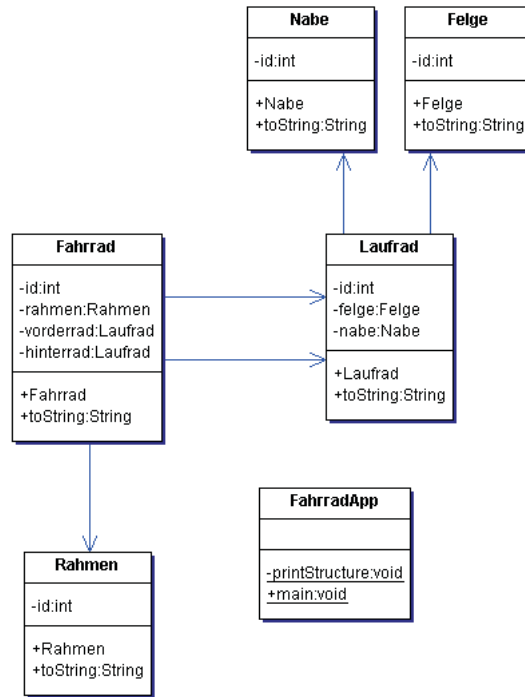
```
List mylist = root.getChildren("myelement");
Element myelement = (Element) (mylist.get(0));
Attribute idAttribute = myelement.getAttribute("id");
int id = idAttribute.getIntValue();
int m = myelement.getAttributeValue("m");
```

Attribute werden wie folgt gesetzt, modifiziert oder entfernt:

```
myelement.setAttribute("m", "0");
myelement.setAttribute("neu", "OK");
myelement.removeAttribute("m");
```

6.12 Komposition mittels Interface-Konstruktion

Als Beispiel für eine Komposition verwenden wir hier ein Fahrrad, das sich aus einem Rahmen und zwei Laufrädern (Vorder- und Hinterrad) zusammensetzt. Ein Laufrad besteht in diesem Beispiel exemplarisch



Legende:

Notation in *Unified Modeling Language* (UML) Class Diagram.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung 6.26: Associations für eine Zusammensetzung

vereinfacht aus einer Felge und einer Nabe. Zunächst ist dieses Fahrrad mittels *Associations* abgebildet. Ein entsprechendes Klassendiagramm in UML-Notation (*Unified Modeling Language*) zeigt \leftrightarrow Abbildung 6.26 S. 299. Die dazugehörigen Java-Klassen sind:

- `FahrradApp.java` — die Java-Applikation (\leftrightarrow S. 300)
- `Fahrrad.java` (\leftrightarrow S. 301)
- `Rahmen.java` (\leftrightarrow S. 302)
- `LaufRad.java` (\leftrightarrow S. 302)
- `Felge.java` (\leftrightarrow S. 303)
- `Nabe.java` (\leftrightarrow S. 303)

Da es sich im Kern um das klassische Stücklistenmuster handelt, kann die Lösung auch allgemein gültig konstruiert werden. Dazu nutzt man einen Baum der zwei Typen von Knoten (\equiv *Component*) aufweist: Zusammengesetzte Knoten (\equiv *Composite*) und Endknoten, sogenannte Baumblätter (\equiv *Leaf*). Das entsprechende Klassendiagramm in UML-Notation zeigt \leftrightarrow Abbildung 6.27 S. 305. Zu den obigen Klassen — natürlich in entsprechend angepasster Form — kommen hinzu, das Interface `Component.java` (\leftrightarrow S. 311) und seine beiden Implementations `Composite.java` (\leftrightarrow S. 309) und `Leaf.java` (\leftrightarrow S. 310). Das Interface `Component.java` (\leftrightarrow S. 311) legt die Methoden fest, die auf jeden Knoten des Baumes anwendbar sind. Hinweis: Bei einem *Leaf*-Knoten entfällt die Methode `add(component)`. Dies wird hier durch einen „leeren Methodenkörper“ angedeutet.

Listing 6.55: `FahrradApp`

```

/**
2  * Pattern "Association"
3  *
4  * @author      Bonin
5  * @version     1.0
6  */
package de.unilueneburg.as.associate;

8
public class FahrradApp
10 {
    private static void printStructure (
12     Fahrrad fahrrad)
    {
14     System.out.println (
        fahrrad.toString());
    }
}

```

```

16     }
18     public static void main(String[] args)
19     {
20         Rahmen rahmen = new Rahmen(1);
21         Felge felge1 = new Felge(2);
22         Felge felge2 = new Felge(3);
23         Nabe nabe1 = new Nabe(4);
24         Nabe nabe2 = new Nabe(5);
25
26         Laufrad vorderrad = new Laufrad(6,
27             felge1, nabe1);
28         Laufrad hinterrad = new Laufrad(7,
29             felge2, nabe2);
30
31         Fahrrad fahrrad = new Fahrrad(1,
32             rahmen, vorderrad, hinterrad);
33
34         printStructure(fahrrad);
35     }
36 }

```

Listing 6.56: Fahrrad

```

/**
2  * Pattern "Association"
3  *
4  * @author    Bonin
5  * @version  1.0
6  */
package de.unilueneburg.as.associate;
7
8
9
10 public class Fahrrad
11 {
12     private int id = 0;
13     private Rahmen rahmen;
14     private Laufrad vorderrad;
15     private Laufrad hinterrad;
16
17     public Fahrrad(int id,
18                 Rahmen rahmen,
19                 Laufrad vorderrad,
20                 Laufrad hinterrad)
21     {
22         this.id = id;
23         this.rahmen = rahmen;
24         this.vorderrad = vorderrad;
25         this.hinterrad = hinterrad;
26     }
27 }

```

```

26     public String toString()
27     {
28         return "Fahrrad_" + id + "\n_" +
29             "_mit_" + rahmen.toString() +
30             "_mit_" + vorderrad.toString() +
31             "_mit_" + hinterrad.toString() +
32             "\n";
33     }
34 }

```

Listing 6.57: Rahmen

```

/**
2  * Pattern "Association"
3  *
4  * @author    Bonin
5  * @version   1.0
6  */
package de.unilueneburg.as.associate;

8
public class Rahmen
10 {
11     private int id = 0;
12
13     public Rahmen(int id)
14     {
15         this.id = id;
16     }
17
18     public String toString()
19     {
20         return "Rahmen_" + id;
21     }
22 }

```

Listing 6.58: Laufrad

```

/**
2  * Pattern "Association"
3  *
4  * @author    Bonin
5  * @version   1.0
6  */
package de.unilueneburg.as.associate;

8
public class Laufrad
10 {
11     private int id = 0;

```

```

12     private Felge felge;
13     private Nabe nabe;
14
15     public Laufrad(int id,
16                   Felge felge,
17                   Nabe nabe)
18     {
19         this.id = id;
20         this.felge = felge;
21         this.nabe = nabe;
22     }
23
24     public String toString()
25     {
26         return "Laufrad_" + id + "_\n_" +
27             "_mit_" + felge.toString() +
28             "_mit_" + nabe.toString() +
29             "\n";
30     }
31 }

```

Listing 6.59: Felge

```

/**
2  * Pattern "Association"
3  *
4  * @author    Bonin
5  * @version   1.0
6  */
7
8  package de.unilueneburg.as.associate;
9
10 public class Felge
11 {
12     private int id = 0;
13
14     public Felge(int id)
15     {
16         this.id = id;
17     }
18
19     public String toString()
20     {
21         return "Felge_" + id;
22     }
23 }

```

Listing 6.60: Nabe

```
/**
```

```

2  * Pattern "Association"
3  *
4  * @author    Bonin
5  * @version   1.0
6  */
package de.unilueneburg.as.associate;

8
public class Nabe
10 {
11     private int id = 0;
12
13     public Nabe(int id)
14     {
15         this.id = id;
16     }
17
18     public String toString()
19     {
20         return "Nabe_" + id;
21     }
22 }

```

Protokolldatei FahrradApp0.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode)

D:\bonin\anwd\code>javac
  de/unilueneburg/as/associate/FahrradApp.java

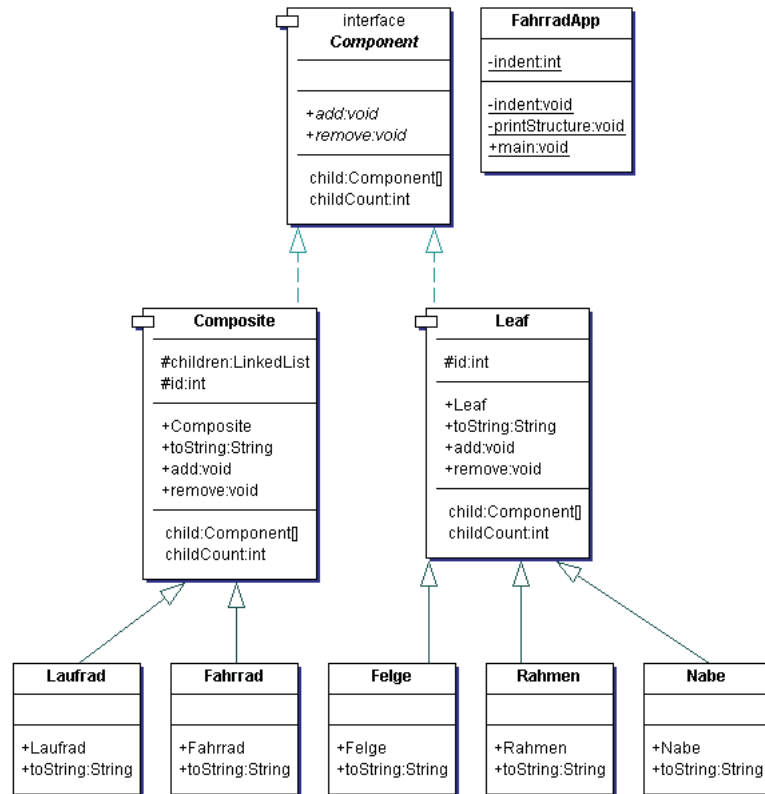
D:\bonin\anwd\code>java
  de.unilueneburg.as.associate.FahrradApp
Fahrrad 1
  mit Rahmen 1 mit Laufrad 6
  mit Felge 2 mit Nabe 4
mit Laufrad 7
  mit Felge 3 mit Nabe 5

D:\bonin\anwd\code>

```

Listing 6.61: FahrradApp

/**



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung 6.27: Komposition mittels Interface-Konstruktion

```

2  * Pattern "Composite"
3  *
4  * @author    Bonin
5  * @version   1.1
6  */
package de.unilueneburg.as.composite;

8
public class FahrradApp
10 {
    private static int indent = 0;

12

14 private static void indent()
    {
16     for (int i = 0; i < indent; i++)
18     {
        System.out.print("└");
19     }
20 }

22
private static void printStructure (
24     Component component)
    {
26     indent();
    System.out.println("Level:└" + component);
28     indent += 4;
    for (int i = 0; i <
30         component.getChildCount(); i++)
        {
32         printStructure (component.getChild(i));
        }
34     indent -= 4;
    }

36
public static void main(String [] args)
38 {
    Fahrrad fahrrad = new Fahrrad(1);
40    Laufrad vorderrad = new Laufrad(2);
    Laufrad hinterrad = new Laufrad(3);

42

    Rahmen rahmen = new Rahmen(1);
44    Felge felge1 = new Felge(2);
    Felge felge2 = new Felge(3);
46    Nabe nabe1 = new Nabe(4);
    Nabe nabe2 = new Nabe(5);

48

    fahrrad.add(rahmen);
50    fahrrad.add(vorderrad);

```

```

        fahrrad.add(hinterrad);
52
        vorderrad.add(felge1);
54        hinterrad.add(felge2);

        vorderrad.add(nabe1);
56        hinterrad.add(nabe2);
58
        printStructure(fahrrad);
60    }
}

```

Listing 6.62: Fahrrad

```

/**
2  * Pattern "Composite"
  *
4  * @author Bonin
  * @version 1.1
6  */
package de.unilueneburg.as.composite;
8
public class Fahrrad extends Composite
10 {
    public Fahrrad(int id)
12     {
        super(id);
14     }

16     public String toString()
    {
18         return "Fahrrad" + super.toString();
    }
20 }

```

Listing 6.63: Rahmen

```

/**
2  * Pattern "Composite"
  *
4  * @author Bonin
  * @version 1.1
6  */
package de.unilueneburg.as.composite;
8
public class Rahmen extends Leaf
10 {
    public Rahmen(int id)
12     {

```

```

    super(id);
14 }
16 public String toString()
    {
18     return "Rahmen" + super.toString();
    }
20 }

```

Listing 6.64: Laufrad

```

/**
2  * Pattern "Composite"
  *
4  * @author    Bonin
  * @version   1.1
6  */
package de.unilueneburg.as.composite;
8
public class Laufrad extends Composite
10 {
    public Laufrad(int id)
12     {
        super(id);
14     }
16     public String toString()
    {
18         return "Laufrad" + super.toString();
    }
20 }

```

Listing 6.65: Felge

```

/**
2  * Pattern "Composite"
  *
4  * @author    Bonin
  * @version   1.1
6  */
package de.unilueneburg.as.composite;
8
public class Felge extends Leaf
10 {
    public Felge(int id)
12     {
        super(id);
14     }

```

```

16     public String toString()
17     {
18         return "Felge" + super.toString();
19     }
20 }

```

Listing 6.66: Nabe

```

/**
2  * Pattern "Composite"
3  *
4  * @author    Bonin
5  * @version   1.1
6  */
package de.unilueneburg.as.composite;
8
public class Nabe extends Leaf
10 {
11     public Nabe(int id)
12     {
13         super(id);
14     }
15
16     public String toString()
17     {
18         return "Nabe" + super.toString();
19     }
20 }

```

Listing 6.67: Comosite

```

/**
2  * Pattern "Composite"
3  *
4  * @author    Bonin
5  * @version   1.1
6  */
package de.unilueneburg.as.composite;
8
import java.util.LinkedList;
10
public class Composite implements Component
12 {
13
14     protected LinkedList children = new LinkedList();
15
16     protected int id = 0;
17
18

```

```

20     public Composite(int id)
21     {
22         this.id = id;
23     }
24
25     public String toString()
26     {
27         return "Composite id=" + id;
28     }
29
30     public void add(Component component)
31     {
32         this.children.add(component);
33     }
34
35     public void remove(Component component)
36     {
37         this.children.remove(component);
38     }
39
40     public Component getChild(int index)
41     {
42         return (Component) this.children.get(index);
43     }
44
45     public int getChildCount()
46     {
47         return this.children.size();
48     }
49 }

```

Listing 6.68: Leaf

```

1  /**
2   * Pattern "Composite"
3   *
4   * @author    Bonin
5   * @version   1.1
6   */
7  package de.unilueneburg.as.composite;
8
9  public class Leaf implements Component
10 {

```

```

12     protected int id = 0;
13
14     public Leaf(int id)
15     {
16         this.id = id;
17     }
18
19     public String toString()
20     {
21         return "mit Leaf id=" + id;
22     }
23
24     public void add(Component component)
25     {
26         // Interface Component requirement
27     }
28
29     public void remove(Component component)
30     {
31         // Interface Component requirement
32     }
33
34     public Component getChild(int index)
35     {
36         return null;
37     }
38
39     public int getChildCount()
40     {
41         return 0;
42     }

```

Listing 6.69: Component

```

/**
2  * Pattern "Composite"
3  *
4  * @author    Bonin
5  * @version   1.1
6  */
package de.unilueneburg.as.composite;
7
8  public interface Component
9  {
10     public void add(Component component);
11     public void remove(Component component);
12     public Component getChild(int index);
13     public int getChildCount();

```

```
}
```

Protokolldatei FahrradApp.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode)
```

```
D:\bonin\anwd\code>javac -Xlint
  de/unilueneburg/as/composite/Composite.java
de/unilueneburg/as/composite/Composite.java:33:
warning: [unchecked] unchecked call to add(E)
as a member of the raw type java.util.LinkedList
  this.children.add(component);
                    ^
1 warning
```

```
D:\bonin\anwd\code>javac
  de/unilueneburg/as/composite/FahrradApp.java
```

```
D:\bonin\anwd\code>java
  de.unilueneburg.as.composite.FahrradApp
Level: Fahrrad mit Composite id = 1
  Level: Rahmen mit Leaf id = 1
    Level: Laufrad mit Composite id = 2
      Level: Felge mit Leaf id = 2
        Level: Nabe mit Leaf id = 4
    Level: Laufrad mit Composite id = 3
      Level: Felge mit Leaf id = 3
        Level: Nabe mit Leaf id = 5
```

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

6.13 Komponentenmodelle

Der Begriff *Component Model* wird in vielfältigen Zusammenhängen verwendet auch in der JavaTM-Welt. Hier sind besonders zu unterscheiden:

1. (ursprüngliche) *JavaBeansTM* (↔ Abschnitt 6.13.1 S. 313)

Sie werden primär verwendet um GUI-Komponenten zu kombinieren. Sie sind jedoch kein Server-seitiges Modell.

2. EJB *Enterprise JavaBeansTM* (↔ Abschnitt 6.13.2 S. 319) CTM
 Sie werden primär verwendet als Server-seitiges Modell zum Kombinieren von Komponenten mit einer Transaktionsteuerung im Sinne eines (*Transaction Processing Monitor²⁶*(s)) verwendet.

Die ursprünglichen *JavaBeansTM* 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 *JavaBeansTM*, wie manchmal formuliert wird, sondern hat eine andere Aufgabe, nämlich die eines *Component Transaction Monitor*(s) (CTM²⁷).

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 genutzt 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.

6.13.1 JavaBeansTM

Die *Plug-In*-Bausteine, konzipiert für GUI-Zwecke, sind die ursprünglichen *JavaBeansTM*. Ihr Vertrag zielt auf eine einfache Anpassbarkeit und zwar mit Hilfe von Werkzeugen.

„A JavaBean is a reusable software component that can be

²⁶Ein 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 zeitkritische (*mission-critical*) Massenanwendungen entwickelt.

²⁷Der Begriff wurde 1999 von Anne Thomas (jetzt Ms. Manes) geprägt. (↔ [Monson01] p. 4)

manipulated visually in a builder tool.“ ([Vanderburg97] p. 578 oder [Flanagan97] p. 233).

BDK

Wenn zur Anpassung kein leistungsfähiges, kommerzielles *Builder*-Werkzeug²⁸ verfügbar ist, kann man *Beans Development Kit*²⁹ (BDK) von Sun Microsystems, Inc. USA, mit seiner Testbox genutzt werden.

Ein *JavaBean* ist ein normales Objekt³⁰, 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:

- `public FooType getFoo() {...}`
- `public boolean isFoo() {...}`
- `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) {...}`

Getter

Setter

Listener

2. Die Ereignisbehandlung basiert auf dem Delegationsmodell (*listener classes für events*, ↔ Abschnitt 6.2 S. 167). Dazu ist folgendes Paar von Methoden zu definieren:

²⁸Liste der Hersteller: <http://splash.javasoft.com/beans/tools.html> (Zugriff: 24-Mai-1998)

²⁹BDK Quelle: http://splash.javasoft.com/beans/bdk_download.html (Zugriff: 24-Mai-1998)

³⁰Auf 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` (\leftrightarrow Abschnitt 6.3 S. 182).
4. Verfügbar wird ein `JavaBean` als ein *Java Archiv (JAR)* mit einem sogenannten *Manifest* (JAR-Parameter `-m`, \leftrightarrow Seite 190). 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

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

³¹Hinweis: Auch auf Windows-Plattformen gilt hier der Schrägstrich (*slash*) und nicht der *Backslash*.

³²Für weitere Informationen zum Schreiben von eigenen *JavaBeansTM* siehe zum Beispiel [Vanderburg97].

bound

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 *JavaBeansTM* zu verbreiten, so ist es doch sinnvoll die Konstruktionsmuster und (Namens-)Regeln direkt zu übernehmen.

Listing 6.70: SimpleBean

```

/**
2  * Grundstruktur fuer JavaBean mit Kontrollmechanismus zum
3  * Update: ,,bound'' und ,,constrained''
4  * Idee aus
5  * Glenn Vanderburg; MAXIMUM Java 1.1, 1997, p. 597
6  *
7  * @since      23-May-1998, 29-May-1998, 29-May-2007
8  * @author     Hinrich E. G. Bonin
9  * @version    1.2
10 */
11 package de.leuphana.ics.beans;
12
13 import java.beans.PropertyChangeListener;
14 import java.beans.PropertyChangeSupport;
15 import java.beans.PropertyVetoException;
16 import java.beans.VetoableChangeListener;
17 import java.beans.VetoableChangeSupport;
18
19 import java.awt.Canvas;
20 import java.awt.Color;
21 import java.awt.Dimension;
22
23 public class SimpleBean extends Canvas
24 {
25     String myWitz = "Piep, ,piep, ,... ,lieb";

```

```
26     /*
27     *   bound-Eigenschaft
28     *   --- Automatisches Informieren ueber ein Update
29     */
30     private PropertyChangeSupport
31         changes = new PropertyChangeSupport( this );
32
33     /*
34     *   constrained-Eigenschaft
35     *   --- Vetomechanismus fuer ein Update
36     */
37     private VetoableChangeSupport
38         vetos = new VetoableChangeSupport( this );
39
40     public SimpleBean()
41     {
42         setBackground( Color.green );
43     }
44
45     public String getWitz()
46     {
47         return myWitz;
48     }
49
50     public void setWitz( String neuerWitz )
51         throws PropertyVetoException
52     {
53         String alterWitz = myWitz;
54         vetos.fireVetoableChange(
55             "Witz",
56             alterWitz,
57             neuerWitz );
58
59         /*
60         *   Kein Veto fuer das Update
61         */
62         myWitz = neuerWitz;
63
64         /*
65         *   Nachtraegliche Information ueber das Update
66         */
67         changes.firePropertyChange(
68             "Witz",
69             alterWitz,
70             neuerWitz );
71     }
72
73     /*
74     *   Vetomechanismus fuer das Update
75     *   mit VetoableChangeListener

```

```

    */
76  public void addVetoableChangeListener (
      VetoableChangeListener l)
78  {
      vetos.addVetoableChangeListener(l);
80  }

82  public void removeVetoableChangeListener (
      VetoableChangeListener l)
84  {
      vetos.removeVetoableChangeListener(l);
86  }

88  /*
      * Updateinformation mit
90  * PropertyChangeListener
      */
92  public void addPropertyChangeListener (
      PropertyChangeListener l)
94  {
      changes.addPropertyChangeListener(l);
96  }

98  public void removePropertyChangeListener (
      PropertyChangeListener l)
100 {
      changes.removePropertyChangeListener(l);
102 }

104 /*
      * Sonstige exportierte Methode
106 */
      public Dimension getMinimuSize ()
108 {
      return new Dimension(100, 150);
110 }
}

```

Protokolldatei SimpleBean.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

```

```
D:\bonin\anwd\code>javac
  de\leuphana\ics\beans\SimpleBean.java

D:\bonin\anwd\code>dir
  de\leuphana\ics\beans\SimpleBean.class
  1.661 SimpleBean.class

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

6.13.2 EJB (*Enterprise JavaBeansTM*)

*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 JMS³³-basierte Server-seitige Komponenten, die asynchrone Nachrichten bearbeiten.

Die Typunterscheidung ist aufgrund der Flexibilität der Beans unscharf. Eine charakteristisches Unterscheidungsmerkmal ist die Persistenz³⁴. *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 6.28 S. 321):

- `RemoteInterface` extends `javax.ejb.EJBObject`
Entity Beans und Session Beans definieren mit diesem Interface

³³JMS ≡ *Java Messaging Service*

³⁴Zum Begriff *Persistenz* ↔ Abschnitt 6.3 S. 182.

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 *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 *Life-cycle Methods* für andere Beans im gleichen EJB-Container.
Namensbeispiel: `RaumHomeLocal`
- `EntityBeanClass` implements `javax.ejb.EntityBean`
Ein Entity Bean muß dieses Interface implementieren.
Namensbeispiel: `RaumBean`
- `SessionBeanClass` implements `javax.ejb.SessionBean`
Ein Session Bean muß dieses Interface implementieren.
Namensbeispiel: `WartungBean`
- `MessageDrivenBean` implements `javax.ejb.MessageDrivenBean`, `javax.jms.MessageListener`
Ein Message-driven Bean muß beide Interfaces implementieren.
- `BeanClass` extends `javax.ejb.EnterpriseBean`
Ein Entity Bean, Session Bean und ein Message-driven Bean erben jeweils von dieser Klasse.
Namensbeispiel: `RaumBean`

Üblicherweise³⁵ bezeichnet der Begriff *Enterprise Bean* (oder auch kurz

³⁵Siehe zum Beispiel ↔ [Monson01].

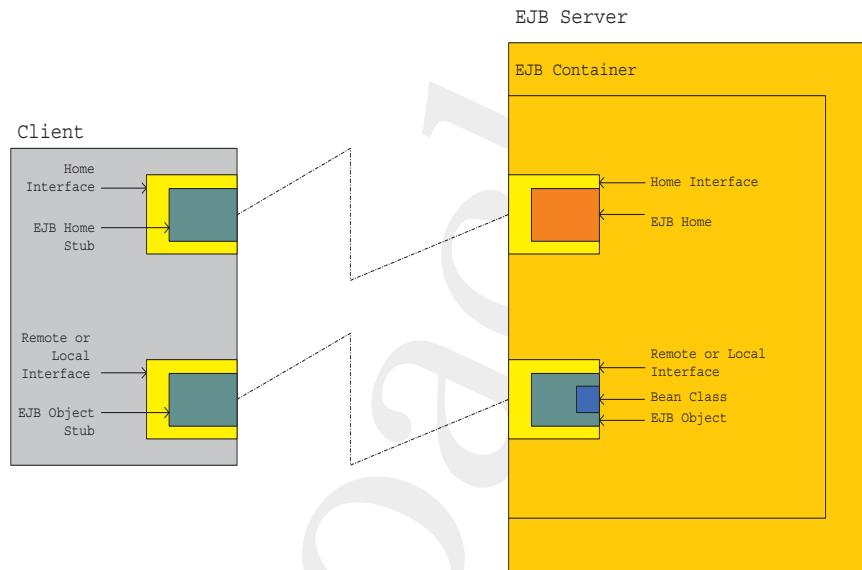


Abbildung 6.28: Skizze der *Enterprise JavaBeans™* -Architektur

Bean) jeden der obigen *Bean*-Typen. Dieser Begriff wird oft als EJB notiert, so auch im *JAVA™ –COACH*. Zum Beispiel bezeichnet „Raum EJB“ ein *Enterprise Bean* mit allen Teilen, also mit den *Component Interfaces* und Klassen. Soll nur das *Remote Interface* angegeben 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.

Die *Business Methodes* sind in der Form von Signaturen im *Remote Interface* angegeben.

Listing 6.71: RaumRemote

```

/**
 2  * Einfaches EJB Beispiel
 3  *
 4  * @since      19-Dec-2002
 5  * @author     Hinrich Bonin
 6  * @version    1.0
 7  */
 8
 9  package de.fhnon.ejb.raum;
10
11  import java.rmi.RemoteException;
12
13  public interface RaumRemote
14      extends javax.ejb.EJBObject
15  {
16      public String getBezeichnung()
17          throws RemoteException;
18
19
20      public void setBezeichnung(String bezeichnung)
21          throws RemoteException;
22
23
24      public int getAnzahlPlaetze()
25          throws RemoteException;
26
27
28      public void setAnzahlPlaetze(int anzahlPlaetze)
29          throws RemoteException;
30
31
32      public abstract Integer getId()
33          throws RemoteException;
34
35
36      public abstract void setId(Integer id)
37          throws RemoteException;
38  }

```

Die *Life-cycle Methods* sind im *Remote Home Interface* definiert. Dabei ist die Methode `create()` zuständig für die Initialisierung einer Bean-Instanz.

Listing 6.72: RaumHomeRemote

```

/**
2  * Einfaches EJB Beispiel
  *
4  * @since    19-Dec-2002
  * @author   Hinrich Bonin
6  * @version  1.0
  */
8
package de.fhnon.ejb.raum;
10
import java.rmi.RemoteException;
12 import javax.ejb.CreateException;
import javax.ejb.FinderException;
14
public interface RaumHomeRemote
16     extends javax.ejb.EJBHome
17 {
18     public RaumRemote create(Integer id)
19         throws CreateException, RemoteException;
20
21
22     public RaumRemote findByPrimaryKey(
23         Integer primaryKey)
24         throws FinderException, RemoteException;
25 }

```

Listing 6.73: RaumLocal

```

/**
2  * Einfaches EJB Beispiel
  *
4  * @since    19-Dec-2002
  * @author   Hinrich Bonin
6  * @version  1.0
  */
8
package de.fhnon.ejb.raum;
10
public interface RaumLocal
12     extends javax.ejb.EJBLocalObject
13 {
14 }

```

Listing 6.74: RaumHomeLocal

```

/**
2  * Einfaches EJB Beispiel
  *

```

```

4  * @since      19-Dec-2002
   * @author     Hinrich Bonin
6  * @version    1.0
   */
8
9  package de.fhnon.ejb.raum;
10
11 import javax.ejb.CreateException;
12 import javax.ejb.FinderException;
13
14 public interface RaumHomeLocal
15     extends javax.ejb.EJBLocalHome
16 {
17     public RaumLocal create(Integer id)
18         throws CreateException;
19
20
21     public RaumLocal findByPrimaryKey(
22         Integer primaryKey)
23         throws FinderException;
24 }

```

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 Entity Bean*³⁶.

Listing 6.75: RaumBean

```

/**
2  * Einfaches EJB Beispiel
   *
4  * @since      19-Dec-2002
   * @author     Hinrich Bonin
6  * @version    1.0
   */
8
9  package de.fhnon.ejb.raum;
10
11 import javax.ejb.EntityContext;
12
13 public abstract class RaumBean implements
14     javax.ejb.EntityBean
15 {
16
17     public Integer ejbCreate(Integer id)
18     {
19         setId(id);

```

³⁶Näheres dazu siehe EJB-Literatur zum Beispiel ↔ [Monson01],

```
20     return null;
21 }
22
23
24 public void.ejbPostCreate(Integer id)
25 {
26     // do nothing
27 }
28
29
30 public abstract String getBezeichnung();
31
32
33 public abstract void setBezeichnung(
34     String bezeichnung);
35
36
37 public abstract int getAnzahlPlaetze();
38
39
40 public abstract void setAnzahlPlaetze(
41     int anzahlPlaetze);
42
43
44 public abstract Integer getId();
45
46
47 public abstract void setId(Integer id);
48
49
50 public void unsetEntityContext()
51 {
52     // not implemented
53 }
54
55
56 public void .ejbActivate()
57 {
58     // not implemented
59 }
60
61
62 public void .ejbPassivate()
63 {
64     // not implemented
65 }
66
67
68 public void .ejbLoad()
```

```

70     {
71         // not implemented
72     }
73
74     public void ejbStore ()
75     {
76         // not implemented
77     }
78
79
80     public void ejbRemove ()
81     {
82         // not implemented
83     }
84 }

```

Der Client wird mit dem EJB Server verbunden. Um auf ein *Enterprise 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.

Listing 6.76: ClientA

```

/**
2  * Einfaches EJB Beispiel
3  *
4  * @since      19-Dec-2002
5  * @author     Hinrich Bonin
6  * @version    1.0
7  */
8
9  package de.fhnon.ejb.raum;
10
11  import java.rmi.RemoteException;
12  import java.util.Properties;
13  import de.fhnon.ejb.raum.RaumHomeRemote;
14  import de.fhnon.ejb.raum.RaumRemote;
15
16  import javax.ejb.CreateException;
17  import javax.ejb.FinderException;
18
19  import javax.naming.InitialContext;
20  import javax.naming.Context;
21  import javax.naming.NamingException;

```

```
22
23 import javax.rmi.PortableRemoteObject;
24
25 public class ClientA
26 {
27     public static void main(String [] args)
28     {
29         try
30         {
31             Context jndiContext =
32                 getInitialContext ();
33             Object ref =
34                 jndiContext.lookup ("RaumHomeRemote");
35             RaumHomeRemote home = (RaumHomeRemote)
36                 PortableRemoteObject.narrow (
37                     ref, RaumHomeRemote.class);
38             RaumRemote raum1 =
39                 home.create (new Integer (1));
40             raum1.setBezeichnung ("Grosser_Saal");
41             raum1.setAnzahlPlaetze (120);
42             raum1.setld (new Integer (1));
43
44             RaumRemote raum2 =
45                 home.findByPrimaryKey (new Integer (2));
46             System.out.println (raum2.getld () + ":\u2013" +
47                 raum2.getBezeichnung () + "\u2013hat\u2013" +
48                 raum2.getAnzahlPlaetze () +
49                 "\u2013Plaetze.");
50         } catch (RemoteException re)
51         {
52             re.printStackTrace ();
53         } catch (NamingException ne)
54         {
55             ne.printStackTrace ();
56         } catch (CreateException ce)
57         {
58             ce.printStackTrace ();
59         } catch (FinderException fe)
60         {
61             fe.printStackTrace ();
62         }
63     }
64 }
65
66
67 public static Context getInitialContext ()
68     throws NamingException
69 {
70
```

```

72     Properties p = new Properties ();
73     /*
74     *   Define the JNDI properties
75     *   specific to the vendor
76     *   here for example: IBM WebSphere
77     */
78     p.put (Context.PROVIDER_URL, "iiop:///");
79     p.put (Context.INITIAL_CONTEXT_FACTORY,
80           "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
81     return new InitialContext (p);
82 }
83
84 }

```

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 6.3.3 S. 190) gepackt.

Zu Beginn einer XML-Datei wird der Dokumententyp angegeben und zwar mit der `<!DOCTYPE>`-Angabe für das Wurzelement, hier `<ejb-jar>`. Diese Angabe umfaßt:

- 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`

Listing 6.77: `ejb-jar.xml`

```

2 <!DOCTYPE eib-jar PUBLIC
  " -//Sun_Microsystems, Inc //DTD_Enterprise_JavaBeans_2.0//EN"

```

JAR

DTD


```
    "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
4 <ejb-jar>
    <enterprise-beans>
6     <entity>
        <ejb-name>RaumEJB</ejb-name>
8         <home>de.fhnon.ejb.raum.RaumHomeRemote</home>
        <remote>de.fhnon.ejb.raum.RaumRemote</remote>
10        <local-home>de.fhnon.ejb.raum.RaumHomeLocal</local-home>
        <local>de.fhnon.eib.raum.RaumLocal</local>
12        <ejb-class>de.fhnon.ejb.raum.RaumBean</ejb-class>
        <persistence-type>Container</persistence-type>
14        <prim-key-class>java.lang.Integer</prim-key-class>
        <reentrant>False</reentrant>
16    </entity>
    </enterprise-beans>
18 </ejb-jar>
```

Java-Coach

Kapitel 7

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 und durchschaubar 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. Realfähigkeit (Einhaltung von definierten Verarbeitungszeiten)
3. Unterbrechungsfreier Betrieb ($365 \frac{\text{Tage}}{\text{Jahr}}$ mit $24 \frac{\text{h}}{\text{Tag}}$)
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 333 ...
 - den Einsatz von Werkzeugen für spezielle Aufgaben
↪ Seite 73 ...
 - die Notwendigkeit von Code-Konventionen und gibt Empfehlungen,
↪ Seite 355 ...
 - die Idee von vorgefertigten Geschäftsobjekten (*Common Business Objects*) und Geschäftsvorgängen (*Core Business Processes*) und
↪ Seite 378 ...
 - Alternativen für die Quellcodegestaltung aus Sicht der Performance.
↪ Seite 361 ...
-

7.1 Einsatz einer teamfähigen IDE

Im November 2001 brachte IBM, Object Technology International (OTI) mit acht anderen Unternehmen *Eclipse* auf den Markt. Eclipse versucht das proprietäre Muster zu überwinden. Eclipse ist eine Integrationsplattform und zwar als ein frei fügbares Werkzeug. Eclipse kann problemlos mit anderen Entwicklungswerkzeugen 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.

Der Name *Eclipse* (deutsch „Verdunkelung“) umfaßt drei Aspekte: **Eclipse**

IBM

- 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).

7.1.1 Eclipse — Überblick

Eclipse hält Projekte, Verzeichnisse und Dateien (\equiv 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:

```
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 Tomcat aus dem Apache Projekt um das gewünschte Dokument auszuliefern.

work-
space

Tomcat

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**.

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 zugehörige Information zu sehen. Die URL enthält eine Portnummer, in unserem Fall die Portnummer 6969, die zur Kommunikation mit der Servletmaschine benutzt wird. Eclipse konfiguriert die Servletmaschine für eine dynamische 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 sowie 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.

7.1.2 Eclipse — Edieren

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.

Die Super- und Subtypen einer Klasse oder eines Interfaces können in einer Hierarchie angezeigt werden. Dazu markiert man ein Element, beispielsweise eine Java-Quelldatei, 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.

7.1.3 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

Debug

Typen-
hierar-
chie

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 Einfügesteuerer 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 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**

**Watch-
point**

aus dem Kontextmenue. Man editiere die Eigenschaften eines Unterbrechungspunktes wenn der Beobachtungspunkt die Ausführung nur beim Zugriff, nur bei der Modifikation oder in beiden Fällen aussetzen soll. Beobachtungspunkte können zusätzlich Trefferzähler und Randbedingungen haben.

7.1.4 Eclipse — CVS (Concurrent Versions System)

Eclipse stellt eine direkte Unterstützung für ein spezielles Repository bereit, das sogenannte *Concurrent Versions System* (CVS). CVS ist ein häufig genutztes Repository mit offenem Quellcode¹. 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 Eclipse-Team 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 7.2 S. 340 zeigt exemplarisch zwei CVS-Repositories mit denen Eclipse verbunden ist. Die Abbildung 7.1 S. 339 zeigt die CVS-Verknüpfung der einzelnen Projekt.

CVS ist 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 vielen 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*.

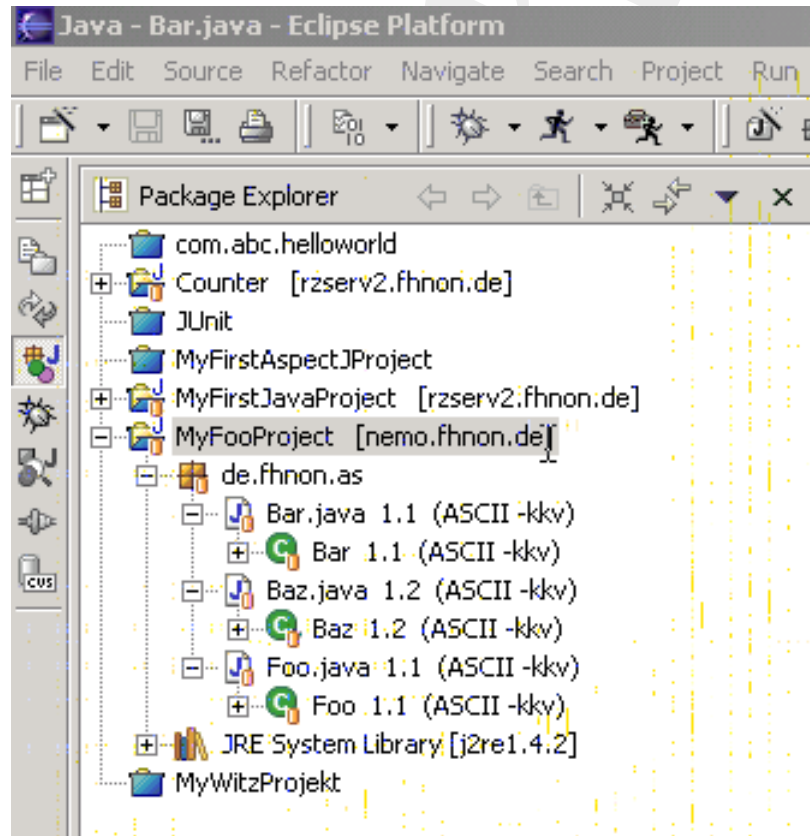
CVS ist verfügbar auf verschiedenen Plattformen, dazu gehören Windows-, UNIX- und Linux-Systeme. Für weitere Informationen über CVS

¹CVS unterliegt der *GNU General Public License*. Eine Kopie der Lizenzbedingungen enthält der *CVS distribution kit*.

²RCS-Web-Site <http://www.gnu.org/software/rcs/rcs.html> (online 11-Nov-2003).

CVS

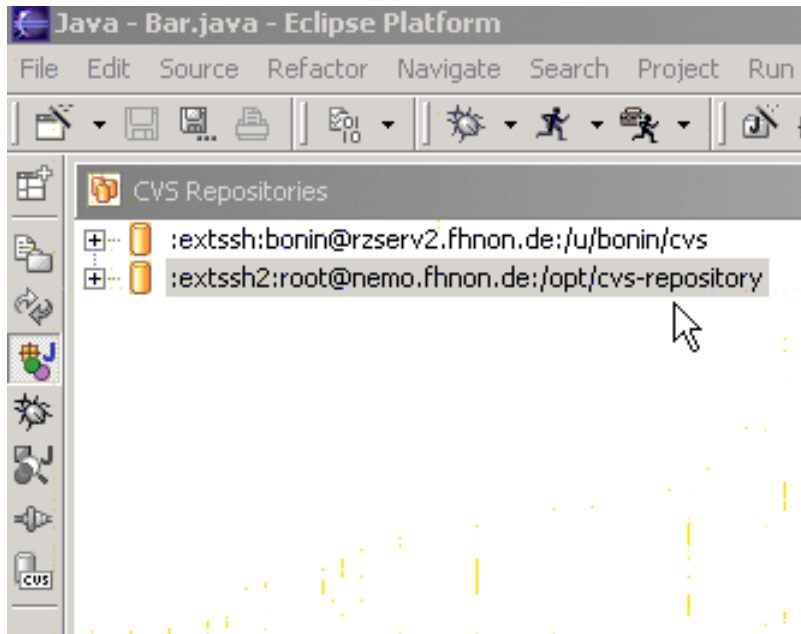
RCS



Legende:

Dieses *Package-Explorer*-Beispiel zeigt mehrere Projekt. Das Projekt MyFoo-Project setzt sich aus Klassen mit unterschiedlichen 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 (\leftrightarrow Abbildung 7.2 S. 340) verwaltet werden.

Abbildung 7.1: Eclipse — Package Explorer



Legende:

Dieses *CVS Repositories*-Beispiel verweist auf zwei Reopositories:

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 7.2: Eclipse — CVS Repositories

siehe `ccvs.cvshome.org`.

Wie jedes *Repository* (\approx 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 leistungsfähigere Repositories³ 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#
```

Dieses Repository wurde im Betriebssystem Linux (Debian) mit folgenden Kommandos eingerichtet:⁴ **CVSROOT**

```
>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:

```
CVSROOT=/opt/cvs-repository
export CVSROOT
```

³Beispielsweise hat CVS keine Funktionen wie Defekt- oder Funktionsnachweis und auch keine Zusammenbaufähigkeit oder Unterstützung des Entwicklungsprozesses.

⁴Hinweis: Für Studierende der Wirtschaftsinformatik wurde ein Repository zum Experimentieren auf dem Rechner `nemo.uni-lueneburg.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 Rechenzentrums der Leuphana Universität Lüneburg (vormals RZ der Fachhochschule Nordostniedersachsen (FHNON)) eingesetzt werden. Für andere Zwecke darf die Kennung auf dem Rechner nicht genutzt werden.

Mit der erfolgreichen Repository-Initiierung ist ein Verzeichnis:

```
/opt/cvs-repository/CVSRROOT
```

entstanden.

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 7.1 S. 343 gibt die CVS-Kommandos an und demonstriert damit das CVS-Leistungsspektrum.

7.1.5 Eclipse — Refactoring

Unter *Refactoring* versteht man eine Überarbeitung von vorhandenem Quellcode (\leftrightarrow [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.

7.2 Einsatz von speziellen Werkzeugen

7.2.1 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

⁵CVS-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 <code>.cvspass</code> 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 7.1: CVS-Repository — Kommandos

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-Datei⁶ befindet sich die JAR-Datei `.junit.jar`. Sie ist in den Klassenpfad einzufügen (siehe Protokolldatei `TestString.log`, S. 345).

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 Methoden `length()`, `toUpperCase()` und `toLowerCase()` der Klasse `String` aus dem Java-Paket `java.lang`.

Wir nutzen ein einfaches Interface auf Basis von Textausgaben in Kommandozeilen (\leftrightarrow `TestString.log` S.345).

Listing 7.1: `TestString`

```

/**
2  * JUnit Beispiel:
3  * Test der Klasse java.lang.String
4  *
5  *
6  * @since      26-Nov-2004, 30-May-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    1.1
9  */
10
11 package de.leuphana.ics.test;
12
13 import junit.framework.TestCase;
14 import junit.textui.TestRunner;
15 import junit.framework.TestSuite;
16
17 public class TestString extends TestCase
18 {
19     static String myText =
20         new String("Alles klar?");
21     static int myLength = 11;
22
23     public TestString(String name)
24     {
25         super(name);
26     }

```

⁶hier `junit3.8.1.zip`


```

28     public void testLength ()
29     {
30         assertEquals (myText.length (), myLength);
31     }
32
33     public void testToUpperCase ()
34     {
35         assertTrue (myText.toUpperCase (). equals (
36             new String ("ALLES_KLAR?")));
37     }
38
39     public void testToLowerCase ()
40     {
41         assertTrue (myText.toLowerCase (). equals (
42             new String ("alles_klar?")));
43     }
44
45
46     static TestSuite suite () throws Exception
47     {
48         return new TestSuite (TestString.class);
49     }
50
51     public static void main (String [] args) throws Exception
52     {
53         TestRunner.run (TestString.suite ());
54     }
55 }

```

Protokolldatei TestString.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>echo %CLASSPATH%
echo %CLASSPATH%
.;...;
c:\programme\junit4.3.1\junit-4.3.1.jar;
c:\programme\junit4.3.1\junit-4.3.1-src.jar

D:\bonin\anwd\code>javac de/leuphana/ics/test/TestString.java

D:\bonin\anwd\code>java de.leuphana.ics.test.TestString

```

```
...
```

```
Time: 0
```

```
OK (3 tests)
```

```
D:\bonin\anwd\code>REM myLength = 13;
```

```
D:\bonin\anwd\code>javac de/leuphana/ics/test/TestString.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.test.TestString  
.F..
```

```
Time: 0
```

```
There was 1 failure:
```

```
1) testLength(de.leuphana.ics.test.TestString)  
   junit.framework.AssertionFailedError:  
     expected:<11> but was:<13>  
at de.leuphana.ics.test.TestString.testLength  
  (TestString.java:30)  
at sun.reflect.NativeMethodAccessorImpl.invoke0  
  (Native Method)  
at sun.reflect.NativeMethodAccessorImpl.invoke  
  (Unknown Source)  
at sun.reflect.DelegatingMethodAccessorImpl.invoke  
  (Unknown Source)  
at de.leuphana.ics.test.TestString.main  
  (TestString.java:53)
```

```
FAILURES!!!
```

```
Tests run: 3, Failures: 1, Errors: 0
```

```
D:\bonin\anwd\code>REM myLength = 11;
```

```
D:\bonin\anwd\code>javac  
  de/leuphana/ics/test/TestString.java
```

```
D:\bonin\anwd\code>java de.leuphana.ics.test.TestString
```

```
...
```

```
Time: 0
```

```
OK (3 tests)
```

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

```
C:
```

```
C:\>cd Programme/junit4.3.1
```

```
C:\Programme\junit4.3.1>java  
  org.junit.runner.JUnitCore org.junit.tests.AllTests
```


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()); }
```

7.2.2 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 Betriebssystem bezogen sind, vermeidet Ant diese Betriebssystemabhängigkeit. 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>
```

Als Beispiel verwenden wir die obige Klasse `TestString` (↔ S. 344). 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. 350). Das dabei erzeugte Manifest zeigt Abbildung 7.3 S. 349.

⁷Steht hier beispielhaft für `make`, `gnumake`, `nmake` oder `jam`



Legende:

Manifest in JAR-Datei dist/lib/MyTestString-20040613.jar

Abbildung 7.3: Mit Ant erzeugtes Manifest

Listing 7.2: build.xml

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <!-- Bonin 12-Jun-2004 -->
3 <project name="MyTestString" default="dist" basedir=".">
4   <description>
5     simple example build file
6   </description>
7   <!-- set global properties for this build -->
8   <property name="src" location="de/fhnon/as/test"/>
9   <property name="build" location="build"/>
10  <property name="dist" location="dist"/>
11
12  <target name="init">
13    <!-- Create the time stamp -->
14    <tstamp/>
15    <!-- Create the build directory structure
16     used by compile -->
17    <mkdir dir="${build}"/>
18  </target>
19
20  <target name="compile" depends="init"
21    description="compile the source" >
22    <!-- Compile the java code
23     from ${src} into ${build} -->
24    <javac srcdir="${src}" destdir="${build}"/>
25  </target>
26
27  <target name="dist" depends="compile"
28    description="generate the distribution" >
29    <!-- Create the distribution directory -->

```

```

30     <mkdir dir="${dist}/lib"/>
32     <!-- Put everything in ${build} into
33     the MyTestString-${DSTAMP}.jar file -->
34     <jar jarfile=
35         "${dist}/lib/MyTestString-${DSTAMP}.jar"
36         basedir="${build}"/>
37     </target>
38
39     <target name="clean"
40         description="clean up" >
41         <!-- Delete the ${build} and ${dist}
42         directory trees -->
43         <delete dir="${build}"/>
44         <delete dir="${dist}"/>
45     </target>
46 </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%;/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)

D:\bonin\anwd\code>

```

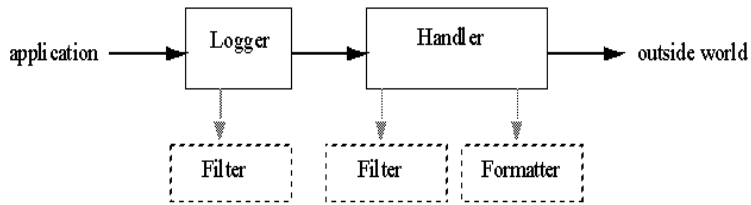
7.2.3 Logging — java.util.logging

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 7.4 S. 352 skizziert diesen Zusammenhang.

Ein einfaches Beispiel für die Nutzung des Logging-Paketes

```
java.util.logging
```

stellt die Klasse *LogSample* (S. 352) dar. Die Ausgabe erfolgt dabei im XML-Format in die Datei *loggingResult.xml* (S. 353). Diese



Legende:

Quelle ↔ <http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html>
(online 16-Jun-2004)

Abbildung 7.4: Logging-Übersicht — `java.util.logging`

XML-Datei entspricht der *Document Type Definition* (DTD) `logger.dtd` (S. 353).

Listing 7.3: LogSample

```

/**
2  * Logging Example
3  *
4  * @since      16-Jun-2004, 30-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8  package de.leuphana.ics.logging;

10 import java.util.logging.Level;
11 import java.util.logging.Logger;
12 import java.util.logging.FileHandler;
13 import java.io.IOException;
14
15 public class LogSample
16 {
17     private static Logger logger =
18         Logger.getLogger("de.leuphana.ics.logging");
19     private static String resultFile =
20         "./de/leuphana/ics/logging/loggingResult.xml";

22     public String working()
23     {
24         return new String("Doing something!");
25     }

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



```

28     throws IOException
    {
30         System.out.println (
            new LogSample (). working ());
32
34         FileHandler fh =
            new FileHandler (resultFile );
36         logger.addHandler (fh);
38         logger.setLevel (Level.ALL);
39         logger.log (Level.FINEST,
40             "Testing - Level = FINEST");
    }

```

Listing 7.4: loggingResult.xml

```

<?xml version="1.0" encoding="windows-1252" standalone="no"?>
2 <!DOCTYPE log SYSTEM "logger.dtd">
<log>
4 <record>
    <date>2007-05-30T22:48:46</date>
6    <millis>1180558126515</millis>
    <sequence>0</sequence>
8    <logger>de.leuphana.ics.logging</logger>
    <level>FINEST</level>
10   <class>de.leuphana.ics.logging.LogSample</class>
    <method>main</method>
12   <thread>10</thread>
    <message>Testing - Level = FINEST</message>
14 </record>
</log>

```

Listing 7.5: logger.dtd

```

<?xml version="1.0" encoding="utf-8"?>
2
3 <!-- DTD used by the java.util.logging.XMLFormatter -->
4 <!-- This provides an XML formatted log message. -->
5
6 <!-- The document type is "log"
7 which consists of a sequence
8 of record elements -->
9
10 <!ELEMENT log (record*)>
11
12 <!-- Each logging call is
13 described by a record element. -->
14 <!ELEMENT record (date, millis, sequence, logger?, level,
    class?, method?, thread?, message, key?, catalog?, param*,

```

```

16 exception?>

18 <!-- Date and time when LogRecord was
    created in ISO 8601 format -->
20 <!ELEMENT date (#PCDATA)>

22 <!-- Time when LogRecord was
    created in milliseconds since
24 midnight January 1st, 1970, UTC. -->
    <!ELEMENT millis (#PCDATA)>

26 <!-- Unique sequence number within source VM. -->
28 <!ELEMENT sequence (#PCDATA)>

30 <!-- Name of source Logger object. -->
    <!ELEMENT logger (#PCDATA)>
32

34 <!-- 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". -->
36 <!ELEMENT level (#PCDATA)>

38 <!-- Fully qualified name of class that issued
    logging call, e.g. "javax.marsupial.Wombat". -->
40 <!ELEMENT class (#PCDATA)>

42 <!-- Name of method that issued logging call.
    It may be either an unqualified method name such as
44 "fred" or it may include argument type information
    in parenthesis, for example "fred(int, String)". -->
46 <!ELEMENT method (#PCDATA)>

48 <!-- Integer thread ID. -->
    <!ELEMENT thread (#PCDATA)>
50

52 <!-- The message element contains the text string
    of a log message. -->
    <!ELEMENT message (#PCDATA)>
54

56 <!-- If the message string was localized, the key element provides
    the original localization message key. -->
    <!ELEMENT key (#PCDATA)>
58

60 <!-- If the message string was localized,
    the catalog element provides
    the logger's localization_resource_bundle_name -->
62 <!ELEMENT catalog (#PCDATA)>

64 <!-- If the message string was localized,

```

```

each_of_the_param_elements
66 provides the String value (obtained using Object.toString())
of the corresponding LogRecord parameter. -->
68 <!ELEMENT param (#PCDATA)>

70 <!-- An exception consists of an optional
message string followed
72 by a series of StackFrames. Exception elements are used
for Java exceptions and other java Throwables. -->
74 <!ELEMENT exception (message?, frame+)>

76 <!-- A frame describes one line in a Throwable backtrace. -->
<!ELEMENT frame (class, method, line?)>
78
<!-- an integer line number within a class's source file. -->
80 <!ELEMENT line (#PCDATA)>

```

Protokolldatei LogSample.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de/leuphana/ics/logging/LogSample.java

D:\bonin\anwd\code>java
de.leuphana.ics.logging.LogSample
Doing something!

D:\bonin\anwd\code>

```

7.3 Konventionen zur Transparenz

7.3.1 Code-Konventionen

Code-Konventionen sind aus einer Menge von Gründen bedeutsam⁸:

⁸Die 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)

- Der wesentlich Anteil der Kosten eines Softwareproduktes ($\approx 80\%$), 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 wesentlich schneller einfinden und das Programm leichter verstehen.

Allgemeinde Regeln

Suffixes Eine Java-Quellcodedatei hat grundsätzlich den Suffix `.java`. Eine Java-Bytedatedatei hat den Suffix `.class`.

Reihenfolge Eine Java-Quellcodedatei ist wie folgt gegliederte:

1. Anfangskommentar
2. `package`-Statement
3. `import`-Statements
4. `class`-Deklaration(en)
5. `interface`-Deklaration(en)

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 JavaTM besteht

Struktur

Kommentar

aus einer beliebig langen Folge⁹ 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 3.6 S. 60ff. Ein Paketname ist stets in kleinen Buchstaben definiert. Damit ergibt sich auch der entsprechende Pfad (*directories*) in kleinen Buchstaben.

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 ...**

⁹Die reservierten Schlüsselwörter (\leftrightarrow Tabelle 5.4 S. 145) dürfen nicht als Bezeichner genutzt werden.

ren:

```
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 ein `break`-Konstrukt oder einen entsprechenden Kommentar, wie zum Beispiel:

```
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

Listing 7.6: SourceFileExample

```

/*
2  * @(#) SourceFileExample
  * 1.4 12-Oct-2003, 30-May-2007
4  * Copyright (c) 2002-2007 Nemo, Inc.
  * An der Eulenburg 6,
6  * D-21391 Reppenstedt, Germany
  */
8  package de.leuphana.ics.codeconvention;

10 import java.*;
  /**
12  * Class JavaSourceFileExample description goes
  * here.
14  *
  * @author      Firstname Lastname
16  * @created    12. Oktober 2003
  * @version    1.3 12-Oct-2003, 30-May-2007
18  */
  public class SourceFileExample extends Object
20  {
  /*
22  * A class implementation

```

```
24     * comment can go here.  
24     */  
26     /**  
26     * Class variable1 documentation comment  
26     */  
28     public static final int CLASSVAR1 = 0;  
  
30     /*  
30     * Class variable2 documentation comment  
30     */  
32     private static int classVar2;  
32     /*  
34     * Instance variable 1 documentation comment  
34     */  
36     public Object instanceVar1;  
36     /*  
38     * Instance variable 2 documentation comment  
38     */  
40     protected Object instanceVar2;  
40     /*  
42     * Instance variable 3 documentation comment  
42     */  
44     private Object instanceVar3;  
44     /*  
46     /**  
46     * Getter description  
46     *  
48     * @return The instanceVar1 value  
48     */  
50     public Object getInstanceVar1()  
50     {  
52     return instanceVar1;  
52     }  
54     /**  
54     * Setter description  
54     *  
56     * @param instanceVar1 The new value of  
56     * instanceVar1 value  
56     */  
58     public void setInstanceVar1(Object instanceVar1)  
58     {  
60     this.instanceVar1 = instanceVar1;  
60     }  
62     /**  
62     * Constructor SourceFileExample documentation  
62     * comment  
62     */
```



```

72  public SourceFileExample () { }
74  /**
76   * Method doSomething () documentation comment
78   * @param parameter1 description
80   * @param parameter2 description
82   * @return          returnValue description
84   */
86  public boolean doSomething (
88      Object parameter1 ,
90      Object parameter2)
92  {
94      // ... implementation goes here ...
96      return true;
98  }
100 /**
    * Method main () documentation comment
    * @param args The command line arguments
    */
    public static void main (String [] args)
    {
        System.out.println (" Start_of_work... ");
        // ... implementation goes here ...
        System.out.println (" ...End_of_work");
    }

```

7.3.2 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 7.1 S. 344. 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 (\hookrightarrow Protokolldatei 7.3.2 S. 370). Die Abbildung 7.5 S. 372 zeigt einen Ergebnisausschnitt.

Anhand des folgenden Beispiels einer rekursiv definierten Funktion werden die Quellcodeunterschiede verdeutlicht.

$$g = \begin{cases} 0 & : x \leq 0 \\ \frac{a}{2} * x & : 0 < x \leq \frac{b}{2} \\ -\frac{a}{2} * x + 2a & : \frac{b}{2} < x \leq b \\ g(x - b) & : b < x \end{cases}$$

Die schnelle, scheinbar sehr transparente Lösung mit der Klasse F (\hookrightarrow S. 362) 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`).

Listing 7.7: F — So nicht!

```

package de.leuphana.ics.function;
2
public abstract class F
4 {
    public static double g(double x, double a, double b)
6 {
8     if (x <= 0.0)
        {
10         return 0.0;
        } else if (x <= b / 2)
12         {
            return (a / (b / 2)) * x;
14         } else if (x <= b)
            {
16         return (-a / (b / 2)) * x + 2 * a;
            } else
18         {
            return g(x - b, a, b);
20         }
        }
22

```

```

24     public static double g(double x)
25     {
26         return g(x, 1.0, 2 * java.lang.Math.PI);
27     }
28     public static double g(double x, double a)
29     {
30         return g(x, a, 2 * java.lang.Math.PI);
31     }
32     public static void main(String [] args)
33     {
34         System.out.println(g(4.5) + "\n" +
35                             g(4.5, 2.0, 2.0));
36     }
37 }

```

Die Aufteilung in die folgenden drei Klassen ist zweckmäßig:

- `Math` (\leftrightarrow S. 366) bildet prinzipiell die obige Funktion `g` (\leftrightarrow S. 362) ab.
Der Methodename `saw` und die Parameternamen `toothHeight` und `toothWidth` vermitteln ihre Bedeutung unmittelbar — haben also mehr Semantik. Die Tabelle 7.2 S. 365 ist daher der bessere Ausgangspunkt.
- `MathValue` enthält die Prüfmethode und die Ersatzwerte bei fehlenden Angaben (\leftrightarrow S. 368).
- `MathProg` enthält die Testbeispiele (\leftrightarrow S. 369).

Nützlich ist die schriftliche Dokumentierung der Anforderungen (*Requirements*), die von der Software, hier primär von der Methode `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 7.2 S. 365.
- 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 7.2 S. 365 der Rechner zu wenig Speicherplatz hat, ist die Zahnhöhe NaN, als Not a Number.

$$\text{saw}(x) = \begin{cases} 0 & : x \leq 0 \\ \frac{\text{toothHeight}}{\text{toothWidth}} * x & : 0 < x \leq \frac{\text{toothWidth}}{2} \\ -\frac{\text{toothHeight}}{\text{toothWidth}} * x + 2 * \text{toothHeight} & : \frac{\text{toothWidth}}{2} < x \leq \text{toothWidth} \\ \text{saw}(x - \text{toothWidth}) & : \text{toothWidth} < x \end{cases}$$

Tabelle 7.2: Math . saw (x) -Formel

Listing 7.8: Math

```

/**
2  * <code>Math</code> class with the recursive specified
3  * <code>saw</code> method The saw has some tooth. A tooth
4  * is symetric construct. It has a height and a width. The
5  * method computes the height at a tooth position. The
6  * default values for tooth height and tooth width contains
7  * the class <code>MathValue</code>
8  *
9  * @since      06-Apr-2004, 30-May-2007
10 * @author     Hinrich E. G. Bonin
11 * @version    1.1
12 */
package de.leuphana.ics.function;
14
public abstract class Math
16 {
17     /**
18      * <code>saw</code> computes the value of the height at
19      * a tooth position x
20      *
21      * @param x          position of the saw from 0.0 to
22      *                  ...
23      * @param toothHeight should be greater zero
24      * @param toothWidth  should be greater zero
25      * @return           height value at the position x. If
26      *                  impossible, it returns "not a number" (<code>NaN</code>
27      *                  ).
28      */
29     public static double saw(
30         double x,
31         double toothHeight,
32         double toothWidth)
33     {
34         /*
35          * implements R01, R02.1, R03.1, R05
36          */
37         try
38         {
39             if (!MathValue.assertGreaterZero(toothHeight) ||
40                 !MathValue.assertGreaterZero(toothWidth))
41             {
42                 return Double.NaN;
43             }
44
45             if (x <= 0.0)
46             {
47                 return 0.0;

```

```

48         } else if (x <= toothWidth / 2)
49         {
50             return (toothHeight / (toothWidth / 2)) * x;
51         } else if (x <= toothWidth)
52         {
53             return (-toothHeight / (toothWidth / 2)) * x +
54                 2 * toothHeight;
55         } else if (toothWidth < x)
56         {
57             return saw(x - toothWidth, toothHeight, toothWidth);
58         } else
59         {
60             return Double.NaN;
61         }
62     } catch (java.lang.StackOverflowError eS)
63     {
64         System.out.println(eS + "\u00x:\u00" + x + "\n");
65         return Double.NaN;
66     }
67 }
68
69 /**
70  * <code>saw</code> computes the value of the height at
71  * a tooth position x
72  *
73  * @param x position of the saw from 0.0 to ...
74  * @return height value at the position x. If
75  *         impossible, it returns "not a number" (<code>NaN</code>
76  *         ).
77  */
78 public static double saw(double x)
79 {
80     /*
81      * implents R02.2, R03.2
82      */
83     return saw(
84         x,
85         MathValue.defaultToothHeight,
86         MathValue.defaultToothWidth);
87 }
88
89 /**
90  * <code>saw</code> computes the value of the height at
91  * a tooth position x
92  *
93  * @param x position of the saw from 0.0 to
94  *         ...
95  * @param toothHeight should be greater zero
96  * @return height value at the position x. If

```

```

    *     impossible , it returns "not a number" (<code>NaN</code>
98     *     ).
    */
100  public static double saw(double x, double toothHeight)
    {
102     /*
        *     implements R03.2
104     */
        return saw(
106         x,
            toothHeight ,
108         MathValue.defaultToothWidth);
    }
110 }

```

Listing 7.9: MathValue

```

/**
2  * <code>MathValue</code> class contains the default values
  * for the class <code>Math</code> and a method to validate
4  * the parameter values for <code>Math.saege(...)</code>
  *
6  * @since      06-Apr-2004, 30-May-2007
  * @author     Hinrich E. G. Bonin
8  * @version    1.1
  */
10 package de.leuphana.ics.function;

12 public abstract class MathValue
    {
14     final static double defaultToothHeight = 1.0;

16     final static double defaultToothWidth = 2 * java.lang.Math.PI;

18     /**
20     * <code>assertGreaterZero</code> checks the value of
  * its parameter. by using the <code>assert</code>
22     * construct
  *
24     * @param value will be checked if it is <code>>= 0.0</code>
  * @return <code>>true</code> or <code>>false</code>
26     */

28     protected static boolean assertGreaterZero(double value)
    {
30         /*
            *     implements R04
32         */
    }

```



```

34     try
35     {
36         assert value >= 0.0;
37     } catch (java.lang.AssertionError eA)
38     {
39         System.err.println(eA + "\uvalue:\u" + value);
40     }
41     return false;
42 }
43
44 return (value >= 0.0 ? true : false);
45 }
46 }
47 }
48 }

```

Listing 7.10: MathProg

```

/**
2  * <code>MathProg</code> class contains test cases for
3  * <code>Math.saw(...)</code>
4  *
5  * @since    06-Apr-2004, 30-May-2007
6  * @author   Hinrich E. G. Bonin
7  * @version  1.1
8  */
9
10 package de.leuphana.ics.function;
11
12 public class MathProg
13 {
14     /**
15     * @param args are not used
16     */
17
18     public static void main(String[] args)
19     {
20         // values are OK!
21         System.out.println (
22             Math.saw(4.5) + "\n" +
23             Math.saw(4.5, 2.0, 2.0));
24
25         // negative toothHeight and toothWidth
26         System.out.println (Math.saw(4.5, -2.0) + "\n" +
27             Math.saw(4.5, 2.0, -3.0));
28
29         // x is to great --> stack overflow
30         System.out.println (Math.saw(100000.0));
31     }
32 }

```

```
}
```

Protokolldatei MathProg.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/function/F.java

D:\bonin\anwd\code>java de.leuphana.ics.function.F
0.567605512172942
1.0

d:\bonin\anwd\code>javac -source 1.4
  de/leuphana/ics/function/MathProg.java

d:\bonin\anwd\code>java -enableassertions
  de.leuphana.ics.function.MathProg
0.567605512172942
1.0
java.lang.AssertionError value: -2.0
java.lang.AssertionError value: -3.0
NaN
NaN
java.lang.StackOverflowError x: 81452.03697319317
java.lang.StackOverflowError x: 81458.32015850036
java.lang.StackOverflowError x: 81464.60334380754
java.lang.StackOverflowError x: 81470.88652911472

NaN

d:\bonin\anwd\code>java -disableassertions
  de.leuphana.ics.function.MathProg
0.567605512172942
1.0
NaN
NaN
java.lang.StackOverflowError x: 72492.21472514895

NaN

d:\bonin\anwd\code>javadoc -version de/leuphana/ics/function/*.java
javadoc -version de/leuphana/ics/function/*.java
Loading source file de/leuphana/ics/function/F.java...
Loading source file de/leuphana/ics/function/Math.java...
Loading source file de/leuphana/ics/function/MathProg.java...
Loading source file de/leuphana/ics/function/MathValue.java...
Constructing Javadoc information...
Standard Doclet version 1.5.0_08
Building tree for all the packages and classes...
Generating de/leuphana/ics/function/\F.html...
```

```

Generating de/leuphana/ics/function/\Math.html...
Generating de/leuphana/ics/function/\MathProg.html...
Generating de/leuphana/ics/function/\MathValue.html...
Generating de/leuphana/ics/function/\package-frame.html...
Generating de/leuphana/ics/function/\package-summary.html...
Generating de/leuphana/ics/function/\package-tree.html...
Generating constant-values.html...
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 help-doc.html...
Generating stylesheet.css...

d:\bonin\anwd\code>

```

Begrenzung der Reichweite

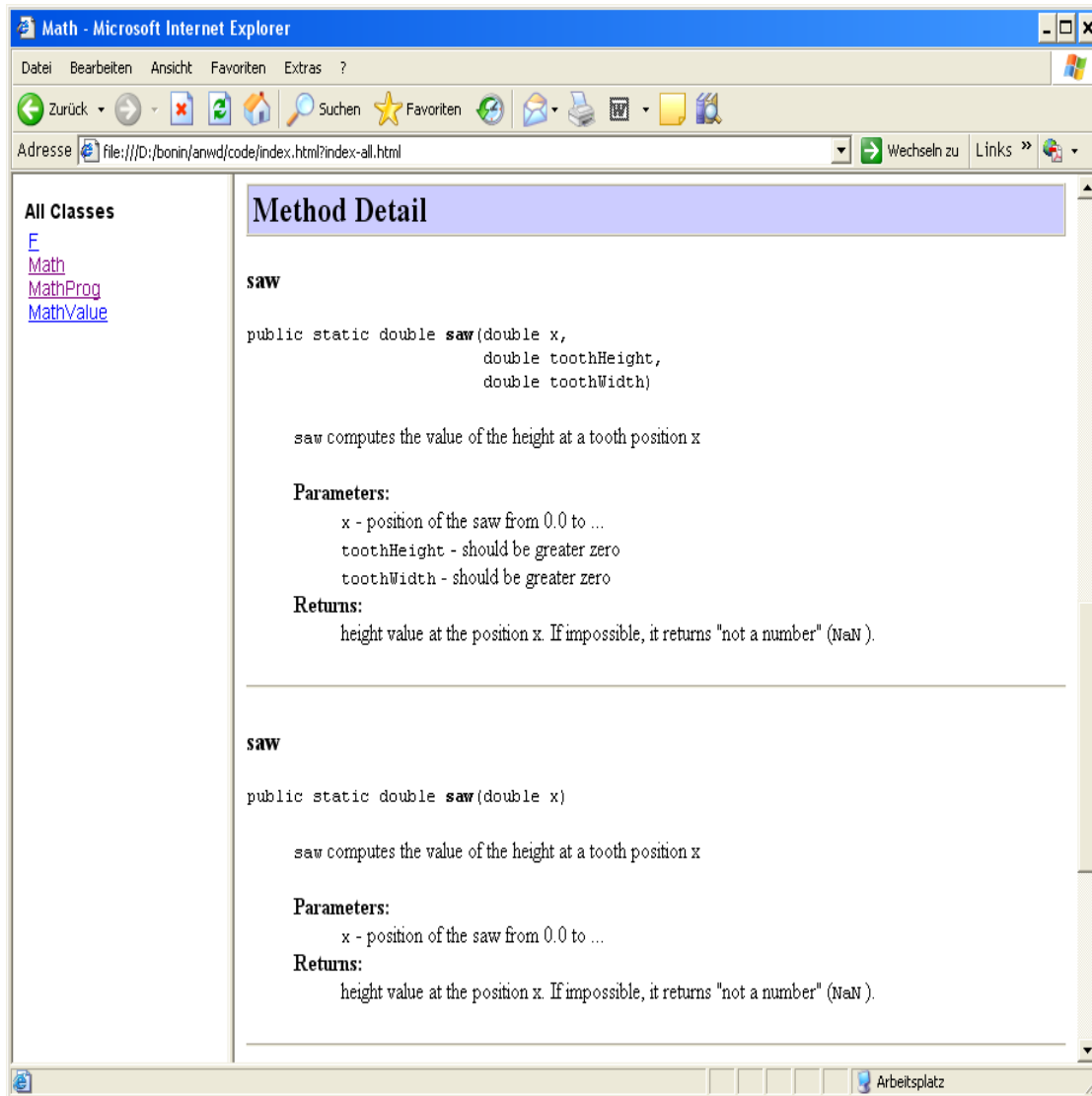
Stets sollte man sich präzise die Reichweite seiner Konstrukte überlegen. Die Reichweite ist möglichst auf das zwingend notwendige Maß zu begrenzen. Dazu eignen sich zusätzliche Blöcke, also {...}-Konstrukte, sowie das *Inner-Class*-Konzept (\leftrightarrow Abschnitt 6.4 S. 195 und z. B. Übungsaufgabe A.19 S. 452). 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()`.

Listing 7.11: Reichweite

```

/**
2  * Beispiel: Reichweiten begrenzen
3  * Optionen: Block und Innere Klasse
4  *
5  *
6  * @since      18-Mar-2004, 27-May-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    1.1
9  */
10
11 package de.leuphana.ics.scope;
12
13 public class Reichweite
14 {
15     private void internalWork(boolean working)
16     {
17         if (working)
18         {

```



Legende:

Dargestellt mit *Microsoft Internet Explorer*, Version 6.0.2900, auf einer Windows-XP-Plattform.

Abbildung 7.5: Dokumentation mittels javadoc

```

class Work
20 {
    private String id;
22
    Work(String s)
24 {
    id = s;
26 }
28
    String getId ()
30 {
    return id;
32 }
34
    void setId (String s)
36 {
    id = s;
38 }
40 }
Work w = new Work ("Emma");
42 w.setId (w.getId () + "_" + "Mustermann");
System.out.println (w.getId ());
44 } else
{
46     class Work
    {
48         String id = "Nothing_to_do!";
50
52         String text ()
    {
54             return "No_Working:_";
    }
56     }
System.out.println ((new Work()).id);
58 System.out.println ((new Work()).text () + "\n" +
    "_No_Spezifikation_Work_&_no_Objekt_w");
60 /*
    * Compiler error if something like
    * Work foo = new Work("Nemo");
62 */
    }
64 }
66
public static void main (String [] args)

```

```

68  {
70    {
72      /*
74       * Block 1 --- the scope of object r1
76       */
78      Reichweite r1 = new Reichweite ();
80      r1.internalWork(true);
82    }
84    {
86      /*
88       * Block 2 --- the scope of object r2
90       */
92      Reichweite r2 = new Reichweite ();
94      r2.internalWork(false);
96    }
98    {
100     /*
102     * Block 3 --- no r1 & r2 reachable
104     * For Example:
106     * Compile error for the following statement
108     * r2.internalWork(false);
110     */
112    }
114  }
116 }

```

Protokolldatei Reichweite.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/scope/Reichweite.java

D:\bonin\anwd\code>java
  de.leuphana.ics.scope.Reichweite
Emma Mustermann
Nothing to do!
No Working:
  No Spezifikation Work & no Objekt w

D:\bonin\anwd\code>cd de/leuphana/ics/scope

```

```
D:\bonin\anwd\code\de\lephana\ics\scope>dir Reichweite*.*
    691 Reichweite$1Work.class
    625 Reichweite$2Work.class
   1.249 Reichweite.class
   1.936 Reichweite.java

D:\bonin\anwd\code\de\lephana\ics\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 Modifizierern `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 Zwischenresultate bei mehreren Manipulationen sind dann alle 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 (\leftrightarrow [IBM-Francisco98]):

```
String myString = "Alles klar?";
```

7.3.3 Rahmen für Geschäftsobjekte und -prozesse

„Es gibt wenige Frameworks,
die wirklich funktionieren,
... die Zahl der unvollendeten,
vorzeitig verschrotteten Frameworks ist Legion.“
(\leftrightarrow [Broy/Siedersleben02] S. 58)

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:

**Fran-
cisco****Objekte**

- Geschäftsobjekte (*Common Business Objects*):
 - Adresse
 - Geschäftspartner (Kunde, Lieferant)
 - Kalender (zum Beispiel perioden-basiert)
 - Identifizierungs-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*

¹⁰Aktuelle Informationen zum San Francisco Projekt der IBM Corporation:
<http://www.ibm.com/java/sanfrancisco> (Zugriff:08-Jul-1998)

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 SDKs 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).

Java-Coach

Kapitel 8

Dokumentieren mit HTML

HTML (*HyperTextMarkup Language*) ist das Esperanto im Web. Hervorragend ist die Ausprägung XHTML (*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 den CSS-Möglichkeiten (*Cascading Style Sheets*) können die vielen Dokumente aus den verschiedenen Quellen (Entwicklungswerkzeugen) einheitlichen und damit leichter überschaubar präsentiert werden.

Trainingsplan

Das Kapitel „Dokumentation mit HTML“ erläutert:

- Möglichkeiten von XHTML und
↔ Seite 382 ...
 - die Layout-Gestaltung mit Hilfe von *Cascading Style Sheets* (CSS).
↔ Seite 384 ...
-

8.1 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 8.2 S. 384)
- 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 SGML¹-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 \leftrightarrow [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.

Listing 8.1: `index.php`

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3 <!-- Bonin 09-Feb-2000 ... 02-May-2007 -->
4 <!-- Hinweis: Kein xml-Prolog weil PHP Fehler meldet -->
5 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
6 <head>
7 <link rel="shortcut icon" href="/as111.ico" />
8 <meta http-equiv="Content-Type"
9   content="text/html; charset=iso-8859-1" />
10 <meta name="robots" content="index, follow" />
11 <meta http-equiv="Last-Modified"
12   content="02-May-2007 13:00:00 GMT" />
13 <meta http-equiv="Expires"
14   content="31-Dec-2010 00:00:00 GMT" />
15 <meta name="DESCRIPTION" content="Bonin's Homepage" />
16 <meta name="KEYWORDS"
17   content="Wirtschaftsinformatik ,
18   Web-Technologie , Aspect-Oriented Programming ,
19   Aspect-Oriented Softwaredevelopment , Verwaltungsinformatik" />
20 <meta http-equiv="Content-Script-Type" content="text/javascript" />
21 <link href="myStyle.css" rel="stylesheet" type="text/css" />
22 <link rev="owns" title="Hinrich E.G. Bonin"
23   href="mailto:h.bonin@uni-lueneburg.de" />
24 <title>as: applied sciences / aspect-oriented software</title>
25 </head>
26 <body>
27 <h1><a href="http://www.uni-lueneburg.de">
28    </a></h1>

```

¹SGML \equiv *Standard Generalized Markup Language*, ISO-Standard 8879:1986

```

32 <h1><a href="content.html">Bonin's Main Web Server </a></h1>
33 <h3><a href="http://nemo.uni-lueneburg.de">
    http://nemo.uni-lueneburg.de</a></h3>
34 <h3><a href="/politik/global.html">
    Politik -- Globalisierung -- äMachtkmpfe </a></h3>
36 <h3><a href="content.html">
    Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch-Ing.
38 Hinrich E. G. Bonin </a></h3>
<h3>email: <a href="mailto:h.bonin@uni-lueneburg.de">
40 bonin@uni-lueneburg.de</a></h3>
<h3>(N 5315.008 E 01023.000) </h3>
42 <h3>(until 31-Dec-2004:
    <a href="http://www.uni-lueneburg.de">
44 Fachhochschule Nordostniedersachsen </a> in
    <a href="http://www.lueneburg.de/index.html">ü
46 Lneburg </a></h3>
<hr />
48 <h3><a href="content.html">Wednesday 13th 2007 f
    June 2007 17:26:51 </a></h3>
50 <h3>Mitwirkung am <a href="http://dog.uni-lueneburg.de/">
    Manuskript Jagdhund und Jagdgebrauchshund </a></h3>
52 <h2>
<a href="http://ics.uni-lueneburg.de"> </a>
56 <a href="http://validator.w3.org/check/referer"> </a>
<a href="lebenslauf.html"> </a>
62 <a href="http://www.anybrowser.org/campaign/"> </a>
<a href="http://www.lsb-niedersachsen.de/"> </a>
</h2>
70 </body>
<!-- Ende der Datei /u/bonin/mywww/index.php -->
72 </html>

```

8.2 Cascading Style Sheets (CSS)

Cascading Style Sheet ist ein zweigestufter Mechanismus (CSS1 \equiv Level 1; CSS2 \equiv Level 2), der jeweils sowohl dem Autor wie dem Leser

ermöglicht Farben, Schriftart, Schriftgröße und Zwischenräume (\approx das gewünschte Layout) mit dem Dokument zu verknüpfen. CSS1 ist eine leicht lesbare und schreibbare Spezifikationsprache, die sich an der üblichen Deskop Publishing Terminology orientiert. **DTP**

8.2.1 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.

8.2.2 HTML-Dokument ⇔ CSS

Das HTML-Dokument kann von dem *Style Sheet* auf verschiedene Weise verknüpft werden. Das folgende Beispiel zeigt vier Möglichkeiten:

1. im <head>-Konstrukt

<link>

- (a) mit dem <link>-Konstrukt wird ein Verweis auf eine externe CSS-Datei angegeben und diese wird über den Web-Server geladen

<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.uni-lueneburg.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>
```

8.2.3 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 {
  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 vererbt. Es empfiehlt sich daher bei einer Angabe von `color` stets auch eine Angabe für `background` zu machen.

```
body {  
  background: url(http://as.uni-lueneburg.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%;  
}
```

8.2.4 Selektor: `class` & `id`

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;  
}
```

`class`

```

...
<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 `id` eine CSS-Spezifikation zugeordnet. Der `id`-Wert muß im Dokument eindeutig sein. Er wird beginnend mit einem Hashzeichen „#“ notiert, wie das folgende Beispiel zeigt.

```

#ZZ981 { letter-spacing: 0.3em }
#ZZ982 { letter-spacing: 0.5em }
...
<p id="ZZ982">Buchstaben mit viel Zwischenraum</p>

```

8.2.5 Kontextabhängige Selektoren

Im folgenden CSS-Beispiel sind alle ``-Konstrukte im Dokument von der Spezifikation (grüne Textfarbe) betroffen:

```

h1 {
  color: red;
}
em {
  color: green;
}

```

Soll sich die Spezifikation nur auf ``-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;
}
```

8.2.6 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.uni-lueneburg.de/gelberBall.gif) white;
  list-style-position: inside;
}
```

8.2.7 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 angegeben, 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>
...
```

8.2.8 Die Cascade & 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.uni-lueneburg.de/mainStlye.css);
@import url(http://as.uni-lueneburg.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 (\leftrightarrow [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-Angaben 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 {...}           /*  $\alpha = 0 \beta = 0 \gamma = 1 \rightarrow$  Gewicht= 1 /*
ul li {...}       /*  $\alpha = 0 \beta = 0 \gamma = 2 \rightarrow$  Gewicht= 2 /*
ul ol li {...}    /*  $\alpha = 0 \beta = 0 \gamma = 3 \rightarrow$  Gewicht= 3 /*
li.red {...}      /*  $\alpha = 0 \beta = 1 \gamma = 1 \rightarrow$  Gewicht= 11 /*
ul ol li.red {...} /*  $\alpha = 0 \beta = 1 \gamma = 3 \rightarrow$  Gewicht= 13 /*
#x123 {...}       /*  $\alpha = 1 \beta = 0 \gamma = 0 \rightarrow$  Gewicht= 100 /*
```

Dabei zählen die *Pseudo*-Selektoren wie zum Beispiel `a:link` als normale Elemente.

- Bei Angaben mit gleichem Gewicht wird die zuletzt spezifizierte gewählt.

8.2.9 CSS-Beispiel

Das HTML-Dokument `exampleCSS.html` enthält in seinem `<link>`-Konstrukt einen Verweis auf die CSS-Datei `myStyle.css`. Die Abbildung 8.1 S. 394 zeigt die Darstellung eines Auszugs des Dokumentes mit dem Browser *Microsoft Internet Explorer*, Version 6.0.2600, auf einer Windows-XP-Plattform.

Listing 8.2: `myStyle.css`

```
/* Basis-Layout fuer das Projekt: FOO */
2 /*   Hinrich E. G. Bonin 23-06-1998   */
/*   Update 13-Jun-2007                 */
4 ul {
    color: black;
6    background-color: white;
    list-style-image:
8     url(http://as.uni-lueneburg.de/gelberBall.gif);
    list-style-position: inside;
10 }
    ol {
12    color: white;
        background-color: black;
14    list-style-type: lower-roman;
    }
16
18 p:first-letter {
    font-size: 24pt;
    float: left;
```



Legende:

Dargestellt mit *Microsoft Internet Explorer*, Version 6.0.2900, auf einer Windows-XP-Plattform.

Quellcode `exampleCss.html` ↔ S. 396, Quellcode `myStyle.css` ↔ S. 393

Abbildung 8.1: XHTML: `exampleCSS.html` & CSS: `myStyle.css`

```
20   color: black;
    background-color: yellow;
22 }
    p {
24   font-family: "Times New Roman", Times, serif;
    font-weight: bold;
26   font-size: 14pt;
    line-height: 150%;
28   letter-spacing: 0.5em;
    color: green;
30   background-color: yellow;
    }
32   h3 {
    font-family: Arial, Helvetica, sans-serif;
34   font-size: 20pt;
    font-weight: bold;
36   text-decoration: underline;
    color: green;
38   background-color: white;
    margin: 3em;
40 }
    h1, h2 {
42   font-size: 24pt;
    font-weight: bold;
44   font-family: Arial, Helvetica;
    color: yellow;
46   background-color: blue ! important;
    margin: 5px;
48 }
    body {
50   background:
        url(http://as.uni-lueneburg.de/gelberBall.gif) black;
52   color: white;
    }
54   a:link {
    font-weight: bold;
56   text-decoration: none;
    color: red;
58   background-color: black;
    }
60   a:visited {
    font-weight: bold;
62   text-decoration: none;
    color: blue;
64   background-color: black;
    }
66   a:active {
    font-weight: bold;
68   text-decoration: none;
```

```

    color:          yellow;
70  background-color: black;
    }
72  em {
    font-style: italic
74  }
    .hinweis {
76  font-style: italic;
    font-weight: bold;
78  color:          yellow;
    background-color: black;
80  }
    .hervorhebung {
82  color:          white;
    background-color: maroon;
84  margin:         10px;
    border:         none;
86  }
    .dickeListe {
88  color:          black;
    background-color: white;
90  list-style-type: square;
    font-family:   "Times New Roman", Times, serif;
92  font-size:     18pt;
    font-weight:   bold;
94  }
    .anhang {
96  color:          white;
    background-color: black;
98  font-style:    italic;
    font-family:   Helvetica, sans-serif;
100 font-size:     12pt;
    text-indent:   25px;
102 }

```

Listing 8.3: exampleCSS.html

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- CSS-Beispiel -->
4 <!-- Hinrich E. G. Bonin 25-Jun-1998 -->
<!-- Update ... 13-Jun-2007 -->
6 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
8 <link href="http://as.uni-lueneburg.de/xml/myStyle.css"
    rel="stylesheet" type="text/css" />
10 <title>CTP-Dokumentation</title>
</head>
12 <body>

```

```

14 <h1><em>CTP</em>–Dokumentation </h1>
15 <p class="hinweis">
16   Graphische Darstellung der
17   <a href="irgenwohin.html">Fachklassen </a>
18 </p>
19 <h3>Vorgehensweise </h3>
20 <p>Es sind folgende Schritte zu vollziehen:</p>
21 <ul>
22   <li>Bestimme die Jahrestrainingsstunden </li>
23   <li>Bestimme Hauptwettkampfkalenderwoche und Jahr </li>
24   <li>üPrfe den Hauptwettkampftermin </li>
25   <li>Berechne die Handlungsspanne </li>
26   <li>Berechne die Wartezeit </li>
27   <li>Berechne üfrhesten Beginntermin des Trainingsplanes </li>
28   <li>Berechne äsptesten Beginntermin des Trainingsplanes </li>
29   <li>Lege den Trainingsbeginn fest </li>
30 </ul>
31 <p>Danach kann ein kompletter Trainingsplan
32   erstellt werden, dessen Hauptwettkampf erreichbar ist.
33 <div class="hervorhebung">
34   Dazu dienen folgende Methoden:
35   <ul class="dickeListe">
36     <li>getFruehestesTrainingsPlanBeginnJahr () </li>
37     <li>getFruehesteTrainingsPlanBeginnWoche () </li>
38     <li>getSpaetestesTrainingsPlanBeginnJahr () </li>
39     <li>getSpaetesteTrainingsPlanBeginnWoche () </li>
40   </ul>
41 <p>Diese Methoden sind mehr als die ü blichen
42   "get–Methoden". Sie berechnen äabhängig von der
43   Handlungsspanne und der aktuellen Woche,
44   den üfrhesten und äsptesten Trainingsplanbeginn.
45   Dabei wird der Jahresumbruch übercksichtigt, das hei&szlig;t,
46   der Hauptwettkampf liegt im äächsten Kalenderjahr.</p>
47 </div>
48 <p class="anhang">
49   <a href="/copyright.html">Copyright </a>
50   <a href="mailto:bonin@uni-lueneburg.de">Bonin </a>
51   Apr–1995, ..., Jun–2007 all rights reserved
52 </p>
53 </body>
54 </html>

```

Java-Coach

Kapitel 9

Java™ — 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 (\leftrightarrow [Dahl+67]) oder CLOS (\leftrightarrow [Gabriel91]). Heute ist Objektorientierung (OO) ein Paradigma der gesamten Softwareentwicklung, das das ganze Spektrum von der Spezifikation, der Konstruktion bis zur Implementierung inklusive Betrieb und Wartung unterstützt. Java™ ist unstrittig im Bereich Internet- und Client/Server-Systeme die zweckmäßige OO-Sprache. Allerdings erfüllt Java™ nur bedingt das OO-Paradigma strikt und vermeidet natürlich auch nicht Mängel dieses Paradigmas. Plakativ formuliert: Java™ ist bewährte „OO-Hausmannskost“ und nicht auf dem heutigen Stand der wissenschaftlich verstandenen Programmiermethodik und Softwaretechnik.

Trainingsplan

Das Kapitel „Java™ — OO-Anspruch und OO-Wirklichkeit“ erläutert:

- das OO-Paradigma, also OO-Grundlagen, OO-Prinzipien, OO-Konzepte im Hinblick auf ihre konkrete Umsetzung und \leftrightarrow Seite 400 ...

- die strikte Objekt-Orientierung im Java™ -Kontext.
↪ Seite 401 ...
-

9.1 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 geeigneten 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 (\leftrightarrow [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 JavaTM 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 JavaTM-OO unterstützt uns nicht bei der notwendigen W -Ermittlung.

9.2 Strikte Objekt-Orientierung

“Although it is based on C++,
Java is more of a ‘pure’ object-oriented language.”
(\leftrightarrow [Eckel02] p. 77)

In JavaTM entspricht die Basis nicht dem Konzept einer strikten Objekt-Orientierung weil die sogenannten primitiven Typen keine Instanzen einer Klasse sind (*PrimitiveType* \leftrightarrow Tabelle 5.5 S. 146). 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

Strikte Objekt-Orientierung		
Ausdruck	OO-Code	OO-Feature
-1	1.negate()	<i>Method invocation</i> Das Objekt 1 appliziert die Integer-Negationsmethode und eine Referenz zum neuen Integer-Objekt -1 wird zurückgegeben.
1 + 2	1.add(2)	<i>Method invocation;</i> <i>Collaboration</i>
Integers: a, b $a + b$	a.add(b)	<i>Method invocation;</i> <i>Collaboration</i>
Integers: a, b, c $a + b * c$	a.add(b.mult(c))	<i>Method invocation;</i> <i>Collaboration;</i> <i>Cascaded method call</i>
Characters: a, b $a < b$	a.lt(b)	<i>Method invocation;</i> <i>Collaboration;</i> <i>Composition</i> (Eine Referenz zum Boolean-Objekt true oder false wird zurückgegeben.)
Booleans: a, b $a \wedge b$	a.and(b)	<i>Method invocation;</i> <i>Collaboration</i>

Legende:

Tabellenidee \leftrightarrow [Ourosuff02] p. 106.

In Java™ 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 9.1: Einfache Ausdrücke : OO-Code

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 9.1 S. 402 verdeutlicht diesen Java™ -OO-Mangel.

Kapitel 10

Java™ N Plattform: Hoffnungen & Visionen

```
    ' '
    () ()
    () ()
    ( o o )
    ^ ( @ _ ) ^
    \\ ( ) //
    \\ ( ) //
    ( )
    ( )
    ( )
    _//~~\\_
    ( ) ( )
```

Lang
lebe
Java!

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™ weiterentwickeln? Was sind unsere Hoffnungen und Visionen? Was kennzeichnet die Plattform $N > 2$? Gibt es im Jahr 2010 noch ein zeitgemäßes Java™ ?

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™ –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 bewegendem Programmen, die sich durch das biologische Vererbungskonzept weiterentwickeln. Der Virus von heute mutiert zum Arbeitspferd von morgen.

Die ganz am Anfang stehende Aussage *Programmieren bleibt schwierig!* wird bestimmt auch in Zukunft gültig bleiben. Wenn sich Java™ kontinuierlich weiterentwickelt, dann gibt es sicherlich andere Kernfragen bei den Schwierigkeiten. Es besteht aber die berechtigte Hoffnung, dass Ihre Kenntnisse, die Sie beim Durcharbeiten gewonnen haben, für Sie nützlich bleiben. Beim Nutzen dieser Kenntnisse wünsche ich Ihnen abschließend viel Erfolg.

**Program-
mieren
bleibt
schwie-
rig!**

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 ≈ 10000 Jagd- und Sportwaffen um. Es werden 11 Zweigstellen beliefert. Die umsatzstärkste Zweigstelle ist in Mannheim. Sie verkauft ≈ 1200 Waffen, die umsatzschwächste ist in Lüneburg und verkauft ≈ 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“ beschreiben.
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 registriert.
11. Es werden derzeit folgende Gewehrtypen verkauft:
 - (a) Querflinte \equiv zwei nebeneinanderliegende Flintenläufe
 - (b) Bockflinte \equiv zwei übereinanderliegende Flintenläufe
 - (c) Bockbüchse \equiv zwei übereinanderliegende Büchsenläufe
 - (d) Bockbüchseflinte \equiv ein Flintenlauf liegt über einem Büchsenlauf
 - (e) Drilling \equiv 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 Lichtstärke 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 möchten 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 Waffenbesitzkarte 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 registriert.

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.

Listing A.1: Wert

```

/**
 2  * Beispiel: Gleichheit in Java
 3  *
 4  * @since      25-May-2007
 5  * @author     Hinrich E. G. Bonin
 6  * @version    1.1
 7  */
 8
 9  package de.leuphana.ics.gleichheit;
10
11  public class Wert
12  {
13      public static void main(String[] args)
14      {
15          String wert = "Software";
16
17          String part1 = "Soft";
18          String part2 = "ware";
19
20          String s1 = new String(wert);
21          String s2 = new String(wert);
22
23          System.out.println (
24              "1:␣" + ("Software" == s1) + "\n" +
25              "2:␣" + (s1 == s2) + "\n" +
26              "3:␣" + ("Software" == "Soft" + "ware") + "\n" +
27              "4:␣" + (wert == "Soft" + "ware") + "\n" +
28              "5:␣" + (wert == "Soft" + part2) + "\n" +
29              "6:␣" + (wert == part1 + part2) + "\n" +
30              "7:␣" + (wert.equals(part1 + part2)) + "\n");
31      }
32  }

```

Geben Sie bei dem folgenden Aufruf das Ergebnis an:

```
>java de.leuphana.ics.gleichheit.Wert
```

A.5 Applikation Scoping

Die Klasse Scoping skizziert eine Möglichkeit zur Begrenzung der Reichweite mittels zusätzlicher Blockstrukturierung (\leftrightarrow Abschnitt 7.3.2 S. 371).

Listing A.2: Scoping

```

/**
 2  * Beispiel: Scoping in Java
 3  *
 4  * @since      18-Mar-2004, 25-May-2007
 5  * @author     Hinrich E. G. Bonin
 6  * @version    1.1
 7  */
 8
 9  package de.leuphana.ics.scope;
10
11  import java.util.Date;
12
13  public class Scoping
14  {
15      public static void main(String [] args)
16      {
17          Date i = new Date ();
18          {
19              Date j = new Date ();
20              System.out.println ("j:␣" + j);
21          }
22          System.out.println ("i:␣" + i);
23
24          System.out.println ("j:␣" + j);
25      }
26 }

```

Geben Sie bei dem folgenden Aufruf das Ergebnis an:

```
>javac de/leuphana/ics/scope/Scoping.java
```

A.6 Applikation Controlling

Listing A.3: Kontrolle

```

/**
 2  * Kleine Kostprobe fuer: data types and control structures
 3  *

```

```

4  * @since      02-Apr-1998, 07-Jan-2003, 27-May-2007
   * @author     Hinrich E. G. Bonin
6  * @version    2.0
   */
8
9  package de.leuphana.ics.control;
10
11 public class Controlling
12 {
13     public static void main(String [] args)
14     {
15         int i = args.length;
16         int j, k;
17         double m = 0.314159265358979e1;
18         int n = (int) m;
19
20         String p = "Java";
21
22         boolean important;
23         boolean maybe = true;
24         boolean ofCourse;
25
26         i += p.length();
27         j = i++;
28         i--;
29         k = ++i;
30         --i;
31
32         important = (i == j && maybe);
33         important = (important != maybe);
34         ofCourse = (i <= k) || (important == true);
35
36         System.out.println(
37             "i=" + i + "\n" +
38             "j=" + j + "\n" +
39             "k=" + k + "\n" +
40             "m=" + m + "\n" +
41             "n=" + n + "\n" +
42             "p=" + p + "\n" +
43             "maybe=" + !maybe + "\n" +
44             "important=" + !important + "\n" +
45             "ofCourse=" + ofCourse);
46     }
47 }

```

Geben Sie bei dem folgenden Aufruf das Ergebnis an:

```
>java de.leuphana.ics.control.Controlling is OK!
```

A.7 Applikation Iteration

Listing A.4: Iteration

```
/**
 2  * Kleine Kostprobe fuer: Iterationen
 3  *
 4  * @since      02-Apr-1998, 26-Nov-2002, 27-May-2007
 5  * @author     Hinrich E. G. Bonin
 6  * @version    1.2
 7  */
 8  package de.leuphana.ics.loop;
 9
10  public class Iteration
11  {
12      public static void main(String [] args)
13      {
14          boolean in = true;
15          int zaehler = 0;
16          int index;
17          String spruchTabelle []
18              = new String [args.length];
19
20          String meinSpruch = "";
21          String wortZumSuchen = "C++";
22          String wortZumErsetzen = "Java";
23
24          spruchTabelle [0] = "Maximum";
25          spruchTabelle [1] = "UML";
26          spruchTabelle [2] = "&";
27          spruchTabelle [3] = "C++";
28          spruchTabelle [4] = "in_nder";
29          spruchTabelle [5] = "Anwendungsentwicklung";
30
31          int anzahlPositionen = spruchTabelle.length;
32
33          while (zaehler < anzahlPositionen)
34          {
35              if (spruchTabelle [zaehler].equals(
36                  wortZumSuchen)
37              {
38                  spruchTabelle [zaehler] =
39                      wortZumErsetzen;
40                  break;
41              }
42          }
43      }
44  }
```

```

42     zaehler++;
    }
44     zaehler = -1;
46     do
    {
48         zaehler++;
        meinSpruch += spruchTabelle[zaehler];
50         meinSpruch += " ";
    } while (zaehler < (anzahlPositionen - 1));
52     System.out.println (meinSpruch +
54         "\nDies sind " + meinSpruch.length() +
        " Zeichen!");
56 }

```

Geben Sie bei den folgenden Aufrufen das Ergebnisse an:

```
>java de.leuphana.ics.loop.Iteration 1 2 3 4 5 6 7
```

```
>java de.leuphana.ics.loop.Iteration 1 2
```

A.8 Applikation LinieProg

Listing A.5: Linie

```

/**
2  * Beispiel zum Java-Training
    *
4  * @since    06-Apr-1998, 15-Jul-1998,
    *          26-Nov-2002, 25-May-2007
6  * @author   Hinrich E. G. Bonin
    * @version  2.2
8  */

10 package de.leuphana.ics.think;

12 public class Linie
    {
14     private int startX, startY, endX, endY;

16     public int getStartX()
18     {
        return startX;
    }

```

```
20     }
22
23     public int getStartY ()
24     {
25         return startY;
26     }
28
29     public int getEndX ()
30     {
31         return endX;
32     }
34
35     public int getEndY ()
36     {
37         return endY;
38     }
40
41     public Linie(int startX , int startY ,
42                int endX, int endY)
43     {
44         this.startX = startX;
45         this.startY = startY;
46         this.endX = endX;
47         this.endY = endY;
48     }
50
51     public void setStartpunkt(int startX , int startY)
52     {
53         this.startX = startX;
54         this.startY = startY;
55     }
56
57     public void setEndpunkt(int endX, int endY)
58     {
59         this.endX = endX;
60         this.endY = endY;
61     }
62
63
64     public double laengeLinie ()
65     {
66         return (Math.sqrt (
67             Math.pow((double) endX - startX , 2.0) +
```

```

70         Math.pow((double) endY - startY , 2.0));
71     }
72
73     public boolean groesserAls(double vorgabeLaenge)
74     {
75         return ( this.laengeLinie () > vorgabeLaenge );
76     }
77 }

```

Listing A.6: LinieProg

```

/**
2  * Beispiel zum Java-Training
3  *
4  * @since      06-Apr-1998, 15-Jul-1998,
5  *             26-Nov-2002, 25-May-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    2.2
8  */
9
10 package de.leuphana.ics.think;
11
12 public class LinieProg
13 {
14
15     public static void main(String [] args)
16     {
17         double vorgabeLaenge = 0.25000e2;
18
19         Linie l1 = new Linie(10, 10, 13, 14);
20         Linie l2 = l1;
21
22         l2.setStartpunkt(0, 0);
23         l2.setEndpunkt(3, 4);
24
25         System.out.println (
26             "" +
27             l1.getStartX () +
28             l1.getStartY () +
29             "\n" +
30             l1.getEndX () +
31             l1.getEndY () +
32             "\n" +
33             l1.laengeLinie ());
34
35         System.out.println (
36             l1.getStartX () +
37             l1.getStartY () +

```

```

38     l1.getEndX() +
39     l1.getEndY() +
40     l1.laengeLinie());
41
42     System.out.println(
43         l1.groesserAls(vorgabeLaenge));
44 }
}

```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.leuphana.ics.think.LinieProg
```

A.9 Applikation Inheritance

Die Abbildung A.1 S.417 zeigt das Klassendiagramm der Applikation Inheritance.

Listing A.7: Inheritance

```

/**
2  * Kleine Kostprobe fuer: Vererbung
3  * --- abstract class
4  *
5  *
6  * @since      05-Apr-1998, 26-Nov-2002, 27-May-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    2.0
9  */
10
11 package de.leuphana.ics.inheritance;
12
13 public class Inheritance extends Bar
14 {
15     int zwei()
16     {
17         return 2;
18     }
19
20     public static void main(String [] args)
21     {
22         Inheritance m = new Inheritance ();
23
24         Bar.setName(new StringBuffer ("Otto AG"));
25         Bar.getName().setCharAt(3, 'i');
26         System.out.println (Bar.getName ());

```

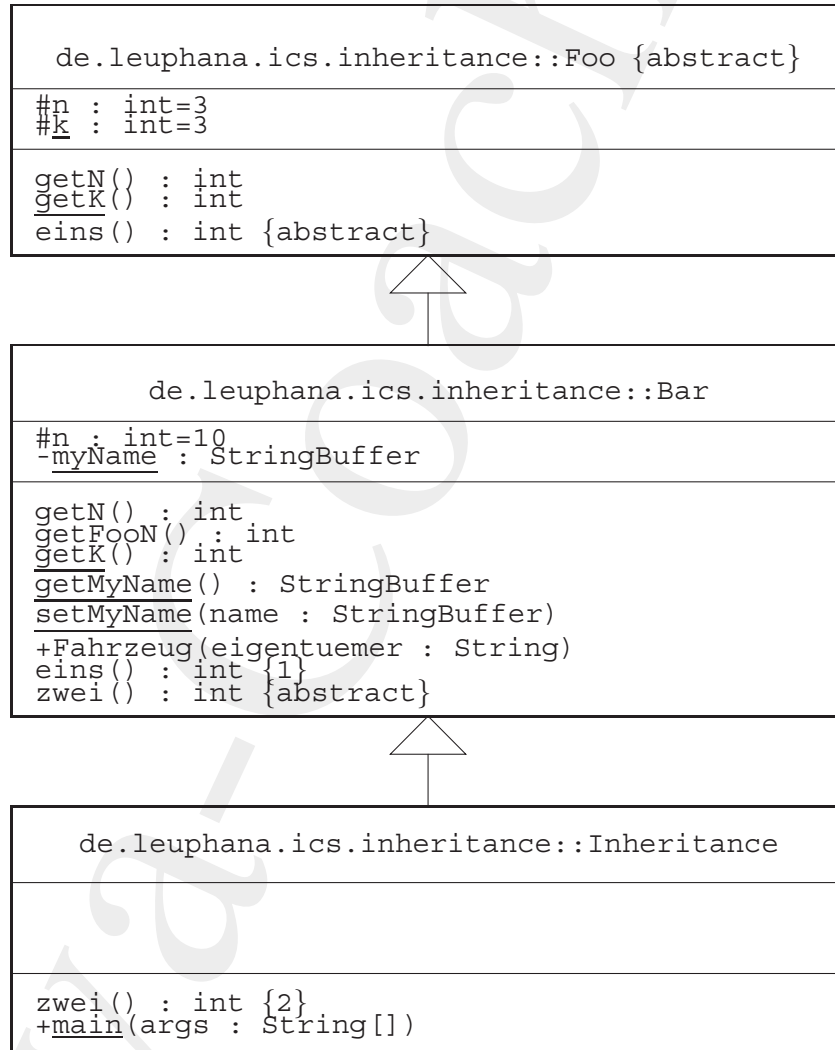



Abbildung A.1: Klassendiagramm für Inheritance

```

28     System.out.println (
29         "I:_" +
30         m.eins () +
31         m.zwei () +
32         m.getFooN () +
33         Foo.getK ());
34
35     System.out.println (
36         "II:_" +
37         m.eins () +
38         m.zwei () +
39         m.getN () +
40         Bar.getK ());
41 }
42 }

```

Listing A.8: Foo

```

/**
2  * Kleine Kostprobe fuer: Vererbung
3  * --- abstract class
4  *
5  *
6  * @since      05-Apr-1998, 26-Nov-2002, 27-May-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    2.0
9  */
10
11 package de.leuphana.ics.inheritance;
12
13 abstract class Foo
14 {
15     protected int n = 3;
16
17     protected static int k = 3;
18
19     abstract int eins ();
20
21     int getN ()
22     {
23         return n;
24     }
25
26     static int getK ()
27     {
28         return k;
29     }
30 }

```

Listing A.9: Bar

```
/**
2  * Kleine Kostprobe fuer: Vererbung
  * --- abstract class
4  *
  *
6  * @since    05-Apr-1998, 26-Nov-2002, 27-May-2007
  * @author   Hinrich E. G. Bonin
8  * @version  2.0
  */
10
package de.leuphana.ics.inheritance;
12
abstract class Bar extends Foo
14 {
16     protected int n = super.n + 7;
18     private static StringBuffer myName;
20     abstract int zwei();
22     int getN()
23     {
24         return n;
25     }
26     int getFooN()
27     {
28         return super.getN();
29     }
30
31     static int getK()
32     {
33         return 2 * k;
34     }
35
36     static void setMyName(StringBuffer name)
37     {
38         myName = name;
39     }
40
41     static StringBuffer getMyName()
42     {
43         return myName;
44     }
45
46     int eins()
```

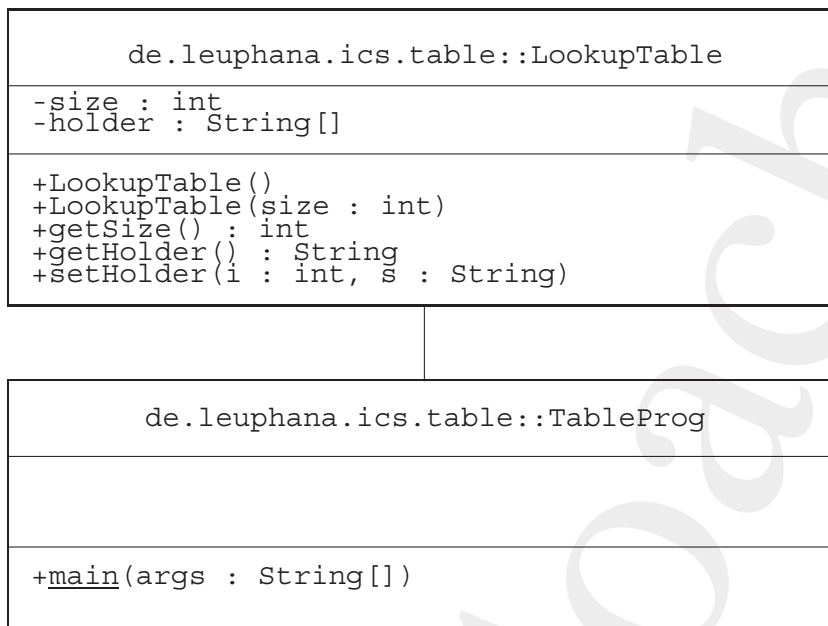


Abbildung A.2: Klassendiagramm für TableProg

```

48     {
50         return 1;
    }
}

```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.leuphana.ics.inheritance.Inheritance
```

A.10 Applikation TableProg

Die Abbildung A.2 S. 420 zeigt das Klassendiagramm der Applikation TableProg.

Listing A.10: TableProg

```

/**
2  * Kleine Kostprobe fuer den Fehler
   * "recursive constructor invocation"
4  *

```

```

6  * @since      13-Apr-1998, 26-Nov-2002, 27-May-2007
7  * @author     Hinrich E. G. Bonin
8  * @version    1.2
9  */
10 package de.leuphana.ics.table;
11
12 public class TableProg
13 {
14     public static void main(String args[])
15     {
16         LookupTable myTable = new LookupTable();
17
18         myTable.setHolder(
19             99, "Alles_richtig ,_oder_was?");
20
21         System.out.println (
22             "Tabelle_mit_" +
23             myTable.getSize() +
24             "_erzeugt!");
25
26         System.out.println (
27             myTable.getHolder(99));
28     }
29 }

```

Listing A.11: LookupTable

```

/**
2  * Kleine Kostprobe fuer den Fehler
3  * "recursive constructor invocation"
4  *
5  * @since      13-Apr-1998, 26-Nov-2002, 27-May-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.2
8  */
9
10 package de.leuphana.ics.table;
11
12 final class LookupTable
13 {
14     private int size;
15     private String holder[];
16
17     LookupTable()
18     {
19         this(100);
20     }

```

```

22     LookupTable(int size)
23     {
24         this();
25         this.size = size;
26         holder = new String[size];
27     }
28
29     public int getSize()
30     {
31         return size;
32     }
33
34     public String getHolder(int i)
35     {
36         return holder[i];
37     }
38
39     public void setHolder(int i, String s)
40     {
41         holder[i] = s;
42     }
43 }

```

Geben Sie bei dem folgenden Aufruf das Ergebnis an:

```
>javac de/leuphana/ics/table/TableProg.java
```

Falls Sie einen Fehler erkennen, korrigieren Sie diesen und geben Sie dann das Ergebnis von folgendem Aufruf an:

```
>java de.leuphana.ics.table.TableProg
```

A.11 Applikation Rekursion

Listing A.12: Rekursion

```

/**
2  * Kleine Kostprobe fuer eine Rekursion
3  * Beispiel Fakultaet:
4  *  $n! = n * (n - 1)!$ 
5  *
6  * @since 10-Apr-1998, 28-Dec-2002, 27-May-2007
7  * @author Hinrich E. G. Bonin
8  * @version 1.2

```

```

10  */
11  package de.leuphana.ics.rekursion;
12
13  public class Rekursion
14  {
15      public static void main(String [] args)
16      {
17
18          Fakultaet foo = new Fakultaet ();
19
20          String in;
21          if (args.length == 0)
22          {
23              in = "0";
24          } else
25          {
26              in = args[0].replace('+', '0');
27          };
28
29          long k = Long.parseLong(in);
30          long grenze = Long.MAX_VALUE;
31
32          if (k <= grenze && k >= 0)
33          {
34              System.out.println (
35                  "Fakultaetsfunktion :_fac(" +
36                  k +
37                  ")_=_ " +
38                  foo.fac(k));
39          } else
40          {
41              System.out.println (
42                  "Wert_=_ " +
43                  k +
44                  "_kann_nicht_berechnet_werden!");
45          }
46
47          System.out.println (
48              "Anzahl_der_Aufrufe_von_fac():_ " +
49              Fakultaet.anzahlAufrufeFac);
50      }
51  }

```

Listing A.13: Fakultaet

```

/**
2  * Kleine Kostprobe fuer eine Rekursion
3  * Beispiel Fakultaet:

```

```
4  *  $n! = n * (n - 1)!$ 
   *
6  * @since      10-Apr-1998, 28-Dec-2002, 27-May-2007
   * @author     Hinrich E. G. Bonin
8  * @version    1.2
   */
10
11 package de.leuphana.ics.rekursion;
12
13 import java.math.BigInteger;
14
15 class Fakultaet
16 {
17     /*
18     * long-Wertebereich: 64 Bit
19     * -9223372036854775808 ... 9223372036854775807
20     * BigInteger von beliebiger Groesse
21     */
22     BigInteger wert;
23     BigInteger basisWert = new BigInteger("1");
24
25     static long anzahlAufrufeFac = 0;
26
27     BigInteger fac(long n)
28     {
29         anzahlAufrufeFac += 1;
30
31         if (n == 0)
32         {
33             return basisWert;
34         } else
35         {
36             System.out.println(
37                 "Aufruf von fac mit n = " + n);
38
39             wert = this.fac(n - 1).multiply(
40                 new BigInteger(Long.toString(n)));
41
42             System.out.println(
43                 "Rueckgabe von fac mit n = " +
44                 n + " ergibt " +
45                 wert.toString());
46
47             return wert;
48         }
49     }
50 }
```


Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

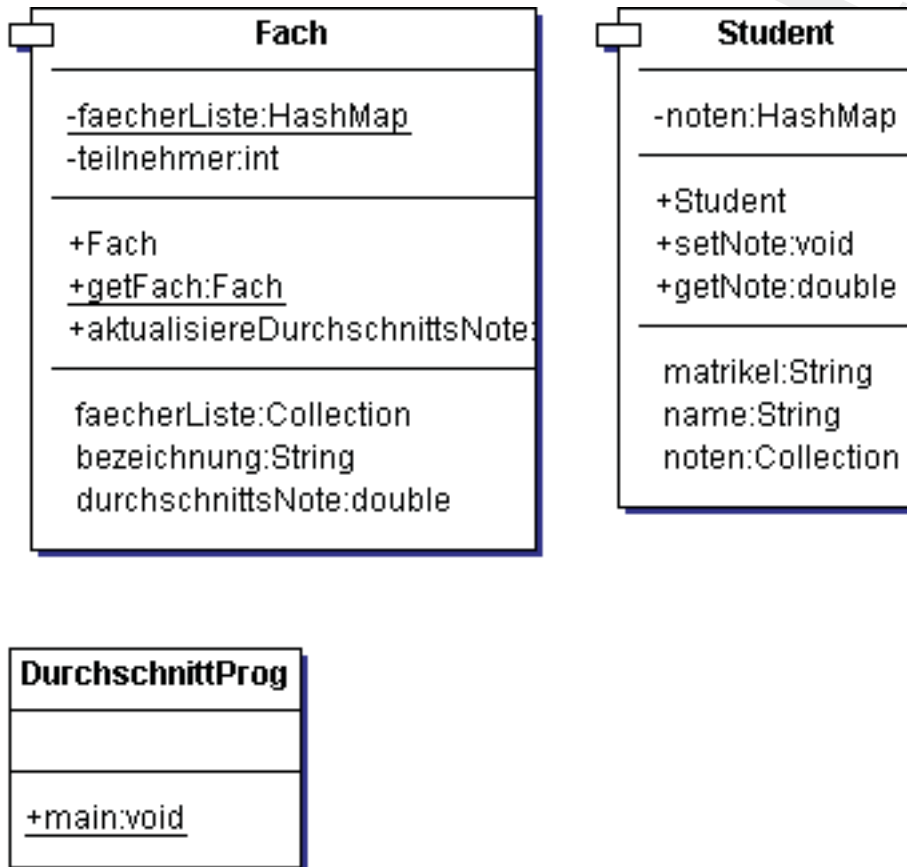
```
>java de.leuphana.ics.rekursion.Rekursion 3
```

A.12 Applikation Durchschnitt mit HashMap

Die HashMap-Klasse entspricht weitgehend der Hashtable-Klasse bis auf die Ausnahme, dass sie unsynchronized ist und nulls erlaubt.

Listing A.14: Fach

```
/**
2  * Fach mit Teilnehmeranzahl und Durchschnittsnote
3  *
4  *
5  * @since      18-May-2005, ... , 18-Jul-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.3
8  */
9
10 package de.leuphana.ics.durchschnitt;
11
12 import java.util.Collection;
13 import java.util.HashMap;
14
15 public class Fach {
16
17     private static HashMap<String, Fach> faecherListe =
18         new HashMap<String, Fach>();
19
20     private String bezeichnung;
21     private int teilnehmer;
22     private double durchschnittsnote;
23
24     public Fach(String bezeichnung)
25     {
26         this.bezeichnung = bezeichnung;
27         faecherListe.put(bezeichnung, this);
28     }
29
30     public static Collection getFaecherListe()
31     {
32         return faecherListe.values();
33     }
34 }
```



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center*™ 6.2.

Abbildung A.3: Beispiel: Fach-Student-Durchschnittsnote

```
34     public static Fach getFach(String bezeichnung)
35     {
36         return (Fach) faecherListe.get(bezeichnung);
37     }
38
39     public String getBezeichnung()
40     {
41         return bezeichnung;
42     }
43
44     public double getDurchschnittsNote()
45     {
46         return durchschnittsNote;
47     }
48
49     public void aktualisiereDurchschnittsNote(double note)
50     {
51         durchschnittsNote =
52             (durchschnittsNote * teilnehmer + note)
53             / ++teilnehmer;
54     }
55 }
56 }
```

Listing A.15: Student

```
/**
2  * Student mit Matrikelnummer und Noten
3  *
4  *
5  * @since 18-May-2005, 12-Nov-2006, 28-May-2007
6  * @author Hinrich E. G. Bonin
7  * @version 1.2
8  */
9  package de.leuphana.ics.durchschnitt;
10
11 import java.util.Collection;
12 import java.util.HashMap;
13
14 public class Student {
15
16     private String name;
17     private String matrikel;
18     private HashMap noten = new HashMap();
19
20     public Student(String name, String matrikel)
21     {
22         this.name = name;
23         this.matrikel = matrikel;
24     }
25 }
```

```
24     }
26     public void setNote(Fach fach , double note)
27     {
28         noten.put(fach , new Double(note));
29         fach.aktualisiereDurchschnittsNote(note);
30     }
32     public String getMatrikel()
33     {
34         return matrikel;
35     }
36     public void setMatrikel(String matrikel)
37     {
38         this.matrikel = matrikel;
39     }
40
42     public String getName()
43     {
44         return name;
45     }
46     public void setName(String name)
47     {
48         this.name = name;
49     }
50
52     public Collection getNoten()
53     {
54         return noten.values();
55     }
56     public double getNote(Fach fach)
57     {
58         return ((Double)
59             noten.get(fach)).doubleValue();
60     }
62 }
```

Listing A.16: DurchschnittProg

```
/**
2  * Applikation zur Durchschnittsnotenberechnung
3  *
4  * @since      18-May-2005, 12-Nov-2006, 28-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.2
*/
```

```
8 package de.leuphana.ics.durchschnitt;
10 public class DurchschnittProg {
12     public static void main(String [] args)
13     {
14
15         Student student1 =
16             new Student("Ewin┘Ete", "12345");
17         Student student2 =
18             new Student("Klara┘Witwe", "444444");
19         Student student3 =
20             new Student("Emma┘Schulze", "98765");
21
22         Fach prog = new Fach("Programmierung");
23         Fach theo = new Fach("Theoretische┘Informatik");
24
25         student1.setNote(Fach.getFach("Programmierung"), 3.0);
26         student2.setNote(Fach.getFach("Programmierung"), 2.0);
27         student3.setNote(Fach.getFach("Programmierung"), 1.0);
28
29         student1.setNote(theo, 1.0);
30         student2.setNote(theo, 2.0);
31         student3.setNote(theo, 2.0);
32
33         System.out.println("Note┘von┘"
34             + student1.getName() + "┘"
35             + prog.getBezeichnung() + "┘:"
36             + student1.getNote(theo));
37
38         System.out.println("\nDurchschnittsnoten:");
39         System.out.println(prog.getBezeichnung() + "┘="
40             + prog.getDurchschnittsNote());
41         System.out.println(theo.getBezeichnung() + "┘="
42             + (double) Math.round(
43             theo.getDurchschnittsNote() * 10) / 10.0);
44     }
45 }
```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.leuphana.ics.durchschnitt.DurchschnittProg
```

A.13 Assoziation: Foo ↔ Bar

Listing A.17: Foo

```

/**
2  * Assoziationsbeispiel Foo --> Bar --> Foo
  *
4  *
  * @since      01-Dec-1998, 03-Dec-2002, 28-May-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.assozi;

10
class Foo
12 {
  private Bar v;
14  public static Foo c;

16  public Bar getV ()
  {
18    return v;
  }

20  public void setV(Bar v)
22  {
  this.v = v;
24  }

26  public static void main(String [] args)
  {
28    Foo b = new Foo ();
    b.setV(new Bar ());
30    Foo.c = b;
    if (Foo.c.getV () instanceof Bar)
32    {
      System.out.println (
34        "Alles durchdacht? Foo!");
    }
36  }
}

```

Listing A.18: Bar

```

/**
2  * Assoziationsbeispiel Bar --> Foo --> Bar
  *

```

```
4  * @since      01-Dec-1998, 03-Dec-2002, 28-May-2007
   * @author    Hinrich E. G. Bonin
6  * @version    1.3
   */
8
9  package de.leuphana.ics.assozi;
10
11 class Bar
12 {
13     private Foo v;
14     public static Bar c;
15
16     public Foo getV ()
17     {
18         return v;
19     }
20
21     public void setV(Foo v)
22     {
23         this.v = v;
24     }
25
26     public static void main(String [] args)
27     {
28         Bar b = new Bar ();
29         b.setV(new Foo ());
30         Bar.c = b;
31         Bar.c.getV (). setV(new Bar ());
32         if ( Bar.c.getV ().getV () instanceof Bar)
33             {
34                 System.out.println (
35                     "Alles durchdacht? Bar!");
36                 Foo.main(new String [0]);
37             } else
38             {
39                 System.out.println ("OK");
40             }
41     }
42 }
```

Geben Sie bei den folgenden Aufrufen das jeweilige Ergebnisse an:

```
>javac de/leuphana/ics/assozi/Foo.java
>dir de\leuphana\ics\assozi\*.class
>java de.leuphana.ics.assozi.Bar
>java de.leuphana.ics.assozi.Foo
```

A.14 Gleichnamige Attributen: SlotI

Listing A.19: SlotI

```

/**
 2  * Beispiel zur Frage der Vererbung bei
 3  * gleichnamigen Slots (Attributen)
 4  *
 5  * @since      21-Dec-1998, 26-Nov-2002, 28-May-2007
 6  * @author     Hinrich E. G. Bonin
 7  * @version    1.3
 8  */
 9  package de.leuphana.ics.vererbung;
10
11  class SlotI
12  {
13      public static void main(String [] args)
14      {
15          Foo f = new Foo();
16          System.out.println(
17              "f.getl()=_" +
18              f.getl());
19
20          Bar b = new Bar();
21          System.out.println(
22              "b.getl()=_" +
23              b.getl());
24
25          b.setl(3);
26          System.out.println(
27              "b.setl(3)_dann_b.getl()=_" +
28              b.getl());
29          System.out.println(
30              "b.setl(3)_dann_b.i=_" +
31              b.i);
32
33          f.setl(4);
34          System.out.println(
35              "f.setl(4)_dann_f.getl()=_" +
36              f.getl());
37          System.out.println(
38              "f.setl(4)_dann_b.getl()=_" +
39              b.getl());
40      }
41  }

```

Listing A.20: Foo

```

/**

```



```

2  * Beispiel zur Frage der Vererbung bei
   * gleichnamigen Slots (Attributen)
4  *
   * @since      21-Dec-1998, 26-Nov-2002, 28-May-2007
6  * @author     Hinrich E. G. Bonin
   * @version    1.3
8  */
package de.leuphana.ics.vererbung;

10
class Foo
12 {
   private int i = 1;
14
   public int getI()
16   {
     return this.i;
18   }

   public void setI(int i)
20   {
     this.i = i;
22   }
24 }

```

Listing A.21: Bar

```

/**
2  * Beispiel zur Frage der Vererbung bei
   * gleichnamigen Slots (Attributen)
4  *
   * @since      21-Dec-1998, 26-Nov-2002, 28-May-2007
6  * @author     Hinrich E. G. Bonin
   * @version    1.3
8  */
package de.leuphana.ics.vererbung;

10
class Bar extends Foo
12 {
   int i = 2;
14 }

```

Geben Sie bei den folgenden Aufrufen das jeweilige Ergebnisse an:

```

>javac de/leuphana/ics/vererbung/SlotI.java
>java de.leuphana.ics.vererbung.SlotI

```

A.15 Applikation QueueProg — Fall I

Hinweis: Aufgabenidee aus [Freeman/Ince96] entnommen. Quellcode stark modifiziert und ergänzt.

Listing A.22: Queue — Fall I

```

/**
2  * Kleine Kostprobe fuer eine „Zirkulaere Liste“
  * mit dem Prinzip „first-in-first-out“
4  *
  * @since      10-Apr-1998, 26-Nov-2002, 28-May-2007
6  * @author     Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.queue;

10
final class Queue
12 {
    private final int queueCapacity = 3;
14     private String circleList[]
        = new String[queueCapacity];
16     private int noOfItemsInQueue = 0;
    private int frontOfTheQueue = 0;
18     static int noOfQueues = 0;

20     Queue()
    {
22         noOfQueues++;
    }

24     public int getQueueCapacity()
26     {
        return queueCapacity;
28     }

30     public int getNoOfItemsInQueue()
    {
32         return noOfItemsInQueue;
    }

34     public boolean isEmpty()
36     {
        return (noOfItemsInQueue == 0);
38     }

40     public boolean isQueueFull()
    {
42         return (noOfItemsInQueue ==

```

```

        queueCapacity);
44     }
46     private void addNthItem (
            int n,
48         String itemToBeAdded)
        {
50         int index;
            index = frontOfTheQueue + (n - 1);
52         if (index >= queueCapacity)
            {
54             index = index % queueCapacity;
            }
56         circleList[index] = itemToBeAdded;
        }
58     public boolean addItem (
            String itemToBeAdded)
        {
62         if (this.isQueueFull ())
            {
64             System.out.println (
                "Item „" + itemToBeAdded +
66             " „nicht „aufgenommen!");
                return false;
68             } else
            {
70             noOfItemsInQueue++;
                this.addNthItem (
72                 noOfItemsInQueue,
                    itemToBeAdded);
74             return true;
            }
76         }
78     public String getFirstItem ()
        {
80         if (this.isQueueEmpty())
            {
82             return "";
            } else
84         {
                return circleList[frontOfTheQueue];
86         }
        }
88     public boolean removeFirstItem ()
90     {
        if (this.isQueueEmpty())

```

```

92     {
93         System.out.println(
94             "Kein Item entfernbar");
95         return false;
96     } else
97     {
98         noOfItemsInQueue--;
99         if (frontOfTheQueue ==
100            (queueCapacity - 1))
101         {
102             frontOfTheQueue = 0;
103         } else
104         {
105             frontOfTheQueue++;
106         }
107         return true;
108     }
109 }
110
111 public boolean isItemInQueue(String item)
112 {
113     int count = 0;
114     while (count < noOfItemsInQueue)
115     {
116         count++;
117         if (this.getNthItem(count) ==
118            item)
119         {
120             return true;
121         }
122     }
123     return false;
124 }
125
126 private String getNthItem(int n)
127 {
128     int index;
129     index = frontOfTheQueue + (n - 1);
130     if (index >= queueCapacity)
131     {
132         index = index % queueCapacity;
133     }
134     return (circleList[index]);
135 }
136 }

```

Listing A.23: QueueProg — Fall I

/**

```

2  * Kleine Kostprobe fuer eine ,,Zirkulaere Liste''
3  * mit dem Prinzip ,,first-in-first-out''
4  *
5  * @since      10-Apr-1998, 26-Nov-2002, 28-May-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.3
8  */
9  package de.leuphana.ics.queue;
10
11 public class QueueProg
12 {
13     public static void main(String [] args)
14     {
15         Queue myQ = new Queue();
16         Queue myL = new Queue();
17         System.out.println (
18             "Step 0: Queue.noOfQueues=" +
19             Queue.noOfQueues);
20         System.out.println (
21             "Step 1: myQ.getQueueCapacity()=" +
22             myQ.getQueueCapacity());
23
24         myQ.addItem ("Otto AG");
25         myQ.addItem ("Emma AG");
26         myQ.addItem ("Klara AG");
27         myQ.removeFirstItem ();
28
29         System.out.println (
30             "Step 2: myQ.getFirstItem()=" +
31             myQ.getFirstItem());
32
33         myQ.addItem ("Willi AG");
34         myQ.addItem ("Ernst AG");
35         myQ.removeFirstItem ();
36         myQ.removeFirstItem ();
37         myQ.addItem ("Ernst AG");
38
39         System.out.println (
40             "Step 3: myQ.getFirstItem()=" +
41             myQ.getFirstItem());
42
43         System.out.println (
44             "Step 4: myQ.getNoOfItemsInQueue=" +
45             myQ.getNoOfItemsInQueue());
46
47         boolean inCircleList =
48             myQ.isItemInQueue ("Ernst AG");
49         System.out.println (
50             "Step 5: myQ.isItemInQueue (Ernst AG)="

```

```

52     + inCircleList);
    }
}

```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.leuphana.ics.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.

Listing A.24: Queue — Fall II

```

/**
2  * Description of QueueProg the Interface
  *
4  * @since      30-May-2005, 28-May-2007
  * @author     Hinrich E. G. Bonin
6  * @version    1.1
  */
8  package de.leuphana.ics.queue;

10 import java.util.Iterator;
import java.util.NoSuchElementException;
12
14 public interface Queue {
    public boolean add(Object o);

16    public Object retrieve()
        throws NoSuchElementException;

18    public Iterator iterator();
20 }

```

Listing A.25: LinkedQueue — Fall II

```

/**
2  * Implements Queue
  *
4  * @since      30-May-2005, 28-May-2007
  * @author     Hinrich E. G. Bonin
6  * @version    1.1

```

```
*/
8 package de.leuphana.ics.queueell;
10 import java.io.Serializable;
11 import java.util.Iterator;
12 import java.util.NoSuchElementException;
14 public class LinkedList implements
15     Queue, Serializable {
16
17     protected ElementWrapper first;
18     protected ElementWrapper last;
19
20     protected int count;
21
22     public LinkedList()
23     {
24         first = last = null;
25         count = 0;
26     }
27
28     public boolean add(Object o)
29     {
30         if (count == 0) {
31             first = new ElementWrapper();
32             last = first;
33             count = 1;
34         } else {
35             last.next = new ElementWrapper();
36             last = last.next;
37             ++count;
38         }
39         last.element = o;
40         last.next = null;
41         return true;
42     }
43
44     public Object retrieve()
45     throws NoSuchElementException
46     {
47         if (count <= 0)
48             {
49                 throw new NoSuchElementException();
50             }
51         ElementWrapper ret = first;
52         --count;
53         first = first.next;
54         if (first == null)
55             {
```

```

56         last = null;
           count = 0;
58     }
    return ret.element;
60 }

62 public Iterator iterator()
    {
64     return
        new Iterator(){
66         ElementWrapper tmp = first;

68         public boolean hasNext()
        {
70             return tmp != null;
        }

72         public Object next()
        {
74             if (tmp == null)
76                 {
78                     throw new
                        NoSuchElementException ();
79                 }
80             Object ret = tmp.element;
81             tmp = tmp.next;
82             return ret;
83         }

84         public void remove()
85         {
86             throw new
87                 UnsupportedOperationException ();
88         }
89     };
90 }

92 class ElementWrapper implements Serializable
93 {
94     public Object element;
95
96     public ElementWrapper next;
97 }
98 }

```

Listing A.26: QueueProg — Fall II

```

/**
2  * Description of the Class QueueProg

```



```
*
* @since      30-May-2005, 28-May-2007
* @author     Hinrich E. G. Bonin
* @version    1.1
*/
8 package de.leuphana.ics.queueell;

10 import java.io.FileInputStream;
import java.io.FileOutputStream;
12 import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
14 import java.util.Iterator;

16 public class QueueProg
{
18     static LinkedList queue = new LinkedList();

20     private static void showQueue()
    {
22         System.out.println("\nElemente_in_der_Queue:");
        Iterator it = queue.iterator();
24         while (it.hasNext())
            {
26             System.out.println(it.next().toString());
            }
28     }

30     private static void testQueue(int anzahl)
    {
32         for (int i = 1; i <= anzahl; ++i)
            {
34             queue.add(Integer.toString(i));
            }
36     }

38     private static void writeQueueToFile(String dateiname)
    {
40         try
            {
42             FileOutputStream fileout =
                new FileOutputStream(dateiname);
44             ObjectOutputStream objout =
                new ObjectOutputStream(fileout);
46             objout.writeObject(queue);
                objout.close();
48             System.out.println("\nQueue_in_Datei_"
                + dateiname
                + "_gespeichert.");
50         } catch (Exception e) {
```

```
52         e.printStackTrace(System.out);
53     }
54 }
55
56 private static void readQueueFromFile(String dateiname)
57 {
58     try
59     {
60         FileInputStream filein =
61             new FileInputStream(dateiname);
62         ObjectInputStream objectin =
63             new ObjectInputStream(filein);
64         queue = (LinkedList)
65             objectin.readObject();
66         System.out.println(
67             "\nQueue wurde aus Datei "
68             + dateiname
69             + " gelesen.");
70     } catch (Exception e) {
71         e.printStackTrace(System.out);
72     }
73 }
74
75 public static void main(String [] args)
76 {
77     int anzahl = 1;
78     try
79     {
80         anzahl = Integer.parseInt(args[0]);
81     } catch (ArrayIndexOutOfBoundsException e)
82     {
83         System.err.println(
84             "Keine Elementanzahl angegeben!");
85         System.exit(1);
86     } catch (NumberFormatException e)
87     {
88         System.err.println(
89             "Elementanzahl nicht interpretierbar!");
90         System.exit(1);
91     }
92     if (anzahl < 1)
93     {
94         anzahl = 1;
95         System.out.println(
96             "Die Queue wurde auf Anfangsgroesse = 1 gesetzt!");
97     }
98     QueueProg.testQueue(anzahl);
99     QueueProg.showQueue();
100 }
```

```

102     Object head = queue.retrieve();
103     if (head != null)
104     {
105         System.out.println (
106             "\n1. Element entnommen: "
107             + head);
108     } else {
109         System.out.println ("\nQueue ist leer!");
110     }
111
112     QueueProg.showQueue ();
113
114     queue.add ("Emma Musterfrau");
115     System.out.println ("üEingefgt : Emma Musterfrau");
116     queue.add ("Hans Otto");
117     System.out.println ("üEingefgt : Hans Otto");
118     queue.add ("Karl Stein");
119     System.out.println ("üEingefgt : Karl Stein");
120
121     QueueProg.writeQueueToFile ("queue.ser");
122     QueueProg.readQueueFromFile ("queue.ser");
123
124     QueueProg.showQueue ();
125 }
126 }

```

Geben Sie bei dem folgenden Aufruf das Ergebnisse an:

```
>java de.leuphana.ics.queueII.QueueProg 3
```

A.17 Applet SimpleThread

Listing A.27: SimpleThread.html

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
4  <!-- Primitives Testbett fuer Applet SimpleThread -->
5  <!-- Bonin 07-May-1998 -->
6  <!-- update 07-Jan-2003 -->
<head>
8  <title>Mond am Himmel</title>
</head>
10 <body>

```

```

<h1>Mond am Himmel</h1>
12 <h1>
<applet name="Mond" code="SimpleThread.class"
14   width="450" height="120" alt="Mond am Himmel">
    Hier soll SimpleThread-Applet laufen!
16 </applet>
</h1>
18 <p>Der Mond ist aufgegangen ...</p>
<p>Copyright Bonin 07-May-1998 all rights reserved</p>
20 <address>
<a href="mailto:bonin@fhnon.de">bonin@fhnon.de</a>
22 </address>
</body>
24 <!-- File C:\bonin\anwd\code\SimpleThread.html -->
</html>

```

Listing A.28: SimpleThread

```

/**
2  * Kleines Beispiel fuer eine „Animation mittels Thread“,
  * Idee aus: Hubert Partl; Java-Einfuehrung, Version April
4  * 1998, S. 82 http://www.boku.ac.at/javaeinf/ Quellcode
  * leicht modifiziert
6  *
  * @author    Hinrich Bonin
8  * @version   1.0
  * @since     08-Mai-1998
10 * /
import java.applet.*;
12 import java.awt.*;

14 public class SimpleThread extends Applet
    implements Runnable
16 {

18     int x, y, width, height;
    Graphics grafik;
20     Image bild;
    Color nachtFarbe = new Color(0, 0, 102);
22     Color mondFarbe = new Color(204, 204, 255);

24     Thread myT = null;

26

28     public void init()
    {
30         Dimension d = getSize();
        width = d.width;
        height = d.height;

```

```
32     bild = createImage(width , height);
33     grafik = bild.getGraphics ();
34     x = width / 2;
35     y = height / 2;
36     System.out.println ("x=" + x + " y=" + y);
37 }
38
39
40 public void start ()
41 {
42     if (myT == null)
43     {
44         myT = new Thread (this);
45         myT.start ();
46     }
47     System.out.println (" start () appliziert");
48 }
49
50
51 public void stop ()
52 {
53     if (myT != null)
54     {
55         myT.stop ();
56         myT = null;
57     }
58     System.out.println ("stop () appliziert");
59 }
60
61
62 public void run ()
63 {
64     while (true)
65     {
66         grafik.setColor (nachtFarbe);
67         grafik.fillRect (0, 0, width, height);
68         grafik.setColor (mondFarbe);
69         grafik.fillArc (x, y - 25, 50, 50, 270, 180);
70         x += 2;
71         if (x > width + 50)
72         {
73             x = -50;
74         }
75         repaint ();
76     }
77     try
78     {
79         System.out.println (
80             "In run () vor Thread.sleep(1000)");
81         Thread.sleep (1000);
82     }
83     catch (InterruptedException e)
84     {
85     }
86 }
```

```

82     System.out.println (
        "In_run() nach Thread.sleep(1000)");
84     } catch (InterruptedException e)
        {
86         System.out.println (
            "In_run() Fehler");
88     }
90 }

92 public void paint(Graphics g)
    {
94     update(g);
96 }

98 public void update(Graphics g)
    {
100     if (bild != null)
        {
102         g.drawImage(bild, 0, 0, this);
104     }
    }
}

```

Skizzieren Sie bei dem folgenden Aufruf das Ergebnis:

```
>appletviewer SimpleThread.html
```

Oder nutzen Sie einen Browser mit einer Java 2 Plattform.

A.18 Applet DemoAWT

Listing A.29: ExampleAWT.html

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
4 <!-- Testbett fuer Applet DemoAWT -->
5 <!-- Bonin 19-April 1998 -->
6 <!-- Update 08-Jan-2003 31-May-2007 -->
<head>
8 <title>Willi liebt Sport</title>
</head>

```

```

10 <body>
    <h1>Willi liebt Sport</h1>
12 <h1>
    <applet name="Willi"
14     code="de.leuphana.ics.awt.DemoAWT.class"
        width="350" height="120"
16     alt="Willi_will_will_Ausdauersport">
        Willi will Ausdauersport!
18 </applet>
    </h1>
20 <p>Copyright Bonin 1998--2007 all rights reserved</p>
    <address>
22 <a href="mailto:bonin@uni-lueneburg.de">bonin@uni-lueneburg.de</a>
    </address>
24 </body>
</html>

```

Listing A.30: MaskeAufbau

```

/**
2  * Kleines Beispiel fuer
    * "Abstract Window Toolkit" (awt)
4  *
    * @since      14-Apr-1998, 31-May-2007
6  * @author     Hinrich E. G. Bonin
    * @version    1.1
8  */
package de.leuphana.ics.awt;
10
import java.applet.Applet;
12 import java.awt.*;

14 class MaskeAufbau extends Applet
    {
16     Panel topPanel,
        leftPanel,
18     centerPanel,
        rightPanel,
20     bottomPanel;

22     public void doUserInterface (Frame frame)
        {
24         frame.setLayout(new BorderLayout());
            topPanel = new Panel();
26         leftPanel = new Panel();
            centerPanel = new Panel();
28         rightPanel = new Panel();
            bottomPanel = new Panel();
30         frame.add("North", topPanel);

```

```

32     frame.add("West", leftPanel);
33     frame.add("Center", centerPanel);
34     frame.add("East", rightPanel);
35     frame.add("South", bottomPanel);
36
37     MenuBar myMbar = new MenuBar();
38
39     Menu myMTria = new Menu("Triathlon");
40     myMTria.add(new MenuItem("Schwimmen"));
41     myMTria.add(new MenuItem("Radfahren"));
42     myMTria.add(new MenuItem("Laufen"));
43     myMbar.add(myMTria);
44
45     Menu myMDua = new Menu("Duathlon");
46     myMDua.add(new MenuItem("1. Laufen"));
47     myMDua.add(new MenuItem("Radfahren"));
48     myMDua.add(new MenuItem("2. Laufen"));
49     myMbar.add(myMDua);
50
51     frame.setMenuBar(myMbar);
52 }

```

Listing A.31: MyCanvas

```

1  /**
2   * Kleines Beispiel fuer
3   * "Abstract Window Toolkit" (awt)
4   *
5   * @since      14-Apr-1998, 31-May-2007
6   * @author     Hinrich E. G. Bonin
7   * @version    1.1
8   */
9  package de.leuphana.ics.awt;
10
11  import java.awt.*;
12
13  class MyCanvas extends Canvas
14  {
15      public final int width = 80;
16      public final int height = 120;
17
18      public void paint(Graphics g)
19      {
20          /**
21           * x-Achse: waagerecht von links nach rechts
22           * y-Achse: senkrecht von oben nach unten
23           * rgbWert: jeweils 0...255
24           */

```



```

26     int x;
        /*
28         * x-Achse:  waagerecht von links nach rechts
        * y-Achse:  senkrecht von oben nach unten
        * rgbWert:  jeweils 0...255
30         */
        int y;
32         /*
        * x-Achse:  waagerecht von links nach rechts
34         * y-Achse:  senkrecht von oben nach unten
        * rgbWert:  jeweils 0...255
36         */
        int rgbWert;
38         for (x = 0, y = 0, rgbWert = 0;
            (x < (width / 2)) &&
40             (y < (height / 2) && (rgbWert < 256));
            x += 2, y += 2, rgbWert += 6)
42             {
                g.setColor(new Color(255 - rgbWert,
44                             rgbWert, 0));
                g.fillRect(x, y, width - (2 * x),
46                             height - (2 * y));
            }
48         g.setColor(Color.blue);
        g.drawString("D_T_U",
50                     (width - g.getFontMetrics().stringWidth(
                        "D_T_U")) / 2,
52                     height / 2);
        }
54
        /*
56         * minimumSize() wird vom Layout-Manger aufgerufen ,
        * um zu erfahren, wie gross der minimale Platz ist ,
58         * der benoetigt wird.
        */
60         public Dimension minimumSize()
        {
62             return new Dimension(width + 20, height + 20);
        }
64
        /*
66         * preferredSize() wird vom Layout-Manager aufgerufen ,
        * um zu erfahren, wie gross man es gern haette.
68         */
        public Dimension preferredSize ()
70         {
            return this.minimumSize();
72         }
    }

```

Listing A.32: SimpleListener

```

/**
2  * Kleines Beispiel fuer
  * "Abstract Window Toolkit" (awt)
4  *
  * @since 14-Apr-1998, 31-May-2007
6  * @author Hinrich E. G. Bonin
  * @version 1.1
8  */
package de.leuphana.ics.awt;

10
import java.awt.*;
12 import java.awt.event.ActionListener;

14 class SimpleListener implements ActionListener
  {
16   private Frame fr;

18   public SimpleListener (Frame fr)
  {
20     this.fr = fr;
  }

22   public void actionPerformed (
24     java.awt.event.ActionEvent e)
  {
26     String name = e.getActionCommand();
     System.out.println (
28       "actionPerformed() appliziert: " +
       name);

30     if (name.equals("Anmelden!"))
32     {
       fr.setTitle("Danke-Willi!");
34     }
     if (name.equals("Absagen!"))
36     {
       fr.setTitle("Schade-Willi!");
38     }
  }
40 }

```

Listing A.33: DemoAWT

```

/**
2  * Kleines Beispiel fuer
  * "Abstract Window Toolkit" (awt)
4  *
  * @since 14-Apr-1998, 31-May-2007

```

```
6  * @author    Hinrich E. G. Bonin
7  * @version   1.1
8  */
9  package de.leuphana.ics.awt;
10
11 import java.awt.*;
12
13 public class DemoAWT extends MaskeAufbau
14 {
15     Frame myFrame =
16         new Frame("Willi_will_Ausdauersport!");
17
18     public void init()
19     {
20         DemoAWT myDemo = new DemoAWT();
21         myDemo.doUserInterface(myFrame);
22         myFrame.pack();
23         myFrame.show();
24     }
25
26     public void stop()
27     {
28         System.out.println("stop()_appliziert!");
29     }
30
31     public void doUserInterface(Frame frame)
32     {
33         super.doUserInterface(frame);
34
35         topPanel.setLayout(new GridLayout(1, 2));
36         topPanel.add(new Checkbox("DTU-Lizenz"));
37         Choice myC = new Choice();
38         myC.addItem("Kurzdistanz");
39         myC.addItem("Mitteldistanz");
40         myC.addItem("Langdistanz");
41         topPanel.add(myC);
42
43         Button anmelden = new Button("Anmelden!");
44         anmelden.addActionListener(
45             new SimpleListener(frame));
46         leftPanel.add(anmelden);
47
48         centerPanel.add(new MyCanvas());
49
50         Button absagen = new Button("Absagen!");
51         absagen.addActionListener(
52             new SimpleListener(frame));
53         rightPanel.add(absagen);
54     }
55 }
```

```

56     int widthDTUinPixel = new MyCanvas().width;
bottomPanel.add(new TextArea(
58         "Beschreiben Sie genau Ihren Leistungsstand!",
3, widthDTUinPixel / 2));
60 }

```

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

```

>javac de/leuphana/ics/regal/Regal.java
>java de.leuphana.ics.regal.Regal$Application

```

Listing A.34: Regal

```

/**
2  * Inner-Classes-Beispiel
3  *
4  * @since 22-Jan-1999, 26-Nov-2002, 31-May-2007
5  * @author Hinrich E. G. Bonin
6  * @version 1.3
7  */
8  package de.leuphana.ics.regal;
9
10 public final class Regal
11 {
12     private static int anzahl = 0;
13     private int anzahlSchubladen = 0;
14     private int benutztAnzahl = 0;
15
16     public Regal()
17     {
18         Regal.anzahl++;

```

```
    }
20
    public class Schublade
22    {
        private boolean belegt = false;
24        private int gezogenAnzahl = 0;
        private String inhalt = "Leer";
26
        public Schublade ()
28        {
            anzahlSchubladen++;
30        }

        public String getInhalt ()
32        {
            benutztAnzahl++;
34            gezogenAnzahl++;
            return inhalt;
36        }

        public void setInhalt(String inhalt)
38        {
            this.inhalt = inhalt;
40            benutztAnzahl++;
            gezogenAnzahl++;
42            belegt = true;
            }
44        }
46    }

    public static class Application
48    {
        public static void main(String argv [])
50        {
            Regal baz = new Regal ();
52            Regal bar = new Regal ();
            Regal foo = baz;
54

            Regal.Schublade fooS1
56                = foo.new Schublade ();
            Regal.Schublade fooS2
58                = foo.new Schublade ();
            fooS1.setInhalt("Java-Disketten");
60            fooS2.setInhalt("Java-Artikel");

            Regal.Schublade barS1
62                = bar.new Schublade ();
            barS1.setInhalt("Java-CD-ROM");
64

            System.out.println (
66
```

```

68         "Das Regalsystem hat " +
        Regal.anzahl +
70         " Regal(e).");

72     System.out.println (
        "Das Regal foo hat " +
74     baz.anzahlSchubladen +
        " Schubladen.");

76     if (fooS1.belegt || fooS2.belegt)
78     {
        System.out.println (
80         "Schubladeneinhalte : " +
        "\n" + fooS1.getInhalt () +
82         "\n" + fooS2.getInhalt ());
    }
84     System.out.println (
        "Das Regal foo wurde " +
86     baz.benutztAnzahl +
        " x benutzt.");
88     }
90 }

```

A.19.1 Erzeugte Klassen feststellen

Stellen Sie fest, ob die Datei `Regal` fehlerfrei kompiliert werden konnte und geben Sie an, welche Dateien nach dem Kompilieren 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 Zwei Main-Methoden

Notieren Sie eine Datei `Ganze.java`, die zwei Main-Methoden enthält und mit den hier protokollierten Aufrufen das folgende Ergebnis erzeugt.

Protokolldatei `Ganze.log`

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/all/Ganze.java

D:\bonin\anwd\code>java
  de.leuphana.ics.all.Ganze
Gib Ganze!

D:\bonin\anwd\code>java
  de.leuphana.ics.all.Ganze$Teil
Gib Teil!

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

A.21 Anonyme Klassen

Die Programmiererin *Emma Softy* will einzelnen Objekten einer Klasse jeweils eine eigene Ausprägung einer Methode zuordnen. Ihr Lösungsansatz basiert auf „anonymen Klassen“. Um ihren Lösungsansatz zu überprüfen schreibt sie folgende Klasse Anonym.

Listing A.35: Anonym

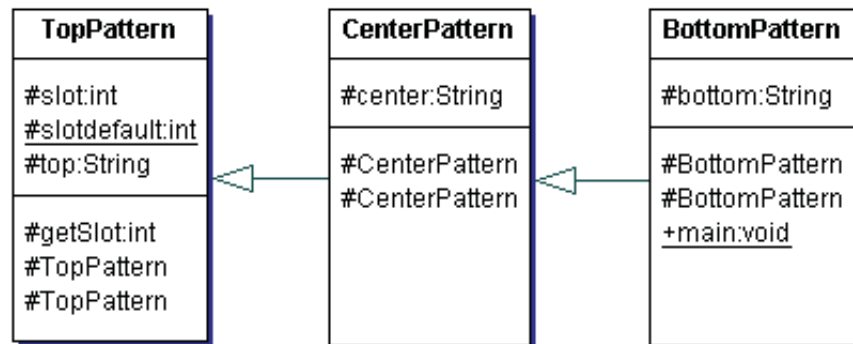
```
/**
2  * Kleines Beispiel fuer
  * Anonyme-Klassen
4  * --- salopp formuliert:
  * Objekte einer Klasse haben
6  * eigene Methoden
  *
8  * @author    Bonin
  * @created   10-May-2007
10 * @version   1.0
  */
12 package de.leuphana.ics.anonym;

14 public class Anonym
  {
16     static int number = 0;

18     String value = "value";
```

```
20     String getValue()
21     {
22         return value;
23     }
24     Anonym()
25     {
26         number++;
27         System.out.println(
28             "Creating object: " + number
29         );
30     }
31
32     Anonym(String value)
33     {
34         this();
35         this.value = value;
36     }
37
38     public static void main(String [] args)
39     {
40
41         Anonym foo = new Anonym()
42         {
43             String getValue()
44             {
45                 return "You are foo!";
46             }
47         };
48
49         Anonym bar = new Anonym()
50         {
51             String getValue()
52             {
53                 return "You are bar!";
54             }
55         };
56
57         Anonym baz = new Anonym("OK!");
58
59         System.out.println(
60             foo.getValue() + "\n" +
61             bar.getValue() + "\n" +
62             baz.getValue());
63     }
64 }
```

Notieren Sie exakt das Ergebnis dieser Java-Applikation und geben Sie dabei an, welche Dateien mit Suffix `.class` beim Compilieren entstanden sind.



Legende:

Notation in *Unified Modeling Language* (UML) *Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center*TM 6.2.

Abbildung A.4: Klassendiagramm von *Emil Cody*

A.22 Konstruktionsalternative

Die Programmiererin *Emma Softy* liegt im harten Konkurrenzkampf mit ihrem Kollegen *Emil Cody*. Sie hat das Java-Package `de.unilueneburg.as.construct` (\leftrightarrow Quellcode S. 460 ff; \leftrightarrow UML-Klassendiagramm A.5 S. 460) erstellt. Er dazu die Alternative `de.unilueneburg.as.structure` (\leftrightarrow S. 457 ff \leftrightarrow UML-Klassendiagramm A.4 S. 457).

A.22.1 Quellcode von *Emil Cody*

Listing A.36: `BottomPattern`

```

/*
2  * Beispiel Construction Reviewing
  *
4  * @author Emil Cody
  * @version 1.0 06-Dec-2006
6  */
package de.unilueneburg.as.structure;
8
public class BottomPattern extends CenterPattern
10 {
    protected String bottom = "";
12
  
```

```

14     protected BottomPattern ()
15     {
16         System.out.println
17             ("-->_BottomPattern()");
18     }

20     protected BottomPattern(int slot ,
21                             String top ,
22                             String center ,
23                             String bottom)
24     {
25         this.slot = slot;
26         this.top = top;
27         this.center = center;
28         this.bottom = bottom;
29         System.out.println
30             ("-->_BottomPattern(int ,_String ,_String ,_String)");
31     }

32     public static void main(String [] args)
33     {
34         BottomPattern foo = new BottomPattern
35             (17, "++", "--", "--");
36
37         System.out.println(foo.getSlot());
38     }
39 }
40 }

```

Listing A.37: CenterPattern

```

/*
2  * Beispiel Construction Reviewing
3  *
4  * @author Emil Cody
5  * @version 1.0 06-Dec-2006
6  */
package de.unilueneburg.as.structure;

8
public class CenterPattern extends TopPattern
10 {
11     protected String center = "";
12
13     protected CenterPattern()
14     {
15         System.out.println
16             ("-->_CenterPattern()");
17     }
18 }

```

```

    protected CenterPattern
20     (int slot , String top , String center)
    {
22         System.out.println
            ("—>_CenterPattern(int ,_String ,_String)");
24         this.slot = slot;
26         this.top = top;
            this.center = center;
28     }
    }

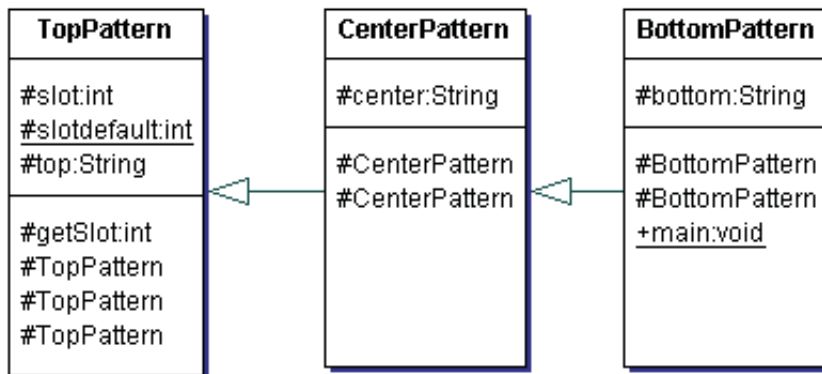
```

Listing A.38: TopPattern

```

/*
2  * Beispiel Construction Reviewing
  *
4  * @author Emil Cody
  * @version 1.0 06-Dec-2006
6  */
package de.unilueneburg.as.structure;
8
public class TopPattern
10 {
    protected int slot = 0;
12     protected static int slotdefault = 13;
14     protected String top = "";
16     protected int getSlot()
    {
18         return slot;
    }
20     protected TopPattern()
    {
22         System.out.println("—>_TopPattern()");
24     }
26     protected TopPattern(int slot , String top)
    {
28         if (slot > 13)
            {
30             System.out.println
                ("slot_=" + slot + "_now_=" +
32                 slotdefault);
                slot = slotdefault;
34             }
            this.slot = slot;

```



Legende:

Notation in *Unified Modeling Language (UML) Class Diagram*.

Hinweis: Gezeichnet mit *Borland Together Control Center™ 6.2*.

Abbildung A.5: Klassendiagramm von *Emma Softy*

```

36     this.top = top;
      System.out.println("—>_TopPattern(int, _String)");
38   }
  }

```

A.22.2 Quellcode von *Emma Softy*

Listing A.39: *BottomPattern*

```

/*
2  * Beispiel Construction Reviewing
  *
4  * @author Emma Softy
  * @version 1.0 06-Dec-2006
6  */
package de.unilueneburg.as.construct;
8
public class BottomPattern extends CenterPattern
10 {
    protected String bottom = "";
12
14     protected BottomPattern()
    {
16         System.out.println
            ("—>_BottomPattern()");
18     }
  }

```

```

20     protected BottomPattern(int slot ,
                               String top ,
22                               String center ,
                               String bottom)
24     {
        super(slot , top , center);
26     this.bottom = bottom;
        System.out.println
28         ("—>_BottomPattern ( int , _String , _String , _String )");
30     }

32     public static void main(String [] args)
    {
34         BottomPattern foo = new BottomPattern
            (17, "++", "—", "--");
36
        System.out.println (foo.getSlot());
38     }
    }

```

Listing A.40: CenterPattern

```

/*
2  * Beispiel Construction Reviewing
  *
4  * @author Emma Softy
  * @version 1.0 06–Dec–2006
6  */
package de.unilueneburg.as.construct;
8
public class CenterPattern extends TopPattern
10 {
    protected String center = "";
12
    protected CenterPattern ()
14     {
        System.out.println
16         ("—>_CenterPattern ()");
    }
18
    protected CenterPattern
20     (int slot , String top , String center)
    {
22         super(slot , top);
        this.center = center;
24         System.out.println
            ("—>_CenterPattern ( int , _String , _String )");

```

```

26     }
    }

```

Listing A.41: TopPattern

```

/*
2  * Beispiel Construction Reviewing
  *
4  * @author Emma Softy
  * @version 1.0 06-Dec-2006
6  */
package de.unilueneburg.as.construct;

8
public class TopPattern
10 {
    protected int slot = 0;
12    protected static int slotdefault = 13;

14    protected String top = "";

16    protected int getSlot()
    {
18        return slot;
    }

20    protected TopPattern()
22    {
        System.out.println
24        ("-->↳TopPattern()");
    }

26

28    protected TopPattern(int slot)
    {
30        if (slot > 13)
        {
32            System.out.println
                ("slot↳↳" +
34                slot +
                "↳NOW↳↳" +
36                slotdefault);
            slot = slotdefault;
38        }
        this.slot = slot;
40        System.out.println
            ("-->↳TopPattern(int)");
42    }

44    protected TopPattern(int slot , String top)

```

```

46     {
        this(slot);
48     this.top = top;
        System.out.println
            ("—>_TopPattern(int, _String)");
50     }
    }

```

A.22.3 Ergebnis feststellen

Stellen Sie fest, ob die Alternativen fehlerfrei kompiliert werden können. Wenn ja, dann geben Sie bitte jeweils das Ergebnis beim Applizieren der Klasse `BottomPattern` an.

A.22.4 Quellcode kürzen

Gibt es in den Alternativen außer den Kommentaren überflüssige Quellcodezeilen? Wenn ja, dann geben Sie bitte die überflüssigen Konstrukte an.

A.22.5 Bewerten der Alternativen

Hat *Emma Softy* oder *Emil Cody* besser programmiert? Begründen Sie Ihre Entscheidung anhand der Vor- und Nachteile der Alternativen.

A.23 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.

Listing A.42: `ptj.opt`

```

1  /**
2   *
   * Konfigurationsdatei üfr das Buch-Beispiel
4   *
5   */
6

```

```
[schemas\dict]
8 name=BuchScholzDict
  oneFile = false
10
[DATABASES\base]
12 name=BuchScholzDB
  schema=dict
14 oneFile = false

16 [classes\Buch]
  persistent = true
18 schema=dict

20 [classes\Autor]
  persistent = true
22 schema=dict

24 [classes\Person]
  persistent = true
26 schema=dict
```

Listing A.43: Buch

```
import com.poet.odmg.*;
2 import java.util.*;

4 public class Buch implements Constraints
  {
6     private String titel;
7     private String isbn;
8     private int erscheinungsJahr;
9     private Autor hauptAutor;
10    private String schlagWoerter;
11    private transient int alter;
12    private String verlagsKurzName;

14
15    public String getTitel()
16    {
17        return titel;
18    }

20
21    public String getIsbn()
22    {
23        return isbn;
24    }

26
```



```
28     public int getErscheinungsJahr ()
29     {
30         return ercheinungsJahr;
31     }
32
33     public Autor getHauptAutor()
34     {
35         return hauptAutor;
36     }
37
38     public String getSchlagWoerter()
39     {
40         return schlagWoerter;
41     }
42
43
44     public int getAlter()
45     {
46         return alter;
47     }
48
49
50     public String getVerlagsKurzName()
51     {
52         return verlagsKurzName;
53     }
54
55
56     public void setTitel(String titel)
57     {
58         this.titel = titel;
59     }
60
61
62     public void setIsbn(String isbn)
63     {
64         this.isbn = isbn;
65     }
66
67
68     public void setErscheinungsJahr (
69         int ercheinungsJahr)
70     {
71         this.ercheinungsJahr = ercheinungsJahr;
72     }
73
74
```

```
76  public void setHauptAutor(Autor hauptAutor)
77  {
78      this.hauptAutor = hauptAutor;
79  }
80
81
82  public void setSchlagWoerter(
83      String schlagWoerter)
84  {
85      this.schlagWoerter = schlagWoerter;
86  }
87
88
89  public void setAlter(int alter)
90  {
91      this.alter = alter;
92  }
93
94
95  public void setVerlagsKurzName(
96      String verlagsKurzName)
97  {
98      this.verlagsKurzName = verlagsKurzName;
99  }
100
101
102  // Methode zur Rekonstruktion
103
104  public void postRead ()
105  {
106      Calendar cal = Calendar.getInstance ();
107      int heute = cal.get(cal.YEAR) - 1900;
108      this.setAlter (heute -
109          this.getErscheinungsJahr ());
110  }
111
112  // Methoden zur üInterfaceerfüllung
113
114  public void preWrite ()
115  {
116      System.out.println (
117          "preWrite-Methode appliziert!");
118  }
119
120
121
122  public void preDelete ()
123  {
124      System.out.println (
```

```
        "preDelete-Methode appliziert!");  
126    }  
    }
```

Listing A.44: Autor

```
import com.poet.odmg.*;  
2 import java.util.*;  
  
4 public class Autor extends Person  
  {  
6     private String themen;  
     private String orgKurzName;  
8  
10    public Autor() { }  
  
12  
     public Autor(String name)  
14    {  
         this();  
16         this.setZuName(name);  
     }  
18  
20    public String getThemen()  
     {  
22         return themen;  
     }  
24  
26    public String getOrgKurzName()  
     {  
28         return orgKurzName;  
     }  
30  
32    public void setThemen(String themen)  
     {  
34         this.themen = themen;  
     }  
36  
38    public void setOrgKurzName(  
         String orgKurzName)  
40    {  
         this.orgKurzName = orgKurzName;  
42    }  
    }
```

Listing A.45: Person

```
import com.poet.odmg.*;
2 import java.util.*;

4 public class Person
  {
6     private String zuName;
    private String vorNamen;
8
10    public String getZuName()
    {
12        return zuName;
    }
14
16    public String getVorNamen()
    {
18        return vorNamen;
    }
20
22    public void setZuName(String zuName)
    {
24        this.zuName = zuName;
    }
26
28    public void setVorNamen(String vorNamen)
    {
30        this.vorNamen = vorNamen;
    }
32 }
```

Listing A.46: BuchBind

```
import com.poet.odmg.*;
2 import org.odmg.ODMGException;
import org.odmg.ObjectNameNotUniqueException;
4 import org.odmg.ODMGRuntimeException;

6 public class BuchBind
  {
8
10    public static void main(String [] argv)
        throws ODMGException
    {
12        Database myDB = new Database ();
        myDB.open ("poet :// oodbserver / BuchScholzDB",
```

```

14         Database.OPEN_READ_WRITE);
Transaction myT = new Transaction(myDB);
16 myT.begin();
try
18 {
    Buch myBuch = new Buch();
20 myBuch.setTitel(
        "Softwarekonstruktion mit LISP");
22 myBuch.setIsbn("3-11-011786-X");
myBuch.setErscheinungsjahr(91);
24 myBuch.setHauptAutor(new Autor("Bonin"));
myBuch.setSchlagwoerter(
26 "Arbeitstechniken, Qualität");
myBuch.postRead();
28 myBuch.setVerlagsKurzName(
        "WalterDeGruyter");
30
    System.out.println(
32         "Zuname des Autors: " +
myBuch.getHauptAutor().getZuName() +
34         "\nAlter des Buches: " +
myBuch.getAlter());
36 myDB.bind(myBuch, "PKS01");
} catch (ObjectNameNotUniqueException exc)
38 {
    System.out.println("PKS01 gibt es schon!");
40 } catch (ODMGRuntimeException exc)
{
42     myT.abort();
    throw exc;
44 }
myT.commit();
46 myDB.close();
}
48 }

```

Listing A.47: BuchLookup

```

import com.poet.odmg.*;
2 import org.odmg.ODMGException;
import org.odmg.ODMGRuntimeException;
4
public class BuchLookup
6 {
8     public static void main(String [] argv)
        throws ODMGException
10    {
        Database myDB = new Database();

```

```

12     myDB.open (
13         "poet://oodbserver/BuchScholzDB",
14         Database.OPEN_READ_WRITE);
15     Transaction myT = new Transaction(myDB);
16     myT.begin();
17     try
18     {
19         Buch myBuch =
20             (Buch) myDB.lookup("PKS01");
21         System.out.println (
22             "Zuname des Autors: " +
23             myBuch.getHauptAutor().getZuName() +
24             "\nAlter des Buches: " +
25             myBuch.getAlter());
26     } catch (ODMGRuntimeException exc)
27     {
28         myT.abort();
29         throw exc;
30     }
31     myT.commit();
32     myDB.close();
33 }
34 }

```

Listing A.48: ListeLookUp

```

/**
2  * Selektieren und Rekonstruieren von mehreren FastObjects
3  * (Buechern)
4  *
5  * @author    Hinrich Bonin
6  * @version   1.0
7  */
8  import com.poet.odmg.util.*;
9  import com.poet.odmg.*;
10 import org.odmg.ODMGException;
11 import org.odmg.ODMGRuntimeException;
12 import java.util.*;
13
14 public class ListeLookUp
15 {
16     public static void main(String[] argv)
17         throws ODMGException
18     {
19         Database myDB = new Database();
20         myDB.open (
21             "poet://oodbserver/BuchScholzDB",
22             Database.OPEN_READ_WRITE);
23         Transaction myT = new Transaction(myDB);

```

```

24     myT.begin();
      try
26     {
          String query =
28         "define_extent_alleBuecher_for_Buch;" +
          "select_buch_from_buch_in_alleBuecher";
30     OQLQuery abfrage = new OQLQuery(query);
          Object result = abfrage.execute();
32     Iterator e =
          ((CollectionOfObject) result).iterator();
34     while (e.hasNext())
          {
36         Buch buch = (Buch) e.next();
          System.out.println(buch.getTitel());
38     }
      } catch (ODMGRuntimeException ore)
40     {
          myT.abort();
42     ore.printStackTrace();
      }
44     myT.commit();
      myDB.close();
46 }
}

```

Skizzieren Sie bei dem folgenden Aufruf das Ergebnis:

```
>java BuchLookup
```

A.24 Vererbung

Die folgende Java-Quelldatei `Foo.java` wurde fehlerfrei kompiliert.

```

>java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)
>javac de/leuphana/ics/innerclass/Foo.java

```

Listing A.49: `Foo`

```
/**
2  * Vererbungsbeispiel
   *
4  * @since      30-Jun-1998 16-May-2007
   * @author     Hinrich Bonin
6  * @version    1.2
   */
8
package de.leuphana.ics.innerclass;
10
class Foo
12 {
13     private static class KlasseA
14     {
15         private static int updateAnzahlSlot = 0;
16         private String slot = "KlasseA";
17
18         public String getSlot()
19         {
20             return slot;
21         }
22
23         public void setSlot(String slot)
24         {
25             updateAnzahlSlot = updateAnzahlSlot + 1;
26             this.slot = slot;
27         }
28     }
29 }
30
31
32 private static class KlasseB extends KlasseA
33 {
34     private static int updateAnzahlSlot = 0;
35     private String slot = "KlasseB";
36
37     public String getSlot()
38     {
39         return slot;
40     }
41
42     public void setSlot(String slot)
43     {
44         updateAnzahlSlot = updateAnzahlSlot + 1;
45         this.slot = slot;
46     }
47 }
48 }
```



```
50     }
52
53     private static class KlasseC extends KlasseB
54     {
55         private static int updateAnzahlSlot = 0;
56         private String slot = "KlasseC";
57
58         public String getSlot()
59         {
60             return slot;
61         }
62     }
63
64     public void setSlot(String slot)
65     {
66         updateAnzahlSlot = updateAnzahlSlot + 1;
67         this.slot = slot;
68     }
69 }
70
71
72 private static class Bar
73 {
74     public static void main(String[] args)
75     {
76         KlasseA a = new KlasseA ();
77         KlasseB b = new KlasseB ();
78         KlasseC c = new KlasseC ();
79
80         b.setSlot(a.getSlot());
81         c.setSlot(b.getSlot());
82         a.setSlot(c.getSlot());
83
84         System.out.println(
85             "Slot-Wert in Instanz c: " +
86             c.getSlot() +
87             "\nAnzahl der Updates in der Klasse A: " +
88             KlasseA.updateAnzahlSlot);
89     }
90 }
91
92 }
```

A.24.1 Erzeugte Dateien angeben

Geben Sie an, welche Dateien nach dem Compilieren von `Foo.java` entstanden sind.

A.24.2 Java-Aufruf angeben

Ersetzen Sie im folgenden Aufruf die drei Punkte.

```
>java de.leuphana.ics.innerclass.Foo...
>
```

A.24.3 Ergebnis des java-Aufrufes angeben

Geben Sie das Ergebnis Ihres Aufrufes an.

A.25 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 Einträgen in das Telefonbuch sorgt.

Listing A.50: `TelefonBuchProg`

```
/**
2  * Einfaches permanentes Telefonbuch
3  * mit Schreibtest
4  *
5  * @since 29-Jun-1998, 30-May-2007
6  * @author Hinrich E. G. Bonin
7  * @version 1.2
8  */
9  package de.leuphana.ics.telefon;
10
11 import java.io.FileInputStream;
12 import java.io.FileOutputStream;
13 import java.io.ObjectInputStream;
14 import java.io.ObjectOutputStream;
15
16 public class TelefonBuchProg
17 {
18     public static void main(String argv[])
19     {
20         TelefonBuch t = new TelefonBuch();
21         t.addEintrag("Key1",
22             new TelefonEintrag(
```

```

24         "Otto",
           "+49/4131/677175");
26     t.addEintrag("Key2",
27                 new TelefonEintrag(
28                     "Emma",
29                     "+49/4131/677144"));
30     try
31     {
32         FileOutputStream fout =
33             new FileOutputStream("tbuch.ser");
34         ObjectOutputStream out =
35             new ObjectOutputStream(fout);
36         out.writeObject(t);
37         out.close();
38         /*
39          * Zur Kontrolle :
40          * Wiedereinlesen und Vergleichen
41          */
42         FileInputStream fin =
43             new FileInputStream("tbuch.ser");
44         ObjectInputStream in =
45             new ObjectInputStream(fin);
46         TelefonBuch copy =
47             (TelefonBuch) in.readObject();
48         in.close();
49         if (t.gleichheit(copy))
50         {
51             System.out.println(
52                 "OK---Objekte_sind_gleich!");
53         } else
54         {
55             System.out.println(
56                 "Fehler---Objekte_sind_ungleich!");
57         }
58     } catch (Exception e)
59     {
60         e.printStackTrace(System.out);
61     }
62 }

```

Listing A.51: TelefonBuch

```

/**
2  * Einfaches permanentes Telefonbuch
3  * mit Schreibtest
4  *
5  * @since 29-Jun-1998, ... , 18-Jul-2007
6  * @author Hinrich E. G. Bonin

```

```

    * @version    1.3
    */
8  package de.leuphana.ics.telefon;
10
11  import java.io.Serializable;
12  import java.util.Enumeration;
13  import java.util.Hashtable;
14
15  public class TelefonBuch implements Serializable
16  {
17      Hashtable<String, TelefonEintrag> tabelle;
18
19      public TelefonBuch()
20      {
21          tabelle = new Hashtable<String, TelefonEintrag>();
22      }
23
24      public TelefonEintrag getEintrag(String key)
25      {
26          return (TelefonEintrag) tabelle.get(key);
27      }
28
29      public TelefonEintrag addEintrag(
30          String key,
31          TelefonEintrag te)
32      {
33          return (TelefonEintrag) tabelle.put(key, te);
34      }
35
36      public int size()
37      {
38          return tabelle.size();
39      }
40
41      public boolean gleichheit(TelefonBuch t)
42      {
43          if ((t == null) || (size() != t.size()))
44          {
45              return false;
46          }
47          Enumeration keys = tabelle.keys();
48          while (keys.hasMoreElements())
49          {
50              String key = (String) keys.nextElement();
51              TelefonEintrag myTe = getEintrag(key);
52              TelefonEintrag otherTe = t.getEintrag(key);
53              if (!myTe.gleichheit(otherTe))
54              {
55                  return false;
56              }
57          }
58      }
59  }

```

```

56         }
57     }
58     return true;
59 }
60 }

```

Listing A.52: TelefonEintrag

```

/**
2  * Einfaches permanentes Telefonbuch
3  * mit Schreibtest
4  *
5  * @since    29-Jun-1998, 30-May-2007
6  * @author   Hinrich E. G. Bonin
7  * @version  1.2
8  */
9  package de.leuphana.ics.telefon;
10
11  import java.io.Serializable;
12
13  public class TelefonEintrag implements Serializable
14  {
15
16     private String kurzname;
17     private String telefon;
18
19     public String getKurzname()
20     {
21         return kurzname;
22     }
23
24     public String getTelefon()
25     {
26         return telefon;
27     }
28
29     public TelefonEintrag(
30         String kurzname,
31         String telefon)
32     {
33         if ((kurzname == null) || (telefon == null))
34         {
35             throw new IllegalArgumentException();
36         }
37         this.kurzname = kurzname;
38         this.telefon = telefon;
39         System.out.println("TelefonEintrag: " +
40             kurzname + " " +
41             telefon);

```

```
42     }
44     public boolean gleichheit (TelefonEintrag te)
45     {
46         return
47             (getKurzname (). equalsIgnoreCase (
48                 te .getKurzname ())) &&
49             (getTelefon (). equalsIgnoreCase (
50                 getTelefon ()));
51     }
52 }
```

A.25.1 Ergebnis von java TelefonBuchProg angeben

Geben Sie das Ergebnis des folgenden Aufrufs an:

```
>java de.leuphana.ics.telefon.TelefonBuchProg
```

A.25.2 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.

```
>java de.leuphana.ics.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.26 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.leuphana.ics.lingo`
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.26.1 Objektbeziehungen in Java™ abbilden

Bilden Sie die obigen Aussagen in Java-Quellcode ab.

A.26.2 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.27 Paket mit Klassen- & Interface-Dateien notieren

Nach einer eingehenden Systemanalyse ergeben sich folgenden Aussagen:

1. Alle Klassen und Interfaces gehören zum Paket `de.leuphana.ics.mix`.

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.27.1 Aussagen als Klassendiagramm in UML-Notation abbilden

Bilden Sie die obigen Aussagen als ein Klassendiagramm in UML-Notation ab.

A.27.2 Aussagen in Java-Quellcode abbilden

Bilden Sie die obigen Aussagen in Java-Quellcode ab.

A.27.3 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.28 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.

Listing A.53: 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>

```

Listing A.54: myPageStyle.css

```

1 /* Cascading Style Sheet: myStyle.css */
2 /* Bonin 30-Jun-1998 ... 24-Dec-2002 */
p {

```

```

4   font-size: 12pt;
   color: red;
6   background: white;
   }
8   h1 em {
   font-size: 28pt;
10  }
   h1 {
12  font-size: 14pt;
   color: white;
14  background: blue;
   }
16  em {
   color: green;
18  background: white;
   font-style: italic;
20  }

```

[Hinweis: Die Zeilennummern sind nicht Bestandteil der HTML-Datei und auch nicht der CSS-Datei.]

A.28.1 Header-Konstrukt interpretieren

Beschreiben Sie die Hauptüberschrift (<h1>-Konstrukt), wenn diese von einem Web-Browser angezeigt wird.

A.28.2 Hervorhebungsspezifikation

Erläutern Sie, warum in der CSS-Datei einerseits `h1 em { ... }` und andererseits `em { ... }` angegeben sind.

A.29 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.

Listing A.55: `myFINAL.html`

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
4  <!-- Bonin: 21-Jan-1995 -->
   <!-- Update: ... 26-Jul-2007 -->
6  <head>

```

```

<title >FINAL, 5(8), 1998 </title >
8 <link href="myFStyle.css"
  rel="stylesheet" type="text/css" />
10 <style type="text/css">
  h1 {
12     color:      black;
     background: yellow;
14  }
  </style>
16 </head>
  <body>
18 <h1>FINAL, 5(8), 1998</h1>
  <ul>
20 <li><em>F</em>orum</li>
  <li><em>In</em>formatics</li>
22 <li><em>A</em>t</li>
  <li><em>L</em>euphana</li>
24 </ul>
  <h>Persistente Objekte ----
26   Der <em>Elchtest</em> f&uuml;r ein Java-Programm</h1>
  <p>FINAL, 8. Jahrgang, Heft 5, Dezember 1998,
28   ISSN 0939-8821<br />
  Beziehbar: Leuphana Universit&auml;t L&uuml;neburg,
30   ICS, Volgershall 1,
  D-21339 L&uuml;neburg</p>
32 </body>
  </html>

```

Listing A.56: myFStyle.css

```

/* Cascading Style Sheet: myStyle.css */
2 /* Bonin 21-Jan-1999 ... 25-Dec-2002 */
  p {
4   font-size: 12pt;
     color: red;
6   background: white;
  }
8  h1 em {
     font-size: 28pt;
10 }
  h1 {
12  font-size: 14pt;
     color: white;
14  background: blue;
  }
16 em {
     color: green;
18  background: white;
     font-style: italic;

```

20 }

[Hinweis: Die Zeilennummern sind nicht Bestandteil der HTML-Datei und auch nicht der CSS-Datei.]

A.29.1 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.29.2 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.29.3 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.30 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[]);
}
>

```

Listing A.57: Hund

```

/**
2  * Beispiel einer Rekursion innerhalb der Klasse: Vater und
  * Mutter sind wieder vom Typ Hund
4  *
  * @author      Hinrich Bonin
6  * @version    1.0
  * @since      22-Jan-1999
8  */
import java.util.*;

10
class Hund
12 {
    private String name = "";
14    public boolean weiblich = true;
    private Hund mutter;
16    private Hund vater;

18
    public Hund(String name, boolean weiblich)
20    {
        this.name = name;
22        this.weiblich = weiblich;
        System.out.println(name + " lebt!");
24    }

26
    public String getName()
28    {
        return name;
30    }

```

```
32     public Hund getMutter ()
34     {
36         return mutter;
38     }
39
40     public Hund setMutter(Hund mutter)
41     {
42         this.mutter = mutter;
43         return this;
44     }
45
46     public Hund getVater ()
47     {
48         return vater;
49     }
50
51     public Hund setVater(Hund vater)
52     {
53         this.vater = vater;
54         return this;
55     }
56
57     public static void main(String [] argv)
58     {
59         System.out.println (
60             (new Hund("Bello_von_der_Eulenburg", false))
61             .setMutter(new Hund("Berta_vom_Lechgraben", true))
62             .setVater(new Hund("Alex_vom_Hirschgarten", false))
63             .getMutter().name);
64     }
65 }
```

A.30.1 Beurteilung von Hund

Ist die Applikation Hund entsprechend dem üblichen Java-Standard programmiert? Wenn nicht, beschreiben Sie kurz die Abweichungen.

A.30.2 Ergebnis von Hund

Geben Sie das Ergebnis von `java Hund` an.

A.30.3 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.31 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

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/sideeffect/CProg.java

D:\bonin\anwd\code>java
  de.leuphana.ics.sideeffect.CProg
Object o1 slot value: C2
Object o2 slot value: C2

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

Listing A.58: C1

```
/**
2  * Example "Side Effect"
  *
4  * @since    01-Feb-2002, 30-May-2007
  * @author   Hinrich E. G. Bonin
6  * @version  1.1
  */
8  package de.leuphana.ics.sideeffect;

10 public class C1
  {
12     private String slot = "C1";

14     public String getSlot()
  {
16         return slot;
  }
}
```

```
18     public void setSlot(String slot)
20     {
21         this.slot = slot;
22     }
23
24     public void m1(C2 o)
25     {
26         this.slot = o.getSlot();
27     }
28
29     public void m2(C2 o)
30     {
31         o.setSlot(this.getSlot());
32     }
33 }
34 }
```

Listing A.59: C2

```
/**
2  * Example "Side Effect"
3  *
4  * @since    01-Feb-2002, 30-May-2007
5  * @author   Hinrich E. G. Bonin
6  * @version  1.1
7  */
8  package de.leuphana.ics.sideeffect;
9
10 public class C2
11 {
12     private String slot = "C2";
13
14     public String getSlot()
15     {
16         return slot;
17     }
18
19     public void setSlot(String slot)
20     {
21         this.slot = slot;
22     }
23
24     public void m1(C1 o)
25     {
26         this.slot = o.getSlot();
27     }
28
29     public void m2(C1 o)
```



```

30     {
31         o.setSlot(this.getSlot());
32     }
33 }

```

Listing A.60: CProg

```

/**
2  * Example "Side Effect"
3  *
4  * @since    01-Feb-2002, 30-May-2007
5  * @author   Hinrich E. G. Bonin
6  * @version  1.1
7  */
8  package de.leuphana.ics.sideeffect;

9
10 public class CProg
11 {
12     public static void main(String [] args)
13     {
14         C1 o1 = new C1();
15         C2 o2 = new C2();
16         o1.m1(o2);
17         o1.setSlot("OK?");
18         o2.m2(o1);
19
20         System.out.println(
21             "Object_o1_slot_value:_" +
22             o1.getSlot());
23         System.out.println(
24             "Object_o2_slot_value:_" +
25             o2.getSlot());
26     }
27 }

```

A.32 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.32.1 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.32.2 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.32.3 XML-Datei visualisieren

Erstellen Sie ein Programm, das Ihre XML-Daten in Form einer Tabelle anzeigt.

Java-Coach

Anhang B

Lösungen zu den Übungen

Lösung Aufgabe A.1 S. 405:

A.1.1:

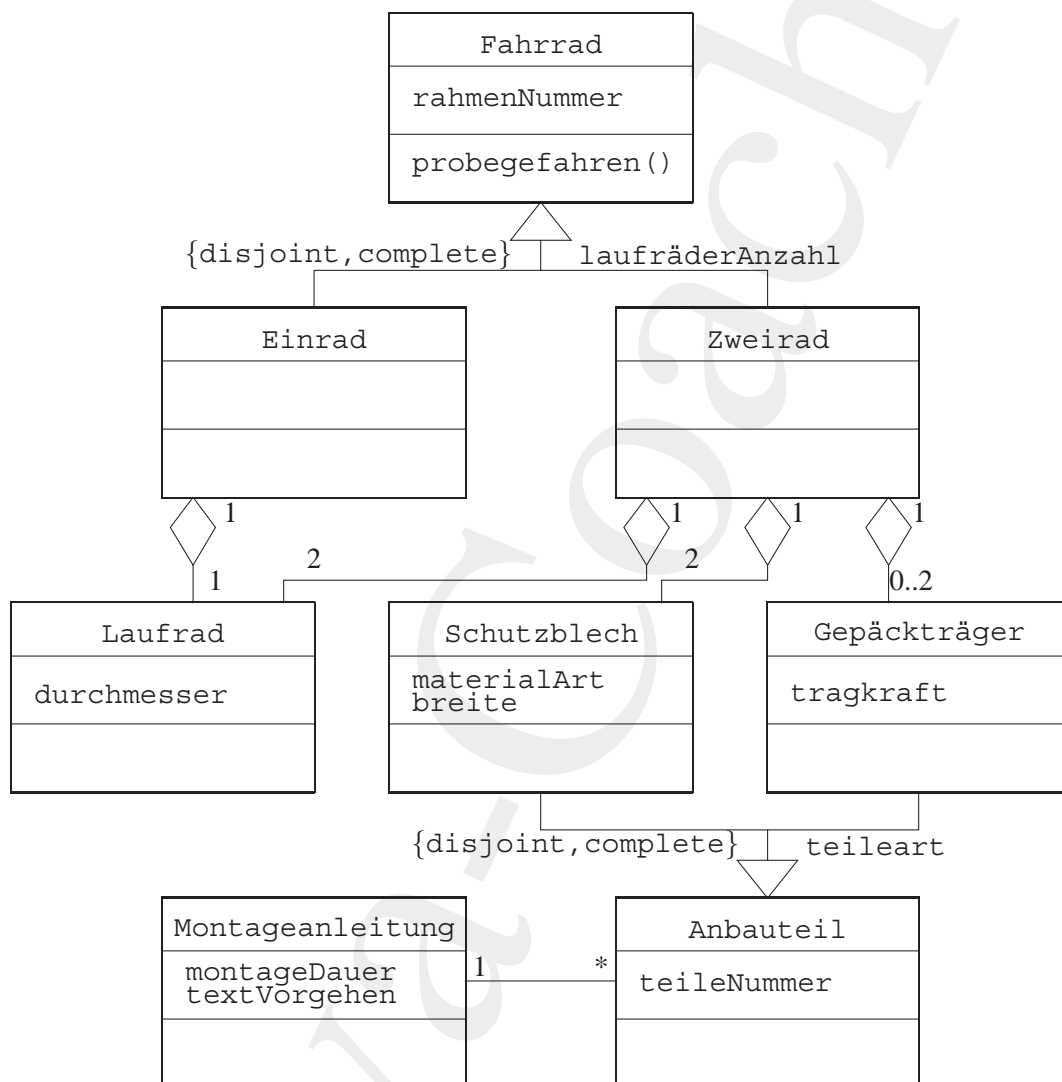
Die Abbildung B.1 S. 494 zeigt das Klassendiagramm für die RVE.

Eine Abbildung dieser Klassen und ihrer Verknüpfungen ist im folgenden angegeben. Um die Hierarchie der Konstruktoren zu zeigen wurde eine zusätzliche Klassenvariable `anzahl` eingeführt.

Listing B.1: Fahrrad

```
/**
2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
  * <h2>Fahrrad: abstrakte Klasse fuer
4  *   Einrad und Zweirad</h2>
  *
6  * @since      30-Mar-2001, 02-May-2007
  * @author      Bonin , Hinrich E.G.
8  * @version    2.0
  */
10 package de.leuphana.ics.rvegbh;

12 public abstract class Fahrrad
  {
14     /*
      * rahmenNummer identifiziert ein Fahrrad
16     */
      private String rahmenNummer;
18     /*
      * Setzt probegefahren()
20     */
      private boolean probegefahren = false;
22
```



Legende: Händisch erstellt.

Abbildung B.1: Aufgabe A.1.1 S. 406: Klassendiagramm für die Montagesicht

```

24  public String getRahmenNummer()
    {
26      return rahmenNummer;
    }
28
30  public void setRahmenNummer(String rahmenNummer)
    {
32      this.rahmenNummer = rahmenNummer;
    }
34
36  public boolean getProbegefahren()
    {
38      return probegefahren;
    }
40
42  /*
    * probegefahren() applizieren nach der Probefahrt
44  */
    public void probegefahren()
    {
46      probegefahren = true;
48  }
    }

```

Listing B.2: Einrad

```

/**
 2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
    * <h2>Einrad: 1 Laufrad</h2>
 4  *
    * @since      30-Mar-2001, 02-May-2007
 6  * @author      Bonin, Hinrich E.G.
    * @version     2.0
 8  */
    package de.leuphana.ics.rvegbh;
10
    public class Einrad extends Fahrrad
12  {
        private Laufrad laufrad;
14
16      Einrad(String rahmenNummer,
            Laufrad laufrad)
18      {
            this.setRahmenNummer(rahmenNummer);

```

```

20     this.laufrad = laufrad;
    }
22 }

```

Listing B.3: Zweirad

```

/**
2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
  * <h2>Zweirad ist Fahrrad mit 2 Laufrädern,
4  * gegebenenfalls mit 2 Schutzblechen und bis
  * zu 2 Gepaecktraegern</h2>
6  *
  * @since      30-Mar-2001, 02-May-2007
8  * @author    Bonin, Hinrich E.G.
  * @version    2.0
10 */
package de.leuphana.ics.rvegmbh;
12
public class Zweirad extends Fahrrad
14 {
    /**
16     * Laufräder, Schutzbleche und
  * Gepaecktraeger als Arrays
18     * abgebildet
  */
20     private Laufrad laufrad [] =
      new Laufrad [2];
22     private Schutzblech schutzblech [] =
      new Schutzblech [2];
24     private Gepaecktraeger gepaecktraeger [] =
      new Gepaecktraeger [2];
26
28     public static int anzahl = 0;
30
32     public Laufrad getVorderrad ()
    {
      return laufrad [0];
    }
34
36     public Laufrad getHinterrad ()
    {
38       return laufrad [1];
    }
40
42     public Schutzblech getVorderradSchutzblech ()
    {

```



```
44     return schutzblech[0];
45 }
46
47 public Schutzblech getHinterradSchutzblech ()
48 {
49     return schutzblech[1];
50 }
51
52
53 public Gepaecktraeger getVorderradGepaecktraeger ()
54 {
55     return gepaecktraeger[0];
56 }
57
58
59 public Gepaecktraeger getHinterradGepaecktraeger ()
60 {
61     return gepaecktraeger[1];
62 }
63
64
65 public void setVorderradGepaecktraeger (
66     Gepaecktraeger gepaecktraeger)
67 {
68     this.gepaecktraeger[0] = gepaecktraeger;
69 }
70
71
72 public void setHinterradGepaecktraeger (
73     Gepaecktraeger gepaecktraeger)
74 {
75     this.gepaecktraeger[1] = gepaecktraeger;
76 }
77
78
79 public Zweirad ()
80 {
81     anzahl++;
82 }
83
84
85 /**
86  *   Zweirad mit 2 Laufrädern ohne Anbauteile
87  *
88  *   @param rahmenNummer ist der Zweirad-Identifizier
89  *   @param vorderrad ist ein Laufrad
90  *   @param hinterrad ist ein Laufrad
91  */
```

```

94     public Zweirad(String rahmenNummer,
95                    Laufrad vorderrad,
96                    Laufrad hinterrad)
97     {
98         this ();
99         this.setRahmenNummer(rahmenNummer);
100        laufrad[0] = vorderrad;
101        laufrad[1] = hinterrad;
102    }
103
104    /**
105     * Zweirad mit 2 Laufrädern und Anbauteilen
106     *
107     * @param rahmenNummer      ist der
108     *       Zweirad-Identifizier
109     * @param vorderrad         ist ein Laufrad
110     * @param hinterrad        ist ein Laufrad
111     * @param vorderradSchutzblech ist ein Schutzblech
112     * @param hinterradSchutzblech ist ein Schutzblech
113     */
114    public Zweirad(String rahmenNummer,
115                   Laufrad vorderrad,
116                   Laufrad hinterrad,
117                   Schutzblech vorderradSchutzblech,
118                   Schutzblech hinterradSchutzblech)
119    {
120        // Konstruktor mit den äLaufrädern
121        this(rahmenNummer, vorderrad, hinterrad);
122        schutzblech[0] = vorderradSchutzblech;
123        schutzblech[1] = hinterradSchutzblech;
124    }
125 }

```

Listing B.4: Laufrad

```

126 /**
127  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
128  * <h2>Laufrad hat ID und Durchmesser </h2>
129  *
130  * @since      30-Mar-2001, 02-May-2007
131  * @author     Bonin, Hinrich E.G.
132  * @version    2.0
133  */
134 package de.leuphana.ics.rvegmbh;
135
136 public class Laufrad
137 {
138     /*

```

```

14     * laufradID identifiziert ein Laufrad
15     */
16     private String laufradID;
17     private int durchmesser;
18
19
20     public int getDurchmesser ()
21     {
22         return durchmesser;
23     }
24
25
26     Laufrad (String laufradID ,
27             int durchmesser)
28     {
29         this.laufradID = laufradID;
30         this.durchmesser = durchmesser;
31     }
32 }

```

Listing B.5: Schutzblech

```

/**
2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
3  * <h2>Schutzblech ist Anbauteil und hat
4  *   MaterialArt und Breite</h2>
5  *
6  * @since    30-Mar-2001, 02-May-2007
7  * @author   Bonin , Hinrich E.G.
8  * @version  2.0
9  */
10 package de.leuphana.ics.rvegmbh;
11
12 public class Schutzblech extends Anbauteil
13 {
14     private String materialArt;
15     private String breite;
16
17
18     public String getMaterialArt ()
19     {
20         return materialArt;
21     }
22
23
24     public String getBreite ()
25     {
26         return breite;
27     }

```

```

28
30  /**
31   * @param teileNummer      hat jedes Anbauteil
32   * @param materialArt     Description of the Parameter
33   * @param breite          Description of the Parameter
34   * @param montageanleitung Description of the Parameter
35   */
36  Schutzblech(String teileNummer,
37              String materialArt,
38              String breite,
39              Montageanleitung montageanleitung)
40  {
41      this.setTeileNummer(teileNummer);
42      this.setMontageanleitung(montageanleitung);
43      this.materialArt = materialArt;
44      this.breite = breite;
45  }
46 }

```

Listing B.6: Gepaecktraeger

```

/**
2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
3  * <h2>Gepaecktraeger ist Anbauteil und hat
4  *   Tragkraft-Slot</h2>
5  *
6  * @since      30-Mar-2001, 02-May-2007
7  * @author     Bonin, Hinrich E.G.
8  * @version    2.0
9  */
10 package de.leuphana.ics.rvegmbh;

12 public class Gepaecktraeger extends Anbauteil
13 {
14     private int tragkraft;

16     public int getTragkraft()
17     {
18         return tragkraft;
19     }
20 }

22
23 /**
24  * @param teileNummer      hat jedes Anbauteil
25  * @param tragkraft        Description of the Parameter
26  * @param montageanleitung Description of the Parameter
27  */

```

```

28     Gepaecktraeger(String teileNummer,
29                     int tragkraft,
30                     Montageanleitung montageanleitung)
31     {
32         this.setTeileNummer(teileNummer);
33         this.tragkraft = tragkraft;
34         this.setMontageanleitung(montageanleitung);
35     }
36 }

```

Listing B.7: Anbauteil

```

/**
2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
3  * <h2>Anbauteil hat Teilenummer und Assoziation
4  * zu Montageanleitung</h2>
5  *
6  * @since      30-Mar-2001, 02-May-2007
7  * @author     Bonin, Hinrich E.G.
8  * @version    2.0
9  */
10 package de.leuphana.ics.rvegmbh;

12 public abstract class Anbauteil
13 {
14     /*
15     * teileNummer identifiziert ein Anbauteil
16     */
17     private String teileNummer;
18     private Montageanleitung montageanleitung;

20
21     public String getTeileNummer()
22     {
23         return teileNummer;
24     }

26     public void setTeileNummer(String teilenummer)
27     {
28         this.teileNummer = teilenummer;
29     }

32     public Montageanleitung getMontageanleitung()
33     {
34         return montageanleitung;
35     }
36 }

```

```
38     public void setMontageanleitung(  
40         Montageanleitung montageanleitung)  
42     {  
44         this.montageanleitung = montageanleitung;  
44     }  
}
```

Listing B.8: Montageanleitung

```
/**  
2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>  
3  * <h2>Montageanleitung hat MontageDauer  
4  *   und TextVorgehen</h2>  
5  *  
6  * @since      30-Mar-2001, 02-May-2007  
7  * @author     Bonin , Hinrich E.G.  
8  * @version    2.0  
9  */  
10 package de.leuphana.ics.rvegmbh;  
  
12 public class Montageanleitung  
13 {  
14     /*  
15     * Achtung kein Identifier fuer Montageanleitung  
16     */  
17     private int montageDauer;  
18     private String textVorgehen;  
  
20     public int getMontageDauer()  
21     {  
22         return montageDauer;  
23     }  
24  
26     public String getTextVorgehen()  
27     {  
28         return textVorgehen;  
29     }  
30  
32     public Montageanleitung(int montageDauer,  
33                             String textVorgehen)  
34     {  
35         this.montageDauer = montageDauer;  
36         this.textVorgehen = textVorgehen;  
37     }  
38 }  
}
```

Listing B.9: RVEGmbH

```

/**
2  * <h1>RadVertriebsExperten GmbH (RVE)</h1>
  * <h2>RVEGmbH stellt Beispiele bereit.</h2>
4  *
  * @since      30-Mar-2001, 02-May-2007
6  * @author     Bonin , Hinrich E.G.
  * @version    2.0
8  */
package de.leuphana.ics.rvegmbh;

10
public class RVEGmbH
12 {
    /**
14     * Beispiele fuer Einrad und Zweirad
    *
16     * @param args The command line arguments
    */
18     public static void main( String [] args)
    {
20         Einrad einradA = new Einrad(
            "A",
22         new Laufrad( "I" , 26));
        einradA.probegefahren();
24
        Einrad einradB = new Einrad(
26         "B",
            new Laufrad( "II" , 28));
28
        Zweirad zweirad1 = new Zweirad(
30         "1",
            new Laufrad( "III" , 26),
32         new Laufrad( "IV" , 26));

34         Montageanleitung anweisungAllgemein =
            new Montageanleitung (
36             120, "Schraube_von_unten_..." );
        Schutzblech schutzblech1 =
38         new Schutzblech (
            "S1", "Kunststoff", "4_cm",
40         anweisungAllgemein );
        Schutzblech schutzblech2 =
42         new Schutzblech (
            "S2", "Blech", "5_cm",
44         anweisungAllgemein );
        Gepaecktraeger gepaecktraegerX =
46         new Gepaecktraeger (
            "GX", 100, anweisungAllgemein );

```

```

48
50 Zweirad zweirad2 = new Zweirad(
    "2", new Laufrad("V", 28), new Laufrad("VI", 28),
    schutzblech1, schutzblech2);
52 zweirad2.setHinterradGepaecktraeger(
    gepaecktraegerX);
54
56 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 + ".")
74 );
76 }

```

Protokolldatei RVEGmbH.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

```

```

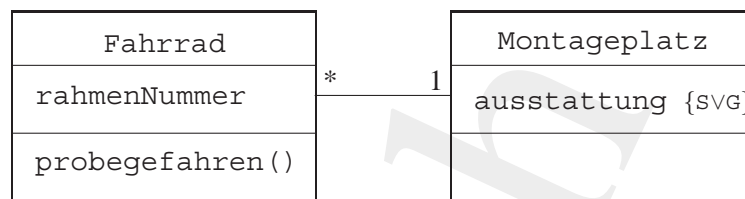
D:\bonin\anwd\code>javac
  de/leuphana/ics/rvegmbh/RVEGmbH.java

```

```

D:\bonin\anwd\code>java
  de.leuphana.ics.rvegmbh.RVEGmbH
RVEGmbH:
Einrad: A probegefahren: true
Einrad: B probegefahren: false
Zweirad: 1 Vorderraddurchmesser: 26

```

Legende: Händisch erstellt.

Abbildung B.2: Aufgabe A.1.2 S. 406: Diagrammerweiterung um den Montageplatz

Zweirad: 2 Schutzblechmontage: Schraube von unten ...
Summe Zweirad = 2.

D:\bonin\anwd\code>

A.1.2:

Die Abbildung B.2 S. 505 zeigt die Diagrammerweiterung um den Montageplatz. Die Abbildung B.3 S. 506 zeigt das Klassendiagramm aus dem Java-Quellcode erzeugt. Genutzt wurde dazu das Werkzeug Enterprise Architect Version 6.5.804 der Firma Sparx Systems Pty Ltd, 7 Curtis Street, Creswick, Victoria, 3363, Australia:
<http://www.sparxsystems.com.au> (online 22-May-2007).

Lösung Aufgabe A.2 S. 406:

A.2.1:

Die Abbildung B.4 S. 507 zeigt den Hauptteil des Klassendiagrammes für die SVI. In Abbildung B.6 S. 528 ist der Aspekt „Zielfernrohr“ abgebildet.

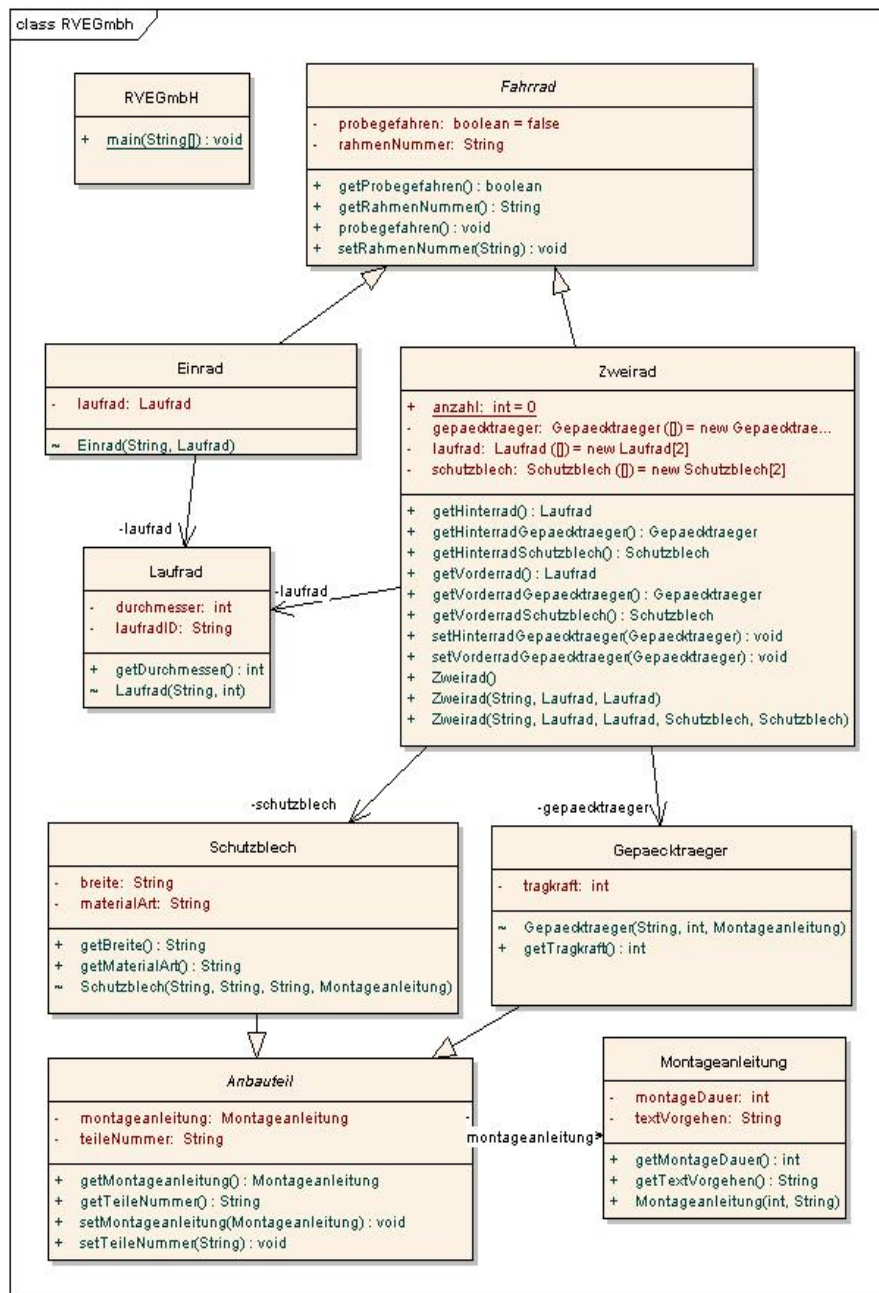
Eine Abbildung dieser Klassen und ihrer Verknüpfungen ist im folgenden angegeben. Dabei sind auch die Restriktionen abgebildet. Zusätzlich wurden Erläuterungen im Quellcode mit HTML-Konstrukten aufbereitet. Das *Application programming interface* (API) wurde mit javadoc erzeugt (↔ Abbildung B.5 S. 527).

javadoc

Listing B.10: SVIProdukt

```

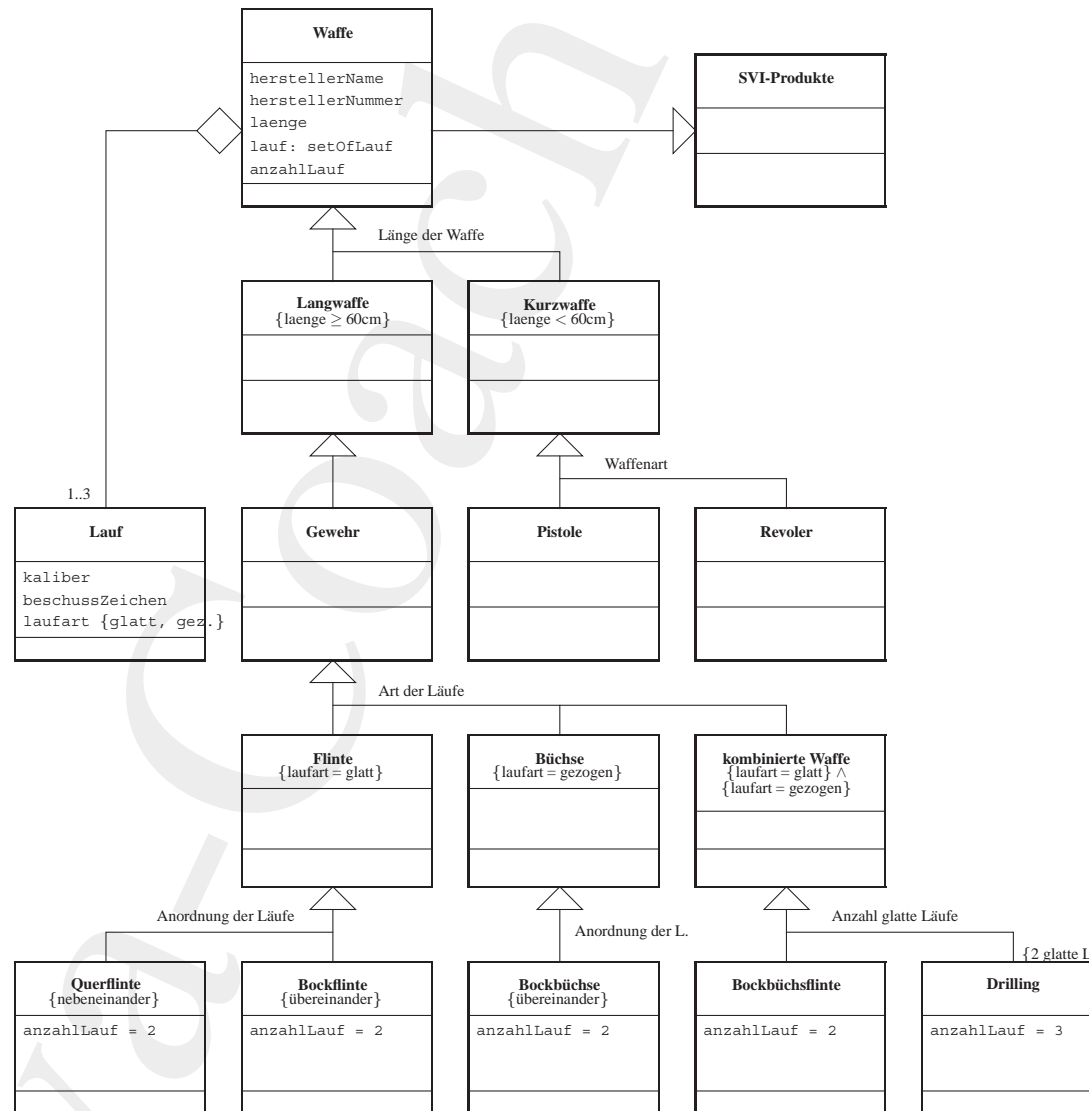
/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>SVI-Produkt: abstracte Klasse üfr Waffe</h2>
4  *
  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
  
```



Legende: Erzeugt aus dem Java-Quellcode mit dem Werkzeug Enterprise Architect Version 6.5.804 der Firma Sparx Systems Pty Ltd:

<http://www.sparxsystems.com.au> (online 22-May-2007).

Abbildung B.3: Aufgabe A.1.2 S. 406: Klassendiagramm erzeugt aus dem Java-Quellcode



Legende: Aus Platzgründen ist der Paketname `de.leuphana.ics.waffenvertrieb` nicht angegeben.

Abbildung B.4: Aufgabe A.2.1 S.408: SVI-Klassendiagramm „Hauptteil“

```

6  * @author    Hinrich E. G. Bonin
   * @version   1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
public abstract class SVIProdukt
12 {
   }

```

Listing B.11: Waffe

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
   * <h2>Waffe hat Herstellerangaben und bis zu 3 Laufen
4  * </h2>
   *
6  * @since     08-Apr-2001, 26-Nov-2002, 03-Jun-2007
   * @author    Hinrich E. G. Bonin
8  * @version   1.3
   */
10 package de.leuphana.ics.waffenvertrieb;

12 public abstract class Waffe extends SVIProdukt
   {
14     private String herstellerName;
       private String herstellerNummer;
16     private int laenge;
       private Lauf lauf[] = new Lauf[3];
18     private int anzahlLauf;

20     public String getHerstellerName ()
       {
22         return herstellerName;
       }

24     public void setHerstellerName(
26         String herstellerName)
       {
28         this.herstellerName = herstellerName;
       }

30     public String getHerstellerNummer()
       {
32         return herstellerNummer;
34     }

36     public void setHerstellerNummer(
38         String herstellerNummer)
       {

```

```

        this.herstellerNummer = herstellerNummer;
40    }
42    public int getLaenge()
    {
44        return laenge;
    }
46    public void setLaenge(int laenge)
48    {
        this.laenge = laenge;
50    }
52    public Lauf getLauf(int key)
    {
54        return lauf[key];
    }
56    public void setLauf(int key, Lauf lauf)
58    {
        this.lauf[key] = lauf;
60    }
62    public int getAnzahlLauf()
    {
64        return anzahlLauf;
    }
66    public void setAnzahlLauf(int anzahlLauf)
68    {
        this.anzahlLauf = anzahlLauf;
70    }
}

```

Listing B.12: Lauf

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>Lauf hat Kaliber, Beschusszeichen und Laufart</h2>
4  *
  * @since    08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author   Hinrich E. G. Bonin
  * @version  1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
public class Lauf
12 {
    private String kaliber;

```

```
14  private String beschussZeichen;  
    private String laufart;  
16  
    public String getKaliber()  
18  {  
        return kaliber;  
20  }  
22  public void setKaliber(String kaliber)  
    {  
24      this.kaliber = kaliber;  
    }  
26  public String getBeschussZeichen()  
28  {  
        return beschussZeichen;  
30  }  
32  public void setBeschussZeichen(String beschussZeichen)  
    {  
34      this.beschussZeichen = beschussZeichen;  
    }  
36  public String getLaufart()  
38  {  
        return laufart;  
40  }  
42  public void setLaufart(String laufart)  
    {  
44      this.laufart = laufart;  
    }  
46  public Lauf(String kaliber ,  
48              String beschussZeichen ,  
              String laufart)  
50  {  
        this.kaliber = kaliber;  
52        this.beschussZeichen = beschussZeichen;  
        if (!((laufart == "glatt") ||  
54            (laufart == "gezogen")))  
            {  
56                laufart = "Nicht OK!";  
            }  
58        this.laufart = laufart;  
60  }  
}
```

Listing B.13: Langwaffe

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>Langwaffe ist laenger als 60 cm </h2>
4  *
  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;

10
public abstract class Langwaffe extends Waffe
12 {
    public static int laengeMinimum()
14     {
        return 60;
16     }
}

```

Listing B.14: Kurzwaffe

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>Kurzwaffe ist unter 60 cm lang und hat nur einen
4  * Lauf</h2>
  *
6  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
  * @author    Hinrich E. G. Bonin
8  * @version    1.3
  */
10 package de.leuphana.ics.waffenvertrieb;

12 public abstract class Kurzwaffe extends Waffe
  {
14     /*
      * Kurzwaffe hat nur einen Lauf, daher ueberschreiben der
16     * Methoden aus Waffe
      */
18     public Lauf getLauf()
      {
20         return this.getLauf(0);
      }

22     public void setLauf(Lauf lauf)
24     {
        this.setLauf(0, lauf);
26     }

28     public static int laengeMaximum()

```

```

30     {
        return 60;
32     }

```

Listing B.15: Gewehr

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>Gewehr ist eine Langwaffe</h2>
4  *
  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
public abstract class Gewehr extends Langwaffe
12 {
}

```

Listing B.16: Pistole

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>Pistole ist eine Kurzwaffe</h2>
4  *
  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
public class Pistole extends Kurzwaffe
12 {
    public static String waffenart()
14     {
        return
16         "Lauf- und Patronenlager bilden eine Einheit."
          + "\nLadevorgang erfolgt durch Schlitten.";
18     }

    Pistole(Lauf lauf)
20     {
22         this.setLauf(0, lauf);
    }

24
    public Pistole(String herstellerName,
26                    String herstellerNummer,

```



```

28         int laenge ,
           Lauf lauf)
29     {
30         this(lauf);
31         this.setHerstellerName(herstellerName);
32         this.setHerstellerNummer(herstellerNummer);
33         if (laenge > Kurzwaffe.laengeMaximum())
34             {
35                 this.setLaenge(Kurzwaffe.laengeMaximum());
36             } else
37             {
38                 this.setLaenge(laenge);
39             }
40     }
}

```

Listing B.17: Revolver

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
3  * <h2>Revolver ist eine Kurzwaffe</h2>
4  *
5  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.3
8  */
9
10 package de.leuphana.ics.waffenvertrieb;
11
12 public class Revolver extends Kurzwaffe
13 {
14     public static String waffenart()
15     {
16         return
17             "Lauf_und_Patronenlager_sind_getrennt."
18             + "\n_Ladevorgang_erfolgt"
19             + "_durch_Drehung_der_Trommel.";
20     }
21
22     Revolver(Lauf lauf)
23     {
24         this.setLauf(0, lauf);
25     }
26
27     public Revolver(String herstellerName ,
28                     String herstellerNummer,
29                     int laenge ,
30                     Lauf lauf)
31     {

```

```

32     this(lauf);
33     this.setHerstellerName(herstellerName);
34     this.setHerstellerNummer(herstellerNummer);
35     if (laenge > Kurzwaffe.laengeMaximum())
36     {
37         this.setLaenge(Kurzwaffe.laengeMaximum());
38     } else
39     {
40         this.setLaenge(laenge);
41     }
42 }

```

Listing B.18: Flinte

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
3  * <h2>Flinte hat Anordnung glatte Laefe</h2>
4  *
5  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
11 public abstract class Flinte extends Gewehr
12 {
13     public static String laufart()
14     {
15         return "glatt";
16     }
17 }

```

Listing B.19: Buechse

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
3  * <h2>Buechse hat gezogene Laeufe</h2>
4  *
5  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
11 public abstract class Buechse extends Gewehr
12 {
13     public static String laufart()
14     {

```

```

    return "gezogen";
16 }
}

```

Listing B.20: KombinierteWaffe

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>KombinierteWaffe hat unterschiedliche Laufarten</h2>
4  *
  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author     Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
public abstract class KombinierteWaffe extends Gewehr
12 {
    public static String laufartGlatt()
14     {
        return "glatt";
16     }

    public static String laufartGezogen()
18     {
        return "gezogen";
20     }
22 }

```

Listing B.21: Querflinte

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>Querflinte hat nebeneinanderliegende Laeufe</h2>
4  *
  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author     Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;
10
public class Querflinte extends Flinte
12 {
    public static String anordnungLaeufe()
14     {
        return "nebeneinander";
16     }

18     Querflinte(Lauf laufLinks , Lauf laufRechts)

```

```

20     {
21         if (laufLinks.getLaufart() != Flinte.laufart())
22             {
23                 laufLinks.setLaufart(
24                     "Falscher_Flintenlaufart_links!");
25             }
26         this.setLauf(0, laufLinks);
27
28         if (laufRechts.getLaufart() != Flinte.laufart())
29             {
30                 laufRechts.setLaufart(
31                     "Falscher_Flintenlaufart_rechts!");
32             }
33         this.setLauf(1, laufRechts);
34
35         this.setAnzahlLauf(2);
36     }
37
38     public Querflinte(String herstellerName,
39                       String herstellerNummer,
40                       int laenge,
41                       Lauf laufLinks,
42                       Lauf laufRechts)
43     {
44         this(laufLinks, laufRechts);
45         this.setHerstellerName(herstellerName);
46         this.setHerstellerNummer(herstellerNummer);
47         if (laenge < Langwaffe.laengeMinimum())
48             {
49                 this.setLaenge(Langwaffe.laengeMinimum());
50             } else
51             {
52                 this.setLaenge(laenge);
53             }
54     }

```

Listing B.22: Bockflinte

```

10  /**
11   * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
12   * <h2>Bockflinte hat Anordnung der Laefe</h2>
13   *
14   * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
15   * @author     Hinrich E. G. Bonin
16   * @version    1.3
17   */
18  package de.leuphana.ics.waffenvertrieb;

```

```

12  public class Bockflinte extends Flinte
13  {
14      public static String anordnungLaefue ()
15      {
16          return "uebereinander";
17      }
18
19      Bockflinte(Lauf laufOben , Lauf laufUnten)
20      {
21          if (laufOben.getLaufart() != Flinte.laufart())
22          {
23              laufOben.setLaufart(
24                  "Falscher_Flintenlauftyp_oben!");
25          }
26          this.setLauf(0, laufOben);
27
28          if (laufUnten.getLaufart() != Flinte.laufart())
29          {
30              laufUnten.setLaufart(
31                  "Falscher_Flintenlauftyp_unten!");
32          }
33          this.setLauf(1, laufUnten);
34
35          this.setAnzahlLauf(2);
36      }
37
38      public Bockflinte(String herstellerName ,
39                      String herstellerNummer ,
40                      int laenge ,
41                      Lauf laufOben ,
42                      Lauf laufUnten)
43      {
44          this(laufOben , laufUnten);
45          this.setHerstellerName(herstellerName);
46          this.setHerstellerNummer(herstellerNummer);
47          if (laenge < Langwaffe.laengeMinimum())
48          {
49              this.setLaenge(Langwaffe.laengeMinimum());
50          } else
51          {
52              this.setLaenge(laenge);
53          }
54      }

```

Listing B.23: Bockbuechse

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>

```

```
* <h2>Bockbuechse hat Anordnung der Laefe</h2>
4 *
* @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6 * @author    Hinrich E. G. Bonin
* @version    1.3
8 */
package de.leuphana.ics.waffenvertrieb;
10
public class Bockbuechse extends Buechse
12 {
    public static String anordnungLaefe ()
14     {
        return "uebereinander";
16     }

    Bockbuechse(Lauf laufOben , Lauf laufUnten)
18     {
20         if (laufOben.getLaufart () != Buechse.laufart ())
                {
22             laufOben.setLaufart (
                    "Falscher_Buechsenlauftyp_oben!");
24         }
        this.setLauf (0 , laufOben);
26
        if (laufUnten.getLaufart () != Buechse.laufart ())
28         {
                laufUnten.setLaufart (
30                 "Falscher_Buechsenlauftyp_unten!");
                }
32         this.setLauf (1 , laufUnten);
34
        this.setAnzahlLauf (2);
        }
36
    public Bockbuechse(String herstellerName ,
38                        String herstellerNummer ,
                        int laenge ,
40                        Lauf laufOben ,
                        Lauf laufUnten)
42     {
        this(laufOben , laufUnten);
44         this.setHerstellerName(herstellerName);
        this.setHerstellerNummer(herstellerNummer);
46         if (laenge < Langwaffe.laengeMinimum ())
            {
48             this.setLaenge (Langwaffe.laengeMinimum ());
            } else
50         {
            this.setLaenge (laenge);
        }
    }
}
```

```

52     }
54 }

```

Listing B.24: Bockbuechsflinte

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
  * <h2>Bockbuechse hat Anzahll der glatten Laeufe</h2>
4  *
  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.3
8  */
package de.leuphana.ics.waffenvertrieb;

10 public class Bockbuechsflinte extends KombinierteWaffe
12 {
    public static int anzahlGlatteLaeufe ()
14     {
        return 1;
16     }

    Bockbuechsflinte(Lauf laufI , Lauf laufII)
18     {
20         if (!
                ((( laufI .getLaufart() ==
22                 KombinierteWaffe .laufartGlatt ())
                  && ( laufII .getLaufart() ==
24                 KombinierteWaffe .laufartGezogen ()))
                ||
26                 (( laufI .getLaufart() ==
                KombinierteWaffe .laufartGezogen ())
28                 && ( laufII .getLaufart() ==
                KombinierteWaffe .laufartGlatt ()))
30             )
32         {
            laufI .setLaufart (
34                 "Falsche_Laufkombination!");
            laufII .setLaufart (
36                 "Falsche_Laufkombination!");
        }
38         this .setLauf (0 , laufI);
        this .setLauf (1 , laufII);
40
        this .setAnzahlLauf (2);
42     }

```

```

44     public Bockbuechsflinte(String herstellerName ,
46                             String herstellerNummer ,
48                             int laenge ,
50                             Lauf laufI ,
52                             Lauf laufII)
54     {
56         this(laufI , laufII);
58         this.setHerstellerName(herstellerName);
60         this.setHerstellerNummer(herstellerNummer);
62         if (laenge < Langwaffe.laengeMinimum())
        {
            this.setLaenge(Langwaffe.laengeMinimum());
        } else
        {
            this.setLaenge(laenge);
        }
    }
}

```

Listing B.25: Drilling

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
3  * <h2>Drilling hat zwei glatte Laeufe</h2>
4  *
5  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.3
8  */
9  package de.leuphana.ics.waffenvertrieb;
10
11  public class Drilling extends KombinierteWaffe
12  {
13      public static int anzahlGlatteLaeufe()
14      {
15          return 2;
16      }
17
18      Drilling(Lauf laufI , Lauf laufII , Lauf laufIII)
19      {
20          int iLaufartGlatt = 0;
21          int iLaufartGezogen = 0;
22
23          if (laufI.getLaufart() ==
24              KombinierteWaffe.laufartGlatt())
25          {
26              iLaufartGlatt++;
27          } else

```



```

28         {
30             if (laufI.getLaufart() ==
31                 KombinierteWaffe.laufartGezogen())
32             {
33                 iLaufartGezogen++;
34             } else
35             {
36                 laufI.setLaufart(
37                     "Falsche_Laufart!");
38             }
39         }
40     ;
41     if (laufII.getLaufart() ==
42         KombinierteWaffe.laufartGlatt())
43     {
44         iLaufartGlatt++;
45     } else
46     {
47         if (laufII.getLaufart() ==
48             KombinierteWaffe.laufartGezogen())
49         {
50             iLaufartGezogen++;
51         } else
52         {
53             laufII.setLaufart(
54                 "Falsche_Laufart!");
55         }
56     }
57 ;
58     if (laufIII.getLaufart() ==
59         KombinierteWaffe.laufartGlatt())
60     {
61         iLaufartGlatt++;
62     } else
63     {
64         if (laufIII.getLaufart() ==
65             KombinierteWaffe.laufartGezogen())
66         {
67             iLaufartGezogen++;
68         } else
69         {
70             laufIII.setLaufart(
71                 "Falsche_Laufart!");
72         }
73     }
74 ;
75     if (!((iLaufartGlatt ==
76         Drilling.anzahlGlattLaeufe())
77         && (iLaufartGezogen == 1)))

```

```

78     {
80         laufI.setLaufart(
82             "Falsche_Laufartkombination_beim_Drilling!");
84         laufII.setLaufart(
86             "Falsche_Laufartkombination_beim_Drilling!");
88         laufIII.setLaufart(
90             "Falsche_Laufartkombination_beim_Drilling!");
92     }
94     this.setLauf(0, laufI);
96     this.setLauf(1, laufII);
98     this.setLauf(2, laufIII);
100
102     this.setAnzahlLauf(3);
104 }
106
108 public Drilling(String herstellerName,
110                 String herstellerNummer,
111                 int laenge,
112                 Lauf laufI,
113                 Lauf laufII,
114                 Lauf laufIII)
115 {
116     this(laufI, laufII, laufIII);
117     this.setHerstellerName(herstellerName);
118     this.setHerstellerNummer(herstellerNummer);
119     if (laenge < Langwaffe.laengeMinimum())
120     {
121         this.setLaenge(Langwaffe.laengeMinimum());
122     } else
123     {
124         this.setLaenge(laenge);
125     }
126 }
127 }

```

Listing B.26: SVIGmbH

```

/**
2  * <h1>SportwaffenVertriebInternational GmbH (SVI)</h1>
3  * <h2>SVIGmbH stellt Beispiele bereit.</h2>
4  *
5  * @since      08-Apr-2001, 26-Nov-2002, 03-Jun-2007
6  * @author     Hinrich E. G. Bonin
7  * @version    1.3
8  */
9  package de.leuphana.ics.waffenvertrieb;
10
11 public class SVIGmbH
12 {

```

```

14 public static void main(String [] args)
15 {
16     System.out.println("SVIGmbH:");
17     /*
18      * Beispiel Revolver
19      */
20     Lauf lauf = new Lauf(".45LC",
21                         "Berlin",
22                         "gezogen");
23
24     Revolver coltSAA =
25         new Revolver("Colt", "23163", 14, lauf);
26
27     System.out.println (
28         "\n" +
29         "\nRevolver: " +
30         coltSAA.getHerstellerNummer() +
31         "\nKaliber: " +
32         coltSAA.getLauf().getKaliber() +
33         "\nWaffenart: " +
34         Revolver.waffenart());
35     /*
36      * Beispiel Bockflinte
37      */
38     Lauf laufA = new Lauf("12/70",
39                         "Braunschweig",
40                         Flinte.laufart());
41     Lauf laufB = new Lauf("12/70",
42                         "Braunschweig",
43                         "dubios");
44     Bockflinte browning425 =
45         new Bockflinte("Browning",
46                       "978545",
47                       57,
48                       laufA,
49                       laufB);
50
51     System.out.println (
52         "\n" +
53         "\nBockflinte: " +
54         browning425.getHerstellerNummer() +
55         "\nLaenge: " +
56         browning425.getLaenge() +
57         "\nLaeufe: " +
58         Bockflinte.anordnungLaeufe() +
59         "\nLaufOben: " +
60         browning425.getLauf(0).getLaufart() +
61         "\nLaufUnten: " +
62         browning425.getLauf(1).getLaufart());

```

```

62     /*
63      *   Beispiel Drilling
64      */
65     Lauf laufI = new Lauf("30-06",
66                          "Nuernberg",
67                          Buechse.laufart());
68     Lauf laufII = new Lauf("12/70",
69                           "Nuernberg",
70                           Flinte.laufart());
71     Lauf laufIII = new Lauf("12/70",
72                             "Nuernberg",
73                             Buechse.laufart());
74
75     Drilling merkel =
76         new Drilling("Merkel",
77                     "662345",
78                     63,
79                     laufI,
80                     laufII,
81                     laufIII);
82     System.out.println(
83         "\n" +
84         "\nDrilling:~~~~~" +
85         merkel.getHerstellerNummer() +
86         "\n~Laenge:~~~~~" +
87         merkel.getLaenge() +
88         "\n~Laufanzahl:~" +
89         merkel.getAnzahlLauf() +
90         "\n~~LaufI:~~~~~" +
91         merkel.getLauf(0).getLaufart() +
92         "\n~~LaufII:~~~~~" +
93         merkel.getLauf(1).getLaufart() +
94         "\n~~LaufIII:~~~~~" +
95         merkel.getLauf(2).getLaufart());
96 }

```

Protokolldatei SVIGmbH.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/waffenvertrieb/SVIGmbH.java

D:\bonin\anwd\code>java de.leuphana.ics.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!
```

```
D:\bonin\anwd\code>javadoc de/leuphana/ics/waffenvertrieb/*.java
Loading source file de/leuphana/ics/waffenvertrieb/Bockbuechse.java...
Loading source file de/leuphana/ics/waffenvertrieb/Bockbuechsflinte.java...
Loading source file de/leuphana/ics/waffenvertrieb/Bockflinte.java...
Loading source file de/leuphana/ics/waffenvertrieb/Buechse.java...
Loading source file de/leuphana/ics/waffenvertrieb/Drilling.java...
Loading source file de/leuphana/ics/waffenvertrieb/Flinte.java...
Loading source file de/leuphana/ics/waffenvertrieb/Gewehr.java...
Loading source file de/leuphana/ics/waffenvertrieb/KombinierteWaffe.java...
Loading source file de/leuphana/ics/waffenvertrieb/Kurzwaffe.java...
Loading source file de/leuphana/ics/waffenvertrieb/Langwaffe.java...
Loading source file de/leuphana/ics/waffenvertrieb/Lauf.java...
Loading source file de/leuphana/ics/waffenvertrieb/Pistole.java...
Loading source file de/leuphana/ics/waffenvertrieb/Querflinte.java...
Loading source file de/leuphana/ics/waffenvertrieb/Revolver.java...
Loading source file de/leuphana/ics/waffenvertrieb/SVIGmbH.java...
Loading source file de/leuphana/ics/waffenvertrieb/SVIProdukt.java...
Loading source file de/leuphana/ics/waffenvertrieb/Waffe.java...
Constructing Javadoc information...
Standard Doclet version 1.5.0_08
Building tree for all the packages and classes...
Generating de/leuphana/ics/waffenvertrieb/\Bockbuechse.html...
Generating de/leuphana/ics/waffenvertrieb/\Bockbuechsflinte.html...
Generating de/leuphana/ics/waffenvertrieb/\Bockflinte.html...
Generating de/leuphana/ics/waffenvertrieb/\Buechse.html...
Generating de/leuphana/ics/waffenvertrieb/\Drilling.html...
Generating de/leuphana/ics/waffenvertrieb/\Flinte.html...
Generating de/leuphana/ics/waffenvertrieb/\Gewehr.html...
Generating de/leuphana/ics/waffenvertrieb/\KombinierteWaffe.html...
Generating de/leuphana/ics/waffenvertrieb/\Kurzwaffe.html...
Generating de/leuphana/ics/waffenvertrieb/\Langwaffe.html...
Generating de/leuphana/ics/waffenvertrieb/\Lauf.html...
Generating de/leuphana/ics/waffenvertrieb/\Pistole.html...
Generating de/leuphana/ics/waffenvertrieb/\Querflinte.html...
```

```
Generating de/leuphana/ics/waffenvertrieb/\Revolver.html...
Generating de/leuphana/ics/waffenvertrieb/\SVIGmbH.html...
Generating de/leuphana/ics/waffenvertrieb/\SVIProdukt.html...
Generating de/leuphana/ics/waffenvertrieb/\Waffe.html...
Generating de/leuphana/ics/waffenvertrieb/\package-frame.html...
Generating de/leuphana/ics/waffenvertrieb/\package-summary.html...
Generating de/leuphana/ics/waffenvertrieb/\package-tree.html...
Generating constant-values.html...
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 help-doc.html...
Generating stylesheet.css...
```

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

Flinte - Microsoft Internet Explorer

Datei Bearbeiten Ansicht Favoriten Extras ?

Zurück Suchen Favoriten

Adresse D:\(bonin)anwd\code\index.html Wechseln zu Links

All Classes

- [Bockbuechse](#)
- [Bockbuechsflinte](#)
- [Bockflinte](#)
- [Buechse](#)
- [Drilling](#)
- [Flinte](#)
- [Gewehr](#)
- [KombinierteWaffe](#)
- [Kurzwaaffe](#)
- [Langwaaffe](#)
- [Lauf](#)
- [Pistole](#)
- [Querflinte](#)
- [Revolver](#)
- [SVIGmbH](#)
- [SVIProdukt](#)
- [Waffe](#)

Package Class Tree Deprecated Index Help

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

de.leuphana.ics.waffenvertrieb

Class Flinte

java.lang.Object

- ↳ [de.leuphana.ics.waffenvertrieb.SVIProdukt](#)
 - ↳ [de.leuphana.ics.waffenvertrieb.Waffe](#)
 - ↳ [de.leuphana.ics.waffenvertrieb.Langwaaffe](#)
 - ↳ [de.leuphana.ics.waffenvertrieb.Gewehr](#)
 - ↳ [de.leuphana.ics.waffenvertrieb.Flinte](#)

Direct Known Subclasses:

[Bockflinte](#), [Querflinte](#)

```
public abstract class Flinte
extends Gewehr
```

Constructor Summary

[Flinte](#)()

Method Summary

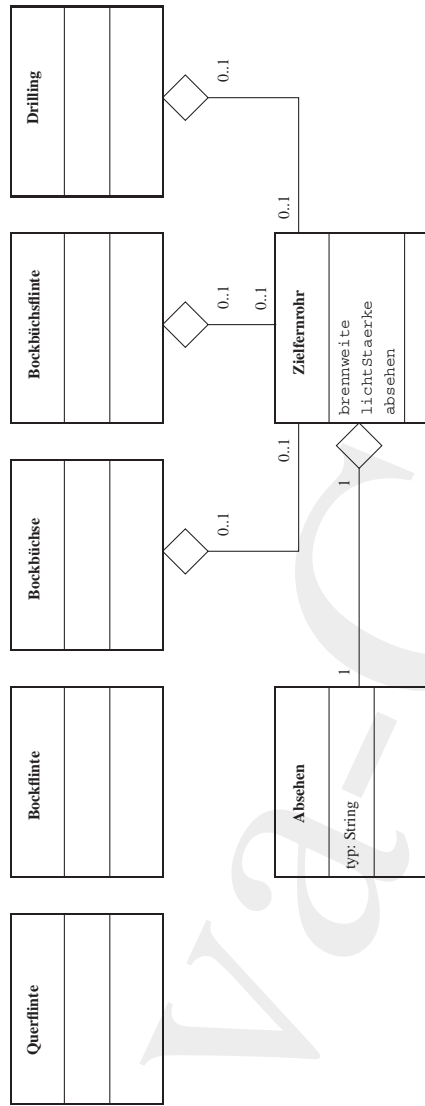
static java.lang.String [laufart](#)()

Arbeitsplatz

Legende:

Quellecode der Klasse Flinte ↔ S.514

Abbildung B.5: API mit javadoc der Klasse Flinte



Legende: Aus Platzgründen ist der Paketname de . leuphana . i . cs . waffenvertrieb nicht angegeben.

Abbildung B.6: Aufgabe A.2.1 S. 408: SVI-Klassendiagramm „Zielfernrohr“

A.2.2:

Die Abbildung B.7 S. 530 zeigt die Diagrammergänzung um den Aspekt „Waffenbesitzkarte“.

Lösung Aufgabe A.3 S. 408:A.3.1:

Listing B.27: echo

```

/**
 2  * Klassenname mit kleinem Anfangsbuchstaben ,
 3  * also echo statt Echo, da in Shells echo
 4  * ueblicherweise in kleinen Buchstaben
 5  * geschrieben wird.
 6  *
 7  * @since      26-Nov-2002, 25-May-2007
 8  * @author     Hinrich E. G. Bonin
 9  * @version    1.1
10  */
11 public class echo
12 {
13     public static void main(String [] args)
14     {
15         for (int i = 0; i < args.length; i++)
16         {
17             System.out.print(args[i] + " ");
18         }
19         System.out.print("\n");
20         System.exit(0);
21     }
22 }

```

A.3.2:

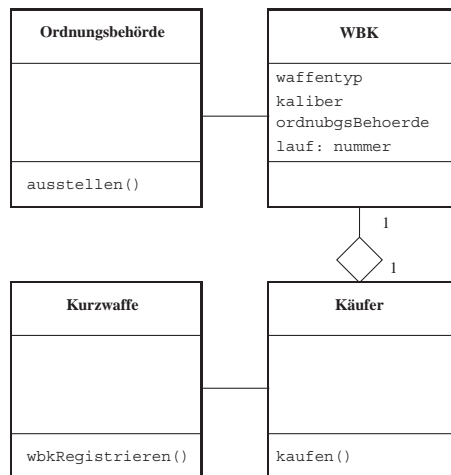
Hinweise: Leerzeichen zwischen den einzelnen Argumenten werden auf ein Leerzeichen reduziert. Kein Rückgabewert — Abhilfe durch Ergänzung mit `System.exit(0)`

Protokolldatei echo.log

```

d:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

```



Legende: Aus Platzgründen ist der Paketname de.leuphana.ics.waffenvertrieb nicht angegeben.

Abbildung B.7: Aufgabe A.2.2 S. 408: SVI-Klassendiagramm „Waffenbesitzkarte“

```

d:\bonin\anwd\code>javac echo.java
d:\bonin\anwd\code>java echo Alles klar?
Alles klar?
d:\bonin\anwd\code>
  
```

Lösung Aufgabe A.4 S. 409:

Protokolldatei Wert.log

```

d:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

d:\bonin\anwd\code>javac
  de/leuphana/ics/gleichheit/Wert.java

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

```
de.leuphana.ics.gleichheit.Wert
1: false
2: false
3: true
4: true
5: false
6: false
7: true

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

Lösung Aufgabe A.5 S. 410:

Protokolldatei Scoping.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/scope/Scoping.java

de/leuphana/ics/scope/Scoping.java:24: cannot find symbol
symbol  : variable j
location: class de.leuphana.ics.scope.Scoping
    System.out.println("j: " + j);
                          ^
1 error

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

Lösung Aufgabe A.6 S. 410:

Protokolldatei Controlling.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac
  de/leuphana/ics/control/Controlling.java

D:\bonin\anwd\code>java
  de.leuphana.ics.control.Controlling ist OK!
i = 6
j = 6
k = 7
m = 3.14159265358979
n = 3
p = Java
maybe = false
important = true
ofCourse = true

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

Lösung Aufgabe A.7 S.412:

Protokolldatei Iteration.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/loop/Iteration.java

D:\bonin\anwd\code>java
  de.leuphana.ics.loop.Iteration 1 2 3 4 5 6 7
Maximum UML & Java in der Anwendungsentwicklung null
Dies sind 53 Zeichen!

D:\bonin\anwd\code>java
  de.leuphana.ics.loop.Iteration 1 2
Exception in thread "main"
  java.lang.ArrayIndexOutOfBoundsException: 2
at de.leuphana.ics.loop.Iteration.main(Iteration.java:26)

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

Lösung Aufgabe A.8 S.413:

Protokolldatei LinieProg.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac
de/leuphana/ics/think/LinieProg.java
```

```
D:\bonin\anwd\code>java
de.leuphana.ics.think.LinieProg
00
34
5.0
12.0
false
```

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

Lösung Aufgabe A.9 S.416:**Protokolldatei Inheritance.log**

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM (
  build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac
de/leuphana/ics/inheritance/Inheritance.java
```

```
D:\bonin\anwd\code>java
de.leuphana.ics.inheritance.Inheritance
Otto AG
I : 9
II: 12106
```

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

Lösung Aufgabe A.10 S.420:

Protokolldatei TableProg.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de/leuphana/ics/table/TableProg.java
.\de\leuphana\ics\table\LookupTable.java:22:
  recursive constructor invocation
    LookupTable(int size)
    ^
1 error

D:\bonin\anwd\code>
  REM Streichung von this() in
  REM LookupTable(int size){...}

D:\bonin\anwd\code>java
de.leuphana.ics.table.TableProg
Tabelle mit 100 erzeugt!
Alles richtig, oder was?

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

Lösung Aufgabe A.12 S. 425:**Protokolldatei Durchschnitt.log**

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de/leuphana/ics/durchschnitt/DurchschnittProg.java

D:\bonin\anwd\code>java
de.leuphana.ics.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.422:

Protokolldatei Rekursion.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac
  de/leuphana/ics/rekursion/Rekursion.java
```

```
D:\bonin\anwd\code>java
  de.leuphana.ics.rekursion.Rekursion
Fakultaetsfunktion: fac(0) = 1
Anzahl der Aufrufe von fac(): 1
```

```
D:\bonin\anwd\code>java
  de.leuphana.ics.rekursion.Rekursion 3
Aufruf  n = 3
Aufruf  n = 2
Aufruf  n = 1
Rueckgabe wert = 1
Rueckgabe wert = 2
Rueckgabe wert = 6
Fakultaetsfunktion: fac(3) = 6
Anzahl der Aufrufe von fac(): 4
```

D:\bonin\anwd\code>

Lösung Aufgabe A.13 S.430:

Protokolldatei FooBar.log

```
D:\bonin\anwd\code>java -version
java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
```

```

Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac de/leuphana/ics/assozi/Foo.java

D:\bonin\anwd\code>dir de\leuphana\ics\assozi\*.class
    1.016 Bar.class
    836 Foo.class

D:\bonin\anwd\code>java de.leuphana.ics.assozi.Bar
Alles durchdacht? Bar!
Alles durchdacht? Foo!

D:\bonin\anwd\code>java de.leuphana.ics.assozi.Foo
Alles durchdacht? Foo!

D:\bonin\anwd\code>

```

Lösung Aufgabe A.14 S. 432:

Protokolldatei SlotI.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/vererbung/SlotI.java

D:\bonin\anwd\code>java
  de.leuphana.ics.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

D:\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 vor-

handen — nur die *Gettern* und *Settern* aus den Definitionskontext greift nicht darauf zu, sondern auf des *i* von *Foo*.

Lösung Aufgabe A.15 S. 434:

Protokolldatei QueueProg.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/queue/QueueProg.java

D:\bonin\anwd\code>java
  de.leuphana.ics.queue.QueueProg
java de.leuphana.ics.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

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

Lösung Aufgabe A.16 S. 438:

Protokolldatei QueueProgII.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/queueII/QueueProg.java
```

```
D:\bonin\anwd\code>java
  de.leuphana.ics.queueII.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
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)
```

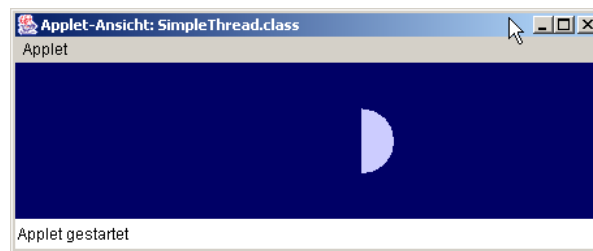


Abbildung B.8: Aufgabe A.17 S. 443: „Animierter Mond“— applet-viewer

```
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

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac -Xlint:deprecation
de/leuphana/ics/awt/DemoAWT.java
de/leuphana/ics/awt/DemoAWT.java:23:
  warning: [deprecation] show() in
  java.awt.Window has been deprecated
    myFrame.show();
           ^
1 warning

D:\bonin\anwd\code>appletviewer ExampleAWT.html
```

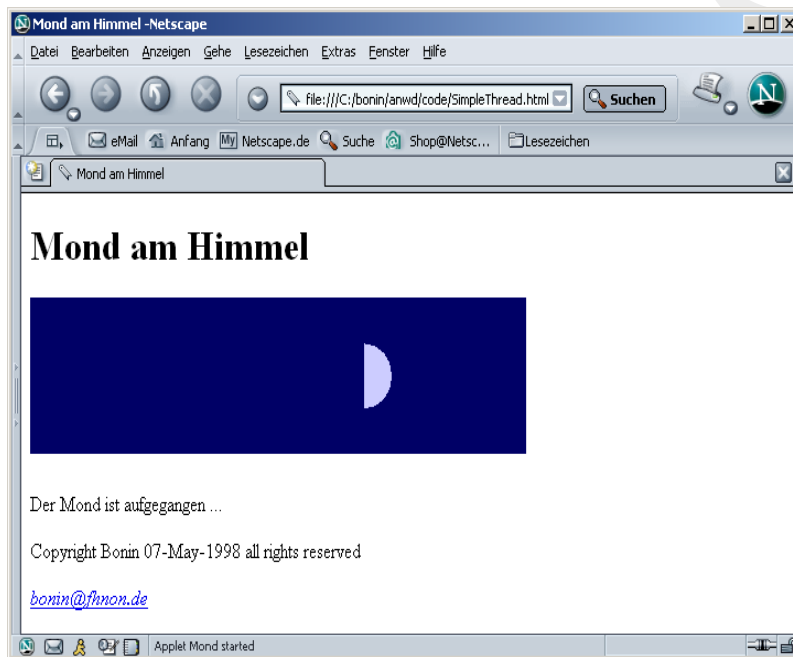


Abbildung B.9: Aufgabe A.17 S.443: „Animierter Mond“ — Netscape 7.0

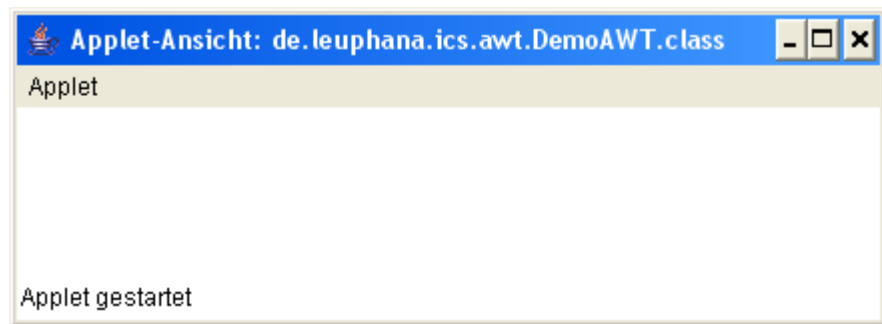


Abbildung B.10: Aufgabe A.18 S. 446: Ausgangsfenster

```
appletviewer ExampleAWT.html
actionPerformed() appliziert: Anmelden!
actionPerformed() appliziert: Absagen!
stop() appliziert!

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

Lösung Aufgabe A.23 S. 463:

Das folgende Session-Protokoll wurde in der Emacs-Shell auf einer NT-Workstation erstellt. Die Abbildung B.12 S. 543 zeigt die FastObjects-Klassen im Werkzeug „POET-Developer“.



Abbildung B.11: Aufgabe A.18 S. 446: Hauptfenster

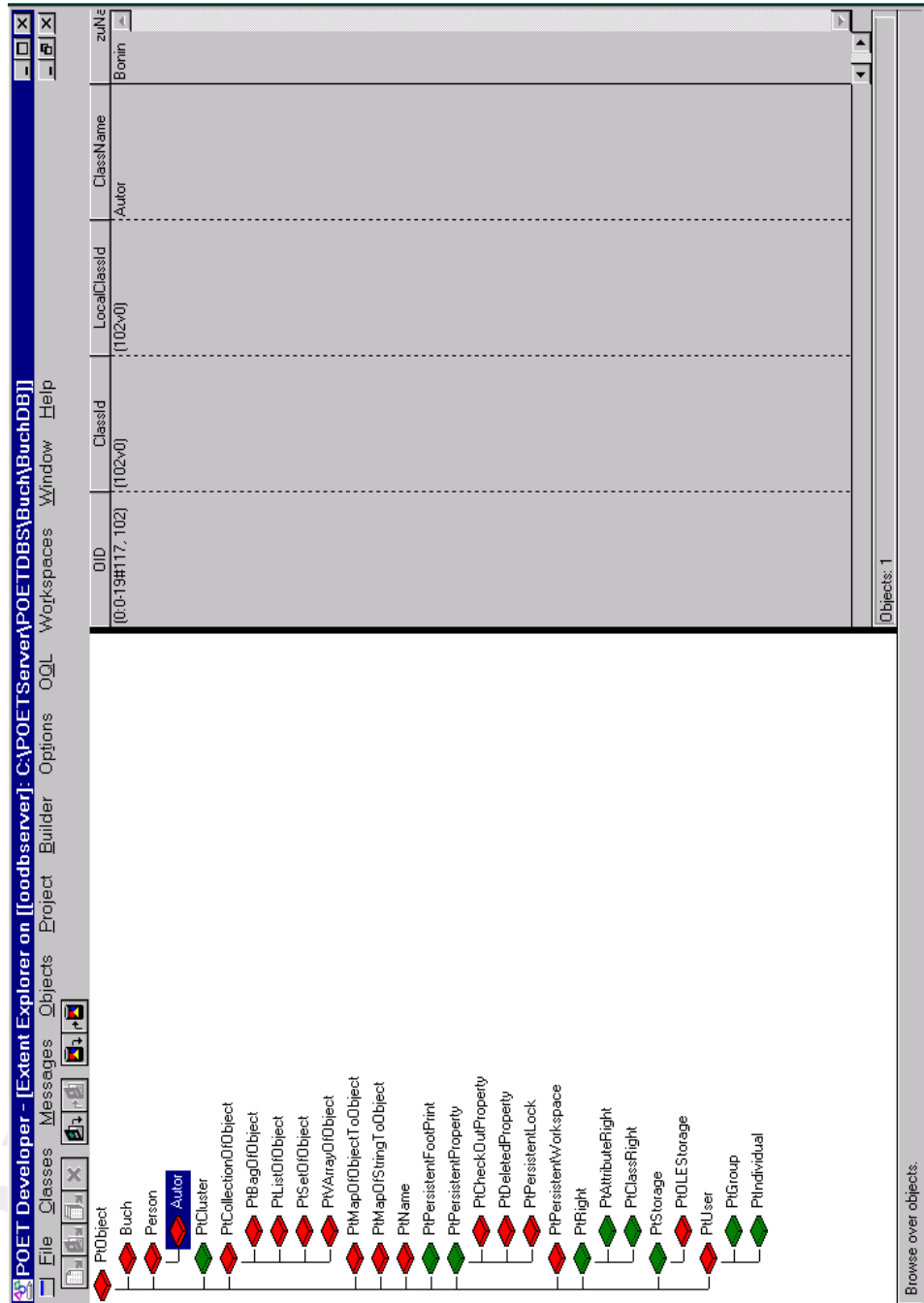


Abbildung B.12: POET Developer: Beispiel Buch

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 Passwort
Zuname des Autors: Bonin
Alter des Buches : 7
preWrite-Methode appliziert!
$ java BuchLookup
---> Hier kommt das POET-Login-Window
      Eingabe von Name und Passwort
Zuname des Autors: Bonin
Alter des Buches : 7
$ java BuchBind
---> Hier kommt das POET-Login-Window
      Eingabe von Name und Passwort
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. 452:

A.19.1:

Der Java-Quellcode läßt sich fehlerfrei compilieren und ausführen. Nach dem Compilieren sind folgende Dateien entstanden:

```

Regal$Application.class
Regal$Schublade.class
Regal.class

```

Der Aufruf ist korrekt.

A.19.2:

Protokolldatei Regal.log

```

D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM

```

```
(build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de/leuphana/ics/regal/Regal.java

D:\bonin\anwd\code>dir de\leuphana\ics\regal\*.class
1.576 Regal$Application.class
 964 Regal$Schublade.class
 883 Regal.class

D:\bonin\anwd\code>java
de.leuphana.ics.regal.Regal$Application
Das Regalsystem hat 2 Regal(e).
Das Regal foo hat 2 Schubladen.
Schubladeninhalte:
  Java-Disketten
  Java-Artikel
Das Regal foo wurde 4x benutzt.

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

Lösung Aufgabe A.20 S. 454:

Die Datei `Ganze.java` hat folgenden Inhalt:

Listing B.28: Ganze

```
/**
2  * Kleines Beispiel fuer main-Methode
  * in Member-class
4  *
  * @author    Bonin
6  * @created  08-May-2007
  * @version  1.0
8  */
package de.leuphana.ics.all;

10
public class Ganze
12 {
    String text = "Gib_Ganze!";
14
    public static void main(String [] args)
16 {
        System.out.println(new Ganze().text);
18 }

20 /* Hinweis: Member-class muss static sein, damit
  main-Methode appliziert werden kann.
22 Also: java Ganze$Teil
```

```
24     */  
25     static class Teil  
26     {  
27         String text = "Gib Teil!";  
28         public static void main(String [] args)  
29         {  
30             System.out.println(new Teil().text);  
31         }  
32     }  
}
```

Lösung Aufgabe A.21 S. 455:

Protokolldatei Anonym.log

```
D:\bonin\anwd\code>java -version  
java version "1.5.0_08"  
Java(TM) 2 Runtime Environment,  
Standard Edition (build 1.5.0_08-b03)  
Java HotSpot(TM) Client VM  
(build 1.5.0_08-b03, mixed mode, sharing)
```

```
D:\bonin\anwd\code>javac  
de/leuphana/ics/anonym/Anonym.java
```

```
D:\bonin\anwd\code>dir  
.\de\leuphana\ics\anonym\*.class
```

```
... 416 Anonym$1.class  
... 416 Anonym$2.class  
... 1.178 Anonym.class
```

```
D:\bonin\anwd\code>java  
de.leuphana.ics.anonym.Anonym  
Creating object: 1  
Creating object: 2  
Creating object: 3  
You are foo!  
You are bar!  
OK!
```

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

Lösung Aufgabe A.22 S. 457:**A.22.3:**

Die Alternativen lassen sich fehlerfrei compilieren. Die Applikation der Klasse `BottomPattern` läuft in beiden Fällen fehlerfrei wie das folgende Protokoll zeigt. Allerdings sind die Ergebnisse verschieden.

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode)
```

```
D:\bonin\anwd\code>javac
  de/unilueneburg/as/structure/BottomPattern.java
```

```
D:\bonin\anwd\code>java
  de.unilueneburg.as.structure.BottomPattern
--> TopPattern()
--> CenterPattern()
--> BottomPattern(int, String, String, String)
17
```

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

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode)
```

```
D:\bonin\anwd\code>javac
  de/unilueneburg/as/construct/BottomPattern.java
```

```
D:\bonin\anwd\code>java
  de.unilueneburg.as.construct.BottomPattern
slot = 17 now = 13
--> TopPattern(int)
--> TopPattern(int, String)
--> CenterPattern(int, String, String)
--> BottomPattern(int, String, String, String)
13
```

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

A.22.4:

Im Quellcode von *Emil Cody* können folgende Konstrukte gestrichen werden:

```
BottomPattern()
CenterPattern(int slot, String top, String center)
TopPattern(int slot, String top)
```

Im Quellcode von *Emma Softy* können folgende Konstrukte gestrichen werden:

```
BottomPattern()
CenterPattern()
TopPattern()
```

A.22.5:

Emma Softy hat besser programmiert, weil eine Restriktion wie zum Beispiel der hier vorgegebene Wertebereiche für `slot` möglichst in dem Konstruktor programmiert wird, dessen Objektattribut (Slot) betroffen ist. Hier steht daher die Restriktion zweckmäßigerweise in der Klasse `TopPattern` und nicht in der Klasse `BottomPattern`. Bei der *Emil-Cody*-Alternative wird der Wert daher nicht überprüft.

Lösung Aufgabe A.24 S. 471:

A.24.1:

Die folgende Protokolldatei dokumentiert die nach dem Compilieren entstandenen Dateien und verdeutlicht welche gebraucht werden:

Protokoll Foo.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
(build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
de\leuphana\ics\innerclass\Foo.java

D:\bonin\anwd\code>dir
de\leuphana\ics\innerclass\*.class
```

```
214 Foo$1.class
1.245 Foo$Bar.class
814 Foo$KlasseA.class
799 Foo$KlasseB.class
799 Foo$KlasseC.class
523 Foo.class
```

```
D:\bonin\anwd\code>erase
de\leuphana\ics\innerclass\Foo$1.class
```

```
D:\bonin\anwd\code>dir
de\leuphana\ics\innerclass\*.class
```

```
1.245 Foo$Bar.class
814 Foo$KlasseA.class
799 Foo$KlasseB.class
799 Foo$KlasseC.class
523 Foo.class
```

```
D:\bonin\anwd\code>java
de.leuphana.ics.innerclass.Foo$Bar
Slot-Wert in Instanz c: KlasseA
Anzahl der Udates in der KlasseA: 1
```

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

A.24.2:

```
>java de.leuphana.ics.innerclass.Foo$Bar
```

A.24.3:

Protokolldatei Foo.log ↔ S. 549

Lösung Aufgabe A.25 S. 474:

A.25.1:

Protokolldatei TelefonBuchProg.log

```
D:\bonin\anwd\code>java -version
java version "1.5.0_08"
```

```

Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/telefon/TelefonBuchProg.java

D:\bonin\anwd\code>java
  de.leuphana.ics.telefon.TelefonBuchProg
TelefonEintrag: Otto +49/4131/677175
TelefonEintrag: Emma +49/4131/677144
OK --- Objekte sind gleich!

D:\bonin\anwd\code>javac
  de/leuphana/ics/telefon/TelefonLookupProg.java

D:\bonin\anwd\code>java
  de.leuphana.ics.telefon.TelefonLookupProg Emma
Emma +49/4131/677144

D:\bonin\anwd\code>

```

Außerdem wird eine Datei `tbuch.ser` erzeugt.

A.25.2:

Listing B.29: TelefonLookupProg.log

```

/**
2  * Primitives TelefonLookupProg.java
  *
4  *
  * @since    29-Jun-1998, 30-May-2007
6  * @author  Hinrich E. G. Bonin
  * @version  1.2
8  */
package de.leuphana.ics.telefon;
10
import java.io.InputStream;
12 import java.io.ObjectInputStream;
import java.util.Enumeration;
14
public class TelefonLookupProg
16 {
  public static void main(String [] args)
18 {
    if (args.length != 1)

```

```

20     {
21         System.out.println (
22             "Usage: java _TelefonLookupProg _name");
23         System.exit(1);
24     }
25     String kurzname = args[0];
26     try
27     {
28         FileInputStream fin =
29             new FileInputStream("tbuch.ser");
30         ObjectInputStream in =
31             new ObjectInputStream(fin);
32         TelefonBuch t =
33             (TelefonBuch) in.readObject();
34         in.close();
35         Enumeration keys =
36             t.tabelle.keys();
37
38         while (keys.hasMoreElements())
39         {
40             String key =
41                 (String) keys.nextElement();
42             TelefonEintrag te = t.getEintrag(key);
43             if (te.getKurzname().equalsIgnoreCase(kurzname))
44             {
45                 System.out.println (
46                     te.getKurzname() +
47                     " " +
48                     te.getTelefon());
49                 break;
50             }
51         }
52     } catch (Exception e)
53     {
54         e.printStackTrace(System.out);
55     }
56 }
57 }
58 }

```

Lösung Aufgabe A.26 S. 479:

A.26.1:

Listing B.30: IO

```

/**
 * Ergebnis der Diskussionsrunde

```



```

*
4 * @since      01-Jun-1998, 31-May-2007
* @author      Hinrich E. G. Bonin
6 * @version    1.1
*/
8 package de.leuphana.ics.lingo;

10 interface I0
11 {
12     public void m1();

14     public void m2();
15 }

```

Listing B.31: K1

```

/**
2 * Ergebnis der Diskussionsrunde
*
4 * @since      01-Jun-1998, 31-May-2007
* @author      Hinrich E. G. Bonin
6 * @version    1.1
*/
8 package de.leuphana.ics.lingo;

10 class K1 implements I0
11 {
12     private K4 v1;
13     private K4 v2;
14     private K4 v3;

16     public void m1() { }

18     public void m2() { }

20     public static void main(String [] args)
21     {
22     }
23 }

```

Listing B.32: K2

```

/**
2 * Ergebnis der Diskussionsrunde
*
4 * @since      01-Jun-1998, 31-May-2007
* @author      Hinrich E. G. Bonin
6 * @version    1.1
*/

```

```
8 package de.leuphana.ics.lingo;
10 class K2
12 {
14     K1 s = new K1();
16 }
```

Listing B.33: K3

```
/**
2  * Ergebnis der Diskussionsrunde
3  *
4  * @since      01-Jun-1998, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8 package de.leuphana.ics.lingo;
10 class K3 extends K2
12 {
14     public static K4 c1;
16 }
```

Listing B.34: K4

```
/**
2  * Ergebnis der Diskussionsrunde
3  *
4  * @since      01-Jun-1998, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8 package de.leuphana.ics.lingo;
10 class K4
12 {
14     public void m3() { }
16 }
```

A.26.2:

Listing B.35: K1mit

```
/**
2  * Ergebnis der Diskussionsrunde
3  * mit Getter und Setter
4  *
5  * @since      01-Jun-1998, 31-May-2007
6  * @author     Hinrich E. G. Bonin
```

```

    * @version    1.1
    */
8  package de.leuphana.ics.lingo;
10
11  class K1mit implements I0
12  {
13      private K4 v1;
14      private K4 v2;
15      private K4 v3;
16
17      public void m1() { }
18
19      public void m2() { }
20
21      public K4 getV1()
22      {
23          return v1;
24      }
25
26      public K4 getV2()
27      {
28          return v2;
29      }
30
31      public K4 getV3()
32      {
33          return v3;
34      }
35
36      public void setV1(K4 v1)
37      {
38          this.v1 = v1;
39      }
40
41      public void setV2(K4 v2)
42      {
43          this.v2 = v2;
44      }
45
46      public void setV3(K4 v3)
47      {
48          this.v3 = v3;
49      }
50
51      public static void main(String [] args)
52      {
53      }
54  }
```

Listing B.36: K2mit

```

/**
2  * Ergebnis der Diskussionsrunde
  * mit Getter und Setter
4  *
  * @since      01-Jun-1998, 31-May-2007
6  * @author    Hinrich E. G. Bonin
  * @version    1.1
8  */
package de.leuphana.ics.lingo;

10
class K2mit
12 {
14     K1mit s = new K1mit();

16     public K1mit getS()
    {
18         return s;
    }

20     public void setS(K1mit s)
22     {
        this.s = s;
24     }
}

```

Hinweis: Klassenvariable (Modifikator `static`) werden üblicherweise direkt zugegriffen, also nicht mit Gettern und Settern versehen.

Lösung Aufgabe A.27 S. 479:

A.27.1:

Die Abbildung B.13 S. 557 zeigt das Klassendiagramm.

A.27.2:

Listing B.37: IO

```

/**
2  * Ergebnis der Systemanalyse
  *
4  * @since      01-Jun-1998, 31-May-2007
  * @author    Hinrich E. G. Bonin
6  * @version    1.1
  */
8  package de.leuphana.ics.mix;

```

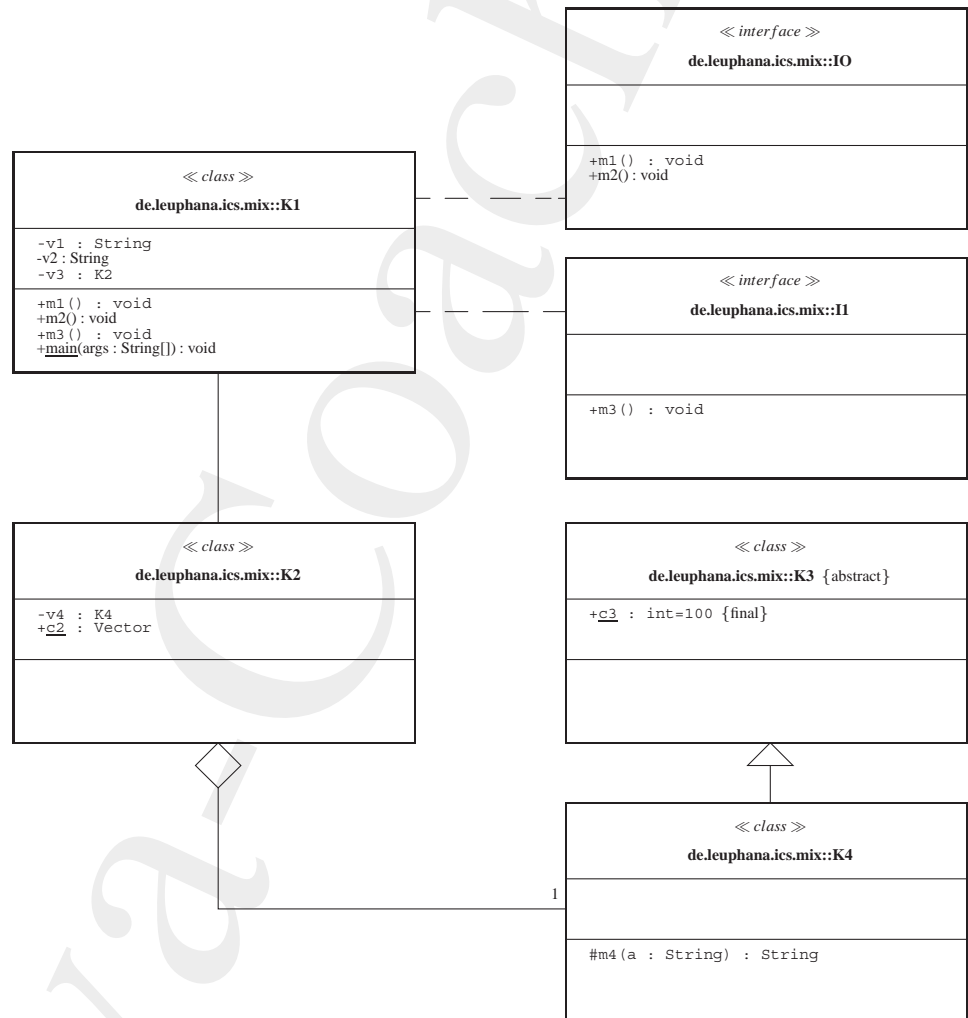


Abbildung B.13: Aufgabe A.27.1 S.480: Klassendiagramm für `de.leuphana.ics.mix`

```
10 interface I0
11 {
12     public void m1();
13
14     public void m2();
15 }
```

Listing B.38: I1

```
/**
2  * Ergebnis der Systemanalyse
3  *
4  * @since      01-Jun-1998, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8 package de.leuphana.ics.mix;

10 class K1 implements I0 , I1
11 {
12     private String v1;
13     private String v2;
14     private K2 v3;

16     public void m1() { }

18     public void m2() { }

20     public void m3() { }

22     public static void main(String [] args)
23     {
24     }
25 }
```

Listing B.39: K1

```
/**
2  * Ergebnis der Systemanalyse
3  *
4  * @since      01-Jun-1998, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8 package de.leuphana.ics.mix;

10 class K1 implements I0 , I1
11 {
12     private String v1;
```

```

14     private String v2;
15     private K2 v3;
16
17     public void m1() { }
18
19     public void m2() { }
20
21     public void m3() { }
22
23     public static void main(String [] args)
24     {
25     }
26 }

```

Listing B.40: K2

```

/**
2  * Ergebnis der Systemanalyse
3  *
4  * @since      01-Jun-1998, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8  package de.leuphana.ics.mix;
9
10 import java.util.Vector;
11
12 class K2
13 {
14     private K4 v = new K4();
15     public static Vector c2;
16 }

```

Listing B.41: K3

```

/**
2  * Ergebnis der Systemanalyse
3  *
4  * @since      01-Jun-1998, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8  package de.leuphana.ics.mix;
9
10 abstract class K3
11 {
12     public final static int c3 = 100;
13 }

```

Listing B.42: K4

```

/**
2  * Ergebnis der Systemanalyse
3  *
4  * @since      01-Jun-1998, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8  package de.leuphana.ics.mix;

10 class K4 extends K3
11 {
12     protected String m4(String a)
13     {
14         return a;
15     }
16 }

```

A.27.3:

```

>javac de/leuphana/ics/mix/K1.java
>java de.leuphana.ics.mix.K1

```

Lösung Aufgabe A.28 S. 481:A.28.1:

Die Abbildung B.14 S. 561 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.28.2:

Die Unterscheidung ist erforderlich, damit das -Konstrukt nur in der Überschrift zu einem Font in der Größe von 28pt führt und nicht auch in der Aufzählung.

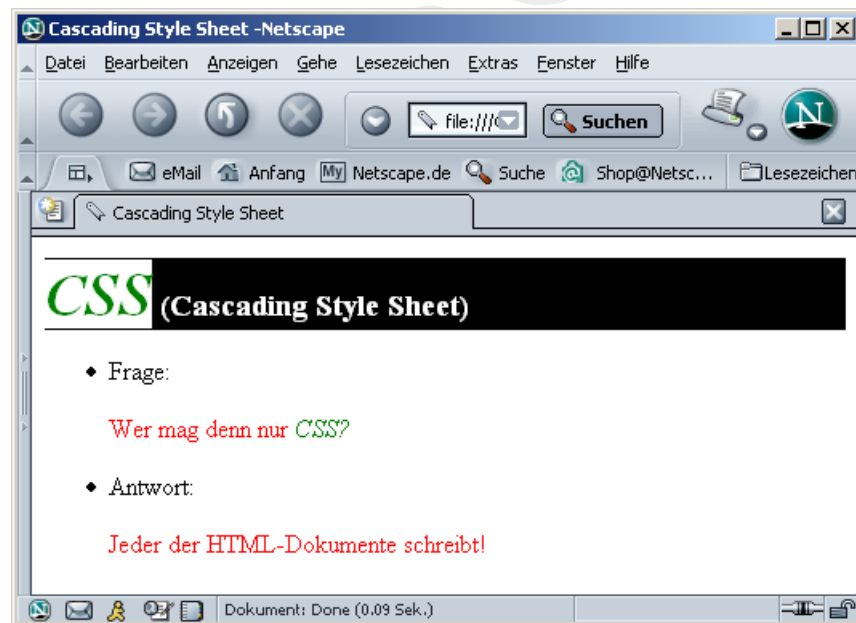


Abbildung B.14: Aufgabe A.28 S. 481: Mehrere Stylespezifikationen

Lösung Aufgabe A.29 S. 482:A.29.1:

Schreibfehler in Zeile 25 — korrekt: <h1>

A.29.2:

```
p {
  font-size: 12pt;
  color: red;
  background: white;
}
h1 em {
  font-size: 28 pt;
}
h1 {
  font-size: 14pt;
  color: black;
  background: white;
}
em {
  font-style: italic;
  color: green;
  background: white;
}
```

A.29.3:

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 gelben Hintergrund in einer Größe von 14pt dargestellt. Da ein Browser für Überschriften häufig die Darstellung in Dickschrift (bold) voreingestellt hat, ist die gesamte Überschrift „dick“ dargestellt.

Lösung Aufgabe A.30 S. 484:A.30.1:

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.30.2:

Protokolldatei Hund.log

```
>java Hund
Bello von der Eulenburg lebt!
Berta vom Lechgraben lebt!
Alex vom Hirschgarten lebt!
Berta vom Lechgraben
>
```

A.30.3:

Listing B.43: HundNorm

```
/**
 2  * Beispiel einer Rekursion innerhalb der Klasse: Vater und
 3  * Mutter sind wieder vom Typ HundNorm
 4  *
 5  * @author    Hinrich Bonin
 6  * @version   1.0
 7  * @since    22-Jan-1999
 8  */
 9  import java.util.*;
10
11  class HundNorm
12  {
13      private String name = "";
14      private boolean weiblich = true;
15      private HundNorm mutter;
16      private HundNorm vater;
17
18      public HundNorm(String name, boolean weiblich)
19      {
20          setName(name);
21          setWeiblich(weiblich);
22          System.out.println(name + " lebt!");
23      }
24  }
25
26
```

```
28     public String getName()  
    {  
30         return name;  
    }  
  
32  
    public void setName(String name)  
34    {  
        this.name = name;  
36    }  
  
38  
    public boolean getWeiblich()  
40    {  
        return weiblich;  
42    }  
  
44  
    public void setWeiblich(boolean weiblich)  
46    {  
        this.weiblich = weiblich;  
48    }  
  
50  
    public HundNorm getMutter()  
52    {  
        return mutter;  
54    }  
  
56  
    public void setMutter(HundNorm mutter)  
58    {  
        this.mutter = mutter;  
60    }  
  
62  
    public HundNorm getVater()  
64    {  
        return vater;  
66    }  
  
68  
    public void setVater(HundNorm vater)  
70    {  
        this.vater = vater;  
72    }  
  
74  
    public static void main(String [] argv)
```

```

76     {
77         HundNorm bello =
78             new HundNorm("Bello_von_der_Eulenburg", false);
79         bello.setMutter(
80             new HundNorm("Berta_vom_Lechgraben", true));
81         bello.setVater(
82             new HundNorm("Alex_vom_Hirschgarten", false));
83         System.out.println(
84             bello.getMutter().getName());
85     }
86 }

```

Lösung Aufgabe A.31 S. 487:

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.32 S. 489:

A.32.1:

Listing B.44: Partner.dtd

```

<?xml version="1.0" encoding="UTF-8"?>
2 <!-- Bonin 30-Jun-2004 -->
<!ELEMENT partner (firma*)>
4 <!ELEMENT firma (adresse+, kontakt+)>
<!ATTLIST firma
6     name CDATA #REQUIRED
     rechtsform CDATA #REQUIRED
8     gerichtsstand CDATA #REQUIRED
>
10 <!ELEMENT adresse EMPTY>
<!ATTLIST adresse

```

```

12     plz CDATA #REQUIRED
13     ort CDATA #REQUIRED
14     strasse CDATA #REQUIRED
15 >
16 <!ELEMENT kontakt EMPTY>
17 <!ATTLIST kontakt
18     telefon CDATA #REQUIRED
19     fax CDATA #REQUIRED
20     email CDATA #REQUIRED
21 >

```

A.32.2:

Listing B.45: Partner0

```

/**
2  * Erzeugung der XML-Datei Partner.xml
3  *
4  * @since      30-Jun-2004, 31-May-2007
5  * @author     Hinrich E. G. Bonin
6  * @version    1.1
7  */
8  package de.leuphana.ics.moritzGmbH;
9
10 import java.io.BufferedReader;
11 import java.io.BufferedWriter;
12 import java.io.FileNotFoundException;
13 import java.io.FileReader;
14 import java.io.FileWriter;
15 import java.io.IOException;
16
17 import org.jdom.DocType;
18 import org.jdom.Document;
19 import org.jdom.Element;
20 import org.jdom.output.Format;
21 import org.jdom.output.XMLOutputter;
22
23 public class Partner0
24 {
25     public void toXML(
26         String sourceFile,
27         String xmlFile,
28         String dtdFile)
29     {
30         Document document = new Document();
31         document.setDocType(new DocType("partner", dtdFile));
32

```

```
34     Element root = new Element("partner");
35     document.setRootElement(root);
36
37     try
38     {
39         BufferedReader reader =
40             new BufferedReader(
41                 new FileReader(sourceFile));
42
43         String line = null;
44         Element aktuelleElement = null;
45
46         while ((line = reader.readLine()) != null)
47         {
48             if (line.equals("!F"))
49             {
50                 Element fElement =
51                     new Element("firma");
52
53                 fElement.setAttribute(
54                     "name",
55                     reader.readLine());
56
57                 fElement.setAttribute(
58                     "rechtsform",
59                     reader.readLine());
60
61                 fElement.setAttribute(
62                     "gerichtsstand",
63                     reader.readLine());
64
65                 root.addContent(fElement);
66                 aktuelleElement = fElement;
67             }
68             if (line.equals("!A"))
69             {
70                 Element aElement =
71                     new Element("adresse");
72
73                 aElement.setAttribute(
74                     "plz",
75                     reader.readLine());
76
77                 aElement.setAttribute(
78                     "ort",
79                     reader.readLine());
80
81                 aElement.setAttribute(
82                     "strasse",
```

```
82         reader.readLine ();
84         aktuelleElement .addContent (
85             aElement);
86     }
87     if (line .equals ("!K"))
88     {
89         Element kElement =
90             new Element ("kontakt");
91
92         kElement .setAttribute (
93             "telefon",
94             reader.readLine ());
95
96         kElement .setAttribute (
97             "fax",
98             reader.readLine ());
99
100        kElement .setAttribute (
101            "email",
102            reader.readLine ());
103
104        aktuelleElement .addContent (
105            kElement);
106    }
107 }
108
109 XMLOutputter outputter = new XMLOutputter ();
110
111 Format format = Format .getPrettyFormat ();
112 format .setEncoding ("ISO-8859-1");
113 outputter .setFormat (format);
114
115 BufferedWriter writer =
116     new BufferedWriter (new FileWriter (xmlFile));
117
118 outputter .output (document , writer);
119
120 writer .close ();
121 } catch (FileNotFoundException e)
122 {
123     e.printStackTrace ();
124 } catch (IOException e)
125 {
126     e.printStackTrace ();
127 }
128 }
129
130 public static void main (String [] args)
```



```
132     {  
133         new Partner0().toXML(  
134             "de/leuphana/ics/moritzGmbH/Partner.txt",  
135             "de/leuphana/ics/moritzGmbH/Partner.xml",  
136             "Partner.dtd");  
137     }  
138 }
```

Java-Coach

Listing B.46: Partner.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE partner SYSTEM "Partner.dtd">
4 <partner>
   <firma name="Otto" rechtsform="GmbH" gerichtstand="Berlin">
6     <adresse plz="D-21391" ort="Reppenstedt" strasse="Eulenburg.6" />
       <adresse plz="D-21339" ort="üLneburg" strasse="Volgershall.1" />
8     <kontakt telefon="04131-677175" fax="04131-677140" email="info@otto-lueneburg.com" />
   </firma>
   <firma name="Meyer" rechtsform="AG" gerichtstand="Hamburg">
10    <adresse plz="D-21000" ort="Hamburg" strasse="Alsterweg.18" />
12    <adresse plz="D-21000" ort="Hamburg" strasse="Vogelsburg.2" />
14    <kontakt telefon="040-11111" fax="040-11112" email="meyer@marktplatz-hamburg.de" />
   </firma>
</partner>
```

A.32.3:

Listing B.47: Partner1

```

/**
2  *  Ausgabe der XML-Datei Partner.xml
  *
4  *  @since      30-Jun-2004, 31-May-2007
  *  @author     Hinrich E. G. Bonin
6  *  @version    1.1
  */
8  package de.leuphana.ics.moritzGmbH;

10 import java.io.File;
  import java.io.IOException;
12 import java.util.List;

14 import org.jdom.Attribute;
  import org.jdom.Document;
16 import org.jdom.Element;
  import org.jdom.JDOMException;
18 import org.jdom.input.SAXBuilder;

20 public class Partner1
  {
22     public void showXML(String sourceFile)
  {
24         try
  {
26             Document document =
  new SAXBuilder().build(
28                 new File(sourceFile));

30             Element root =
  document.getRootElement();
32
  List firmen = root.getChildren();
34
  for (int i = 0; i < firmen.size(); i++)
36     {
38         Element firma =
  (Element) firmen.get(i);

40         System.out.println(
  "Firma:_" +
42         firma.getAttributeValue("name"));

```

```
44      System.out.println(  
45          "\tRechtsform :␣" +  
46          firma.getAttributeValue(  
47              "rechtsform" ));  
48  
49      System.out.println(  
50          "\tGerichtsstand :␣" +  
51          firma.getAttributeValue(  
52              "gerichtsstand" ));  
53  
54      List infos = firma.getChildren();  
55  
56      for (int j = 0; j < infos.size(); j++)  
57          {  
58              Element element =  
59                  (Element) infos.get(j);  
60  
61              List attribute =  
62                  element.getAttributes();  
63              Attribute attr1 =  
64                  (Attribute) attribute.get(0);  
65              Attribute attr2 =  
66                  (Attribute) attribute.get(1);  
67              Attribute attr3 =  
68                  (Attribute) attribute.get(2);  
69  
70              System.out.println(  
71                  "\t" +  
72                  element.getName() + ":" );  
73  
74              System.out.println(  
75                  "\t\t" +  
76                  attr1.getName() + " :␣" +  
77                  attr1.getValue());  
78  
79              System.out.println(  
80                  "\t\t" +  
81                  attr2.getName() + " :␣" +  
82                  attr2.getValue());  
83  
84              System.out.println(  
85                  "\t\t" +  
86                  attr3.getName() + " :␣" +  
87                  attr3.getValue());  
88          }  
89      } catch (JDOMException e)  
90      {
```

```
92         e.printStackTrace();
93     } catch (IOException e)
94     {
95         e.printStackTrace();
96     }
97 }
98
99 public static void main(String [] args)
100 {
101     new Partner1().showXML(
102         "de/leuphana/ics/moritzGmbH/Partner.xml");
103 }
104 }
```

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 "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode, sharing)

D:\bonin\anwd\code>javac
  de/leuphana/ics/moritzGmbH/Partner0.java

D:\bonin\anwd\code>java
  de.leuphana.ics.moritzGmbH.Partner0

D:\bonin\anwd\code>javac
  de/leuphana/ics/moritzGmbH/Partner1.java

D:\bonin\anwd\code>java
  de.leuphana.ics.moritzGmbH.Partner1 >
  de/leuphana/ics/moritzGmbH.PartnerList.txt

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

Anhang C

Hinweise zur Nutzung von J2SE SDK

C.1 Java™ 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. 576). (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 5.1.2 S. 81):

```
> 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.)
#
# Erlaueerterung 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™ auf der Windows-Plattform

Es wird als Erläuterungsbeispiel wieder¹ 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:

```
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
```

¹↔ Abschnitt 5.1.2 S. 81.

```
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 Toshiba 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

Notationshinweis:

Literaturangaben zum Vertiefen des Stoffes dieses Manuskripts sind vor grauem Hintergrund ausgewiesen.

Java-Coach

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: [Traummüller91], 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 Institute, Washington, DC, June 1988 (veröffentlicht in: SIG-PLAN Notices, Band 23, Special Issue, September 1988).
- [Broy/Siedersleben02]
 Manfred Broy / Johannes Siedersleben; Objektorientierte 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: Originalarbeit 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 language, Reading, Massachusetts u. a. (Addison-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/Shao96] Arthur van Hoff / Sami Shao / Orca Starbuck; HOOKED ON JAVA, (Addison-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 (Addison-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™, 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.
- [Oestereich06] Bernd Oestereich; Analyse und Design mit UML 2.1 — Objektorientierte Softwareentwicklung, München Wien (Oldenbourg Wissenschaftsverlag GmbH) 2006, 8., aktualisierte Auflage, ISBN 3-486-57926-6. [Hinweis: Eine praxisorientierte Darstellung der Grundlagen objektorientierte Entwicklungsmethodik auf der Basis von UML.]
- [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 evaluation 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-Quelleanalysen.]
- [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™ 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
- [Traunmüller91] 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

Application Server — JBoss

JBoss

JBoss ist ein sehr verbreiteter *Open Source Application Server* (↔ Bild D.1 S.587). 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)

Data Binding Framework — Castor

Castor

Castor ist ein *Open Source Data Binding Framework* für Java™ . 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 — SAX & DOM

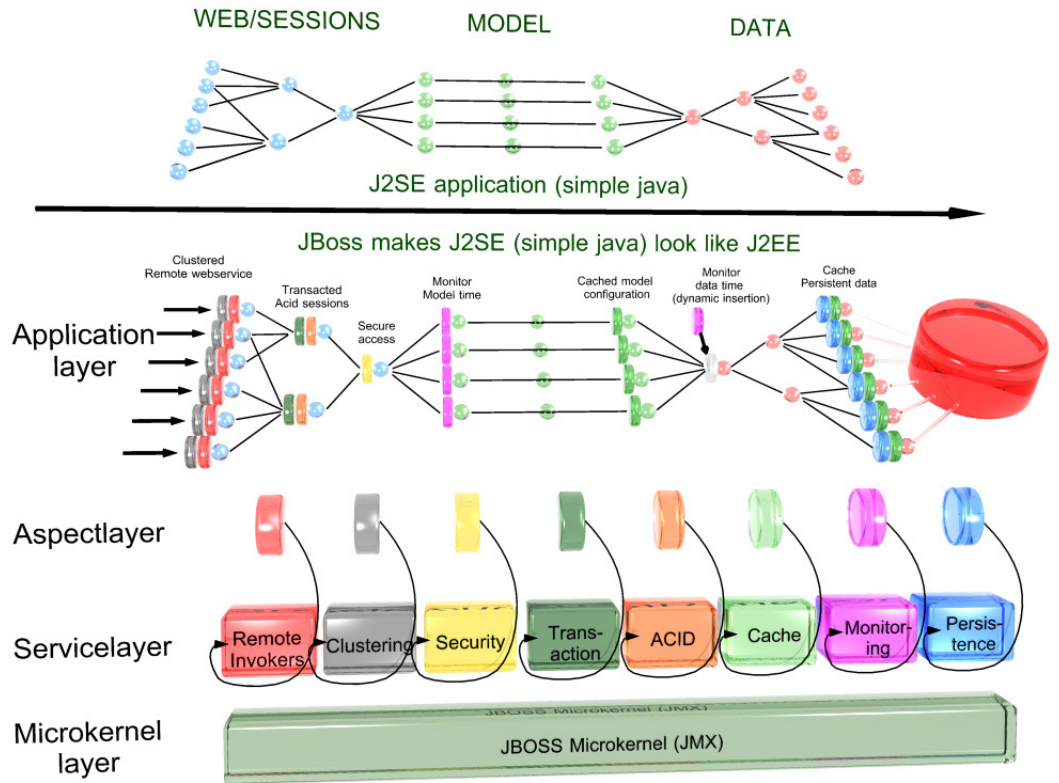
JDOM

JDOM stellt Klassen für SAX (*Simple API for XML*) und DOM (*Document Object Model*) bereit.
<http://www.jdom.org> (online 19-May-2004)

Ant — *Build Tool*

Ant

Ant ist ein Werkzeug zum Erstellen einer Anwendung auf der Basis ei-



Legende:

Quelle: <http://www.jboss.org/overview> (online 23-Jan-2004)

Abbildung D.1: JBoss

ner XML-Datei.

<http://ant.apache.org/>
(online 10-Jun-2004)

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™

<http://java.sun.com/nav/whatis/storyofjava.html>

Java™ -Erweiterung — *Pizza Compiler*

Der *Pizza Compiler* erweitert Java um drei neue Features:

- *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™ –COACH

Mit folgender Software wurde das Dokument JAVA™ –COACH erstellt:

Editor: GNU Emacs 21.3.1; 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: L^AT_EX-File (Device Independent) to Postscript: dvips(k) 5.90a Copyright 2002 Radical Eye Software (www.radical-eye.com)

PS→PDF: Postscript file to PDF-File: Adobe Acrobat Distiller 8.0 Professional

Security: Adobe Acrobat 8.0 Professional (Dokumenten-Version 5.0)

D.4 Abkürzungen und Akronyme

ACID	<u>A</u> tomicity, <u>C</u> onsistency, <u>I</u> solation, <u>D</u> urability
AIX	<u>A</u> dvanced <u>I</u> nteractive <u>E</u> xecutive IBM's Implementation eines UNIX-Operating Systems
AJDE	<u>A</u> spectJ <u>D</u> evelopment <u>E</u> nvironment
AOP	The term <u>a</u> spect- <u>o</u> riented <u>p</u> rogramming is attributed to Kiczales et. a.
API	<u>A</u> pplication <u>p</u> rogramming <u>i</u> nterface
BDK	<u>B</u> eans <u>D</u> evelopment <u>K</u> it
BNF	<u>B</u> ackus- <u>N</u> aur- <u>F</u> orm
CICS	<u>C</u> ustomer <u>I</u> nformation <u>C</u> ontrol <u>S</u> ystem
CGI	<u>C</u> ommon <u>G</u> ateway <u>I</u> nterface
CLOS	<u>C</u> ommon <u>L</u> isp <u>O</u> bject <u>S</u> ystem

CORBA	<u>C</u> ommon <u>O</u> bject <u>R</u> quest <u>B</u> roker <u>A</u> rchitecture
CSS	<u>C</u> ascading <u>S</u> tyl <u>S</u> heets
CVS	<u>C</u> oncurrent <u>V</u> ersions <u>S</u> ystem
CTM	<u>C</u> omponent <u>T</u> ransaction <u>M</u> onitor
DBMS	<u>D</u> ata <u>B</u> ase <u>M</u> anagement <u>S</u> ystem
DOM	<u>D</u> ocument <u>O</u> bject <u>M</u> odel
DTD	<u>D</u> ocument <u>T</u> ype <u>D</u> efinition
DTP	<u>D</u> esktop <u>P</u> ublishing
DVI	<u>L</u> A _T <u>E</u> X's <u>D</u> evice <u>I</u> ndependent File Format
EIS	<u>E</u> nterprise <u>I</u> nfomation <u>S</u> ystems (Server)
EJB	<u>E</u> nterprise <u>J</u> ava <u>B</u> eans
Emacs	<u>E</u> ditng <u>M</u> acros gigantic, functionally rich editor
ERP	<u>E</u> nterprise <u>R</u> esource <u>P</u> laning Systems
FTP	<u>F</u> ile <u>T</u> ransfer <u>P</u> rogram
GJ	A <u>G</u> eneric <u>J</u> ava Language Extension
GNU	<u>G</u> NU's <u>N</u> ot <u>U</u> nix
GUI	<u>G</u> raphical <u>U</u> ser <u>I</u> nterface
HTTP	<u>H</u> ypertext <u>T</u> ransfer <u>P</u> rotocol
HTTPS	<u>H</u> TT <u>P</u> <u>S</u> ecure
IANA	<u>I</u> nternet <u>A</u> ssigned <u>N</u> umbers <u>A</u> uthority
IBM	<u>I</u> nternational <u>B</u> usiness <u>M</u> achines Corporation
IDE	<u>I</u> ntegrated <u>D</u> evelopment <u>E</u> nvironment
IIOp	<u>I</u> nternet <u>I</u> nter- <u>O</u> perability <u>P</u> rotocol
IP	<u>I</u> nternet <u>P</u> rotocol
J2EE	<u>J</u> ava 2 <u>E</u> nterprise <u>E</u> dition
J2ME	<u>J</u> ava 2 <u>M</u> icro <u>E</u> dition
J2SE	<u>J</u> ava 2 <u>S</u> tandard <u>E</u> dition
JAR	<u>J</u> ava <u>A</u> rchiv
JDBC	<u>J</u> ava <u>D</u> atabase <u>C</u> onnectivity
JDK	<u>J</u> ava <u>D</u> evelopment <u>K</u> it
JIT	<u>J</u> ust- <u>I</u> n- <u>T</u> ime-Compiler
JMI	<u>J</u> ava <u>M</u> etad <u>a</u> ta <u>I</u> nterface
JMS	<u>J</u> ava <u>M</u> essaging <u>S</u> ervice
JNI	<u>J</u> ava <u>N</u> ative <u>M</u> ethod <u>I</u> nterface
JNDI	<u>J</u> ava <u>N</u> aming and <u>D</u> irectory <u>I</u> nterface
JPEG	<u>J</u> oint <u>P</u> hotographics <u>E</u> xpert <u>G</u> roup
JRE	<u>J</u> ava <u>R</u> untime <u>E</u> nvironment
JSP	<u>J</u> ava <u>S</u> erver <u>P</u> ages

JTA	<u>J</u> ava <u>T</u> ransaction <u>A</u> PI
JVM	<u>J</u> ava <u>V</u> irtual <u>M</u> aschine
JXTA	<u>J</u> uxt <u>a</u> pose (pronounced <i>juxta</i> — Peer-to-Peer-Framework)
LDAP	<u>L</u> ightweighth <u>D</u> irectory <u>A</u> ccess <u>P</u> rotocol
LISP	<u>L</u> ist <u>P</u> rocessing
MDA	OMG's <u>M</u> odel <u>D</u> riven <u>A</u> rchitecture
MOF	OMG's <u>M</u> eta- <u>O</u> bject <u>F</u> acility
MOM	<u>M</u> essage- <u>O</u> riented <u>M</u> iddleware
MOP	<u>M</u> eta <u>o</u> bject <u>p</u> rotocol; a programmer could override the default behavior of the dispatch method in order to affect what happens when a virtual function is called.
NaN	<u>N</u> ot <u>a</u> <u>N</u> umber
NT	Microsoft Windows <u>N</u> ew <u>T</u> echnology 32-Bit-Betriebssystem mit Multithreading und Multitasking
OAK	<u>O</u> bject <u>A</u> pplication <u>K</u> ernel
OCL	<u>O</u> bject <u>C</u> onstraint <u>L</u> anguage
OGSA	<u>O</u> pen <u>G</u> rid <u>S</u> ervices <u>A</u> rchitecture
OGSI	<u>O</u> pen <u>G</u> rid <u>S</u> ervices <u>I</u> nfrast <u>r</u> ucture
OMG	<u>O</u> bject <u>M</u> anagement <u>G</u> roup
OOP	<u>O</u> bject- <u>o</u> riented <u>P</u> rogramming
OQL	<u>O</u> bject <u>Q</u> uery <u>L</u> anguage
P2P	<u>P</u> eer- <u>t</u> o- <u>P</u> eer
PGP	<u>P</u> retty <u>G</u> ood <u>P</u> rivacy
POP	<u>P</u> ost <u>o</u> bject- <u>o</u> riented <u>p</u> rogramming
POP3	<u>P</u> ost <u>O</u> ffice <u>P</u> rotocol <u>3</u>
PROLOG	<u>P</u> ROgramming in <u>L</u> OGic
PS	<u>P</u> ostscript file
RAS	<u>R</u> eliability, <u>A</u> vailability, <u>S</u> erviceability
RISC	<u>R</u> educed <u>I</u> nstruction <u>S</u> et
RMI	Java <u>R</u> emote <u>M</u> ethod <u>I</u> nvo <u>c</u> ation Protocol
RCS	<u>R</u> evision <u>C</u> ontrol <u>S</u> ystem
RPC	<u>R</u> emote <u>P</u> rocedure <u>C</u> all
SAX	<u>S</u> imple <u>A</u> PI for <u>X</u> ML
SCCS	<u>S</u> ource <u>C</u> ode <u>C</u> ontrol <u>S</u> ystem
SDK	Software <u>D</u> eveloper's <u>K</u> it
SGML	<u>S</u> tandard <u>G</u> eneralized <u>M</u> arkup <u>L</u> anguage
SMTP	<u>S</u> imple <u>M</u> ail <u>T</u> ransport <u>P</u> rotocol
SSH	<u>S</u> ecure <u>S</u> hell

SSL	<u>S</u> ecure <u>S</u> ocket <u>L</u> ayer
SQL	<u>S</u> tandard <u>Q</u> uery <u>L</u> anguage
TCP	<u>T</u> ransmission <u>C</u> ontrol <u>P</u> rotocol
UML	<u>U</u> nified <u>M</u> odeling <u>L</u> anguage
URI	<u>U</u> niversal <u>R</u> esource <u>I</u> dentifier
URL	<u>U</u> niform <u>R</u> esource <u>L</u> ocator
W3C	<u>W</u> orld <u>W</u> ide <u>W</u> eb <u>C</u> onsortium
WSDL	<u>W</u> eb <u>S</u> ervices <u>D</u> escription <u>L</u> anguage
XHTML	<u>E</u> xtensible <u>H</u> ypertext <u>M</u> arkup <u>L</u> anguage
XMI	<u>X</u> ML <u>M</u> etadata <u>I</u> nterchange format
XML	<u>E</u> xtensible <u>m</u> arkup <u>l</u> anguage

Java-Coach

Abbildungsverzeichnis

1.1	Java — wunderschöne Insel Indonesiens	18
1.2	J2SE: Skizze der Komponenten	21
3.1	UML-Beziehungselement: Assoziation	50
3.2	Beispiel einer Assoziation: Ein Unternehmen beschäftigt viele Mitarbeiter	51
3.3	Beispiel einer direkten rekursiven Assoziation	52
3.4	Beispiel einer Assoziationsklasse: ArbeitsVerhält- nis	53
3.5	Beispiel einer qualifizierenden Assoziation (mitId) . .	54
3.6	UML-Beziehungselement: Aggregation	55
3.7	Beispiel einer Aggregation: Ein Fahrrad hat zwei Laufräder mit jeweils 36 Speichen	55
3.8	UML-Beziehungselement: Komposition	56
3.9	Beispiel einer Komposition: Window mit Slider, Header und Panel	57
3.10	UML-Beziehungselement: Vererbung	59
3.11	Beispiel einer Vererbung	61
4.1	Von der Bytecode-Produktion bis zur Ausführung	69
5.1	Klassendiagramm für HelloWorld	77
5.2	HelloWorld.java in <i>Eclipse Platform</i>	79
5.3	Beispiel HelloWorld — javadoc	82
5.4	Klassendiagramm für Foo	83
5.5	Beispieldateien im Editor jEdit	86
5.6	Beispieldateien im Editor GNU Emacs	87
5.7	Klassendiagramm für FahrzeugApp	89
5.8	Ei-Huhn-Reihenfolgenproblem	102
5.9	Klassendiagramm der Applikation MyNetProg.	106

5.10	Darstellung der CGI-Datei <code>spass.ksh</code>	107
5.11	Interface-Vererbung	116
5.12	Abstrakte Klasse	120
5.13	Klassendiagramm für <code>ActionApplet</code>	123
5.14	Beispiel: <code>ActionApplet</code>	127
5.15	Beispiel: <code>CounterApplet</code>	134
5.16	Klassendiagramm für <code>CounterGUI</code>	135
5.17	Beispiel: <code>MyProgressBar</code> — Klassenhierarchie . . .	140
5.18	Beispiel: <code>MyProgressBar</code> — im Browser	142
5.19	Beispiel: <code>MyProgressBar</code> — im Appletviewer . . .	142
5.20	Klassendiagramm für <code>MyProgressBar</code>	143
6.1	Klassendiagramm für das Multithreading-Beispiel „Textausgabe“	158
6.2	Ergebnis von <code>SequenzArbeit</code>	162
6.3	Ergebnis von <code>ThreadedArbeit</code>	165
6.4	Java™ AWT: Konstruktion des Delegationsmodells . .	168
6.5	Klassendiagramm <code>SetFarbe</code>	171
6.6	Ergebnis: <code>java de.unilueneburg.as.farbe.-TextEingabeFarbe</code>	174
6.7	Ergebnis: <code>java de.unilueneburg.as.farbe.-ListWahlFarbe</code>	176
6.8	Klassendiagramm <code>ZeigeTastenWert</code>	179
6.9	Ergebnis: <code>java de.unilueneburg.as.keyvalue.-ZeigeTastenWert</code>	181
6.10	Beispiel <code>PButton</code>	187
6.11	Beispiel <code>UseButton</code>	190
6.12	Klassendiagramm für <code>WitzA</code>	195
6.13	Klassendiagramm für <code>WitzB</code>	196
6.14	Klassendiagramm für <code>WitzC</code>	197
6.15	Klassendiagramm für <code>WitzD</code>	198
6.16	Klassendiagramm für <code>WitzE</code>	200
6.17	Klassendiagramm für <code>WitzF</code>	201
6.18	Klassendiagramm für <code>WitzG</code>	202
6.19	Klassendiagramm für <code>WitzGa</code>	204
6.20	Klassendiagramm für <code>WitzH</code>	205
6.21	Klassendiagramm für <code>WitzJ</code>	206
6.22	<i>Inner class</i> — Beispiel <code>BlinkLicht</code>	216
6.23	JPEG-Bildbeispiel	256
6.24	<i>Own Distributed Object Protocol</i> — Klassendiagramm .	259

6.25	Applikation KontoClient	265
6.26	Associations für eine Zusammensetzung	299
6.27	Komposition mittels Interface-Konstruktion	305
6.28	Skizze der Enterprise JavaBeans™ -Architektur	321
7.1	Eclipse — Package Explorer	339
7.2	Eclipse — CVS Repositories	340
7.3	Mit Ant erzeugtes Manifest	349
7.4	Logging-Übersicht — java.util.logging	352
7.5	Dokumentation mittels javadoc	372
8.1	XHTML: exampleCSS.html & CSS: myStyle.css	394
A.1	Klassendiagramm für Inheritance	417
A.2	Klassendiagramm für TableProg	420
A.3	Beispiel: Fach-Student-Durchschnittsnote	426
A.4	Klassendiagramm von Emil Cody	457
A.5	Klassendiagramm von Emma Softy	460
B.1	Aufgabe A.1.1 S. 406: Klassendiagramm für die Montage- sicht	494
B.2	Aufgabe A.1.2 S. 406: Diagrammerweiterung um den Montageplatz	505
B.3	Aufgabe A.1.2 S. 406: Klassendiagramm erzeugt aus dem Java-Quellcode	506
B.4	Aufgabe A.2.1 S. 408: SVI-Klassendiagramm „Hauptteil“	507
B.5	API mit javadoc der Klasse Flinte	527
B.6	Aufgabe A.2.1 S. 408: SVI-Klassendiagramm „Zielfern- rohr“	528
B.7	Aufgabe A.2.2 S. 408: SVI-Klassendiagramm „Waffen- besitzkarte“	530
B.8	Aufgabe A.17 S. 443: „Animierter Mond“— applet- viewer	539
B.9	Aufgabe A.17 S. 443: „Animierter Mond“— Netscape 7.0	540
B.10	Aufgabe A.18 S. 446: Ausgangsfenster	541
B.11	Aufgabe A.18 S. 446: Hauptfenster	542
B.12	POET Developer: Beispiel Buch	543
B.13	Aufgabe A.27.1 S. 480: Klassendiagramm für de.leuphana. ics.mix	557
B.14	Aufgabe A.28 S. 481: Mehrere Stylespezifikationen . . .	561

D.1 JBoss	587
-----------------	-----

Java-Coach

Tabellenverzeichnis

1	Internet Smileys	7
1.1	Java-Beschreibung von Sun Microsystems, Inc. USA . .	19
1.2	J2SE: Pakete und ihre Aufgabe	21
2.1	Ausrichtung von objekt-orientierten Ansätzen	33
2.2	Java im Vergleich mit Smalltalk und CLOS	36
2.3	Aspekte Menge und Struktur	37
3.1	Klassifikation von UML-Diagrammen	44
3.2	UML-Basiselement: Klasse	46
3.3	Kennzeichnung einer Variablen in UML	47
3.4	Kennzeichnung einer Methode in UML	48
3.5	Angabe der Multiplizität	52
5.1	Beispiel: ET-Ernährung	98
5.2	Syntax eines XHTML-Konstruktes	129
5.3	Applet-Einbindung: Einige Attribute des <code><object></code> - Konstruktes	131
5.4	Reservierte Java™-Schlüsselwörter	145
5.5	Datentypbeschreibung anhand von Produktionsregeln . .	146
5.6	Javas einfache Datentypen	147
5.7	Java-Zugriffsrechte für Klasse, Schnittstelle, Variable und Methode	150
5.8	Zugriffssituationen	151
5.9	Operator — Priorität und Assoziativität	153
6.1	Listener-Typen: <code>event</code> → <code>listener</code> → <code>method</code>	177
6.2	<code>Object</code> → <code>listener</code>	178
6.3	Benutzeraktion auf das GUI <i>object</i>	178
6.4	Rückgabewert von <code>getActionCommand()</code>	178

6.5	FastObjects Java Binding Collection Interfaces	236
6.6	RMI: <i>Stub/Skeleton</i> , <i>Remote-Reference</i> und <i>Transport</i> . .	273
7.1	CVS-Repository — Kommandos	343
7.2	Math. <i>saw</i> (<i>x</i>)-Formel	365
9.1	Einfache Ausdrücke : OO-Code	402
C.1	Java-Umgebungsvariablen für die AIX-Plattform	576

Listings

5.1	HelloWorld	78
5.2	Foo	81
5.3	FahrzeugApp	88
5.4	Fahrzeug	91
5.5	Fahrt	93
5.6	Counter	95
5.7	CounterApplication	96
5.8	Essen	98
5.9	Console	99
5.10	Ei	101
5.11	Huhn	103
5.12	MyNet	105
5.13	MyNetProg	108
5.14	MyGetWebPage	111
5.15	ImpulseGenerator	114
5.16	TypApplication	117
5.17	Grundtyp	118
5.18	Untertyp	118
5.19	AbClassApplication	121
5.20	AbClass	121
5.21	ActionApplet	123
5.22	ActionApplet.html	126
5.23	PruefeApplet.html	132
5.24	CounterApplet.html	134
5.25	Counter	136
5.26	CounterGUI	137
5.27	CounterApplet	138
5.28	MyProgressBar	139
5.29	MyProgressBar.html	143
6.1	Teil des Dokumentes SequenzArbeit.html	158

6.2	SequenzArbeit	159
6.3	EinfacherAusgeber	160
6.4	Teil des Dokumentes ThreadedArbeit	163
6.5	ThreadedArbeit	163
6.6	SetFarbe	170
6.7	TextEingabeFarbe	172
6.8	ListWahlFarbe	173
6.9	ZeigeTastenWert	176
6.10	TastenWert	180
6.11	PersButton	186
6.12	PersButtonProg	186
6.13	UseButton	188
6.14	FileApp	193
6.15	WitzA	195
6.16	WitzB	196
6.17	WitzC	197
6.18	WitzD	199
6.19	WitzE	199
6.20	WitzF	200
6.21	WitzG	202
6.22	WitzGa	203
6.23	WitzH	204
6.24	WitzJ	205
6.25	WitzK	207
6.26	Aussen	209
6.27	BlinkLicht.html	212
6.28	BlinkLicht	212
6.29	ZeigeKlasse	218
6.30	Aufruf	225
6.31	Person	229
6.32	MyClass	244
6.33	Bind	245
6.34	Lookup	246
6.35	Delete	248
6.36	Foo	250
6.37	MyImgStore	252
6.38	ImgJpegStore	254
6.39	Konto	258
6.40	Konto_Stub	258

6.41	KontoServer	261
6.42	Konto_Skeleton	262
6.43	KontoClient	263
6.44	KontoServerProg	264
6.45	Bank	272
6.46	RemoteBankServer	277
6.47	Detail XML-Input	284
6.48	Detail XML-Output	284
6.49	lief-dat.dtd	285
6.50	lief-dat.xml	286
6.51	jahr-tab.dtd	288
6.52	Aggregation	288
6.53	AggregationProg	295
6.54	jahr-tab.xml	296
6.55	FahrradApp	300
6.56	Fahrrad	301
6.57	Rahmen	302
6.58	Lauftrad	302
6.59	Felge	303
6.60	Nabe	303
6.61	FahrradApp	304
6.62	Fahrrad	307
6.63	Rahmen	307
6.64	Lauftrad	308
6.65	Felge	308
6.66	Nabe	309
6.67	Comosite	309
6.68	Leaf	310
6.69	Component	311
6.70	SimpleBean	316
6.71	RaumRemote	322
6.72	RaumHomeRemote	323
6.73	RaumLocal	323
6.74	RaumHomeLocal	323
6.75	RaumBean	324
6.76	ClientA	326
6.77	ejb-jar.xml	328
7.1	TestString	344
7.2	build.xml	349

7.3	LogSample	352
7.4	loggingResult.xml	353
7.5	logger.dtd	353
7.6	SourceFileExample	359
7.7	F — So nicht!	362
7.8	Math	366
7.9	MathValue	368
7.10	MathProg	369
7.11	Reichweite	371
8.1	index.php	383
8.2	myStyle.css	393
8.3	exampleCSS.html	396
A.1	Wert	409
A.2	Scoping	410
A.3	Kontrolle	410
A.4	Iteration	412
A.5	Linie	413
A.6	LinieProg	415
A.7	Inheritance	416
A.8	Foo	418
A.9	Bar	419
A.10	TableProg	420
A.11	LookupTable	421
A.12	Rekursion	422
A.13	Fakultaet	423
A.14	Fach	425
A.15	Student	427
A.16	DurchschnittProg	428
A.17	Foo	430
A.18	Bar	430
A.19	SlotI	432
A.20	Foo	432
A.21	Bar	433
A.22	Queue — Fall I	434
A.23	QueueProg — Fall I	436
A.24	Queue — Fall II	438
A.25	LinkedQueue — Fall II	438
A.26	QueueProg — Fall II	440
A.27	SimpleThread.html	443

A.28 SimpleThread	444
A.29 ExampleAWT.html	446
A.30 MaskeAufbau	447
A.31 MyCanvas	448
A.32 SimpleListener	450
A.33 DemoAWT	450
A.34 Regal	452
A.35 Anonym	455
A.36 BottomPattern	457
A.37 CenterPattern	458
A.38 TopPattern	459
A.39 BottomPattern	460
A.40 CenterPattern	461
A.41 TopPattern	462
A.42 ptj.opt	463
A.43 Buch	464
A.44 Autor	467
A.45 Person	468
A.46 BuchBind	468
A.47 BuchLookUp	469
A.48 ListeLookUp	470
A.49 Foo	471
A.50 TelefonBuchProg	474
A.51 TelefonBuch	475
A.52 TelefonEintrag	477
A.53 myPage.html	481
A.54 myPageStyle.css	481
A.55 myFINAL.html	482
A.56 myFStyle.css	483
A.57 Hund	485
A.58 C1	487
A.59 C2	488
A.60 CProg	489
B.1 Fahrrad	493
B.2 Einrad	495
B.3 Zweirad	496
B.4 Laufrad	498
B.5 Schutzblech	499
B.6 Gepaecktraeger	500

B.7	Anbauteil	501
B.8	Montageanleitung	502
B.9	RVEGmbH	503
B.10	SVIProdukt	505
B.11	Waffe	508
B.12	Lauf	509
B.13	Langwaffe	511
B.14	Kurzwaffe	511
B.15	Gewehr	512
B.16	Pistole	512
B.17	Revolver	513
B.18	Flinte	514
B.19	Buechse	514
B.20	KombinierteWaffe	515
B.21	Querflinte	515
B.22	Bockflinte	516
B.23	Bockbuechse	517
B.24	Bockbuechsflinte	519
B.25	Drilling	520
B.26	SVIGmbH	522
B.27	echo	529
B.28	Ganze	546
B.29	TelefonLookupProg.log	551
B.30	I0	552
B.31	K1	553
B.32	K2	553
B.33	K3	554
B.34	K4	554
B.35	K1mit	554
B.36	K2mit	556
B.37	I0	556
B.38	I1	558
B.39	K1	558
B.40	K2	559
B.41	K3	559
B.42	K4	560
B.43	HundNorm	563
B.44	Partner.dtd	565
B.45	Partner0	566

B.46 Partner.xml 570
B.47 Partner1 571

Java-Coach

Java-Coach

Anhang E

Index

Java-Coach

Index

Java-Coach

Index

- (), 153
- *, 153
- *=, 153
- +, 153, 409
- ++, 153
- +=, 153
- , 153
- , 153
- =, 153
- ., 153
- /, 153
- /=, 153
- <, 153
- <<, 153
- <<=, 153
- <=, 98, 153
- =, 153
- =, 153, 409
- >, 153
- >=, 153
- >>, 153
- >>=, 153
- >>>, 153
- >>>=, 153
- ? :, 153
- [], 153
- %, 153
- %=, 153
- &, 153
- &=, 153
- &&, 153
- , 356
- {, 371, 410
- }, 371, 410
- 2U, 62
- 3C, 62
- Abelson, Harold, 581
- abstract, 145, 416
- abstract, 119
- accept (), 262
- ACID, 589
- Acrobat, 589
 - Distiller, 589
- ActionApplet, 127
- ActionEvent, 137, 139
- ActionListener, 170
- ActionListener, 139
- ActionListener (), 137
- actionPerformed (), 137, 139
- add (), 138, 139, 297
- addActionListener (), 137, 139
- addChangeListener (), 139
- addContent (), 298
- addPropertyChangeListener (), 316
- addVetoableChangeListener (), 316
- Adobe
 - Acrobat, 589
 - Distiller, 589
- Aggregation, 50, 55, 400
- AIX, 73, 589
 - CLASSPATH, 575
 - JDK, 575
- AJDE, 589
- Alternative, 358
- Alur, Deepak, 581
- Anfangskommentar, 356
- Anforderung, 363
- Ant, 348–351, 586
- Anwendungsfelder, 37
- AOP, 586, 589

- Apache Projekt, 334
- Apache Software Foundation, 79
- API, 22, 527, 589
- Applet, 20, 127
 - HTML-Einbindung, 127
- appletviewer, 74
- Applikation, 20, 127
- Architektur
 - EJB, 321
- Archiv
 - JAR, 190
- archive, 131
- args, 77
- argv, 77
- assert, 249
- assertEquals(), 344
- Assertion, 249
- AssertionFailedError, 345
- assertTrue(), 344
- Assoziation, 49, 50, 55, 430
 - degenerierte, 51
 - Klasse, 51
 - rekursive, 52
- Assoziativität, 153
- Attribut, 45
- Ausführungsmodell, 401
- Authentifikation, 72
- AWT
 - Beispiel, 446
- backward chaining, 32
- BASIC, 35
- BDK, 589
- Beans
 - Entity, 319
 - Message-driven, 319
 - Session, 319
- Beck, Kent, 344
- Belli, Fevzi, 581
- Berliner, Brian, 341
- Betriebssystem, 348
- Bezeichner, 356, 361–371
- Bill Joy, 583
- bind(), 233
- Blaha, M., 585
- BNF, 589
- Bobrow, Daniel G., 582–584
- Bonin, Hinrich E.G., 581, 582
- Booch, G., 582
- Booch, Grady, 6
- Boolean, 99
- boolean, 145–147
- booleanValue(), 99
- BorderFactory, 139
- Borland Together
 - Control Center, 89, 102, 116, 120, 135, 140, 143, 158, 171, 179, 187, 299, 305, 426, 457, 460
- Bos, Bert, 584
- bottom, 131
- Bowser, 74
- Boxologie, 581
 - Begriff, 28
- break, 145
- Broy, Manfred, 582
- BufferedReader, 99, 111
- BufferedWriter(), 566
- build(), 297, 570
- build.xml, 348
- Bycode
 - verifier, 70
- byte, 145–147
- byte[10], 114
- Bytecode, 69
- byvalue, 145
- C, 70
- C++, 70, 585
- C-Shell, 341
- cancel(), 114
- cannot resolve symbol, 531
- Canvas, 316
- Cascading Style Sheets, 384–397, 584
- case, 145
- cast, 145
- Casting, 182
- Castor, 586
- catch, 99, 145

- catch(), 375
- CGI, 105, 589
- chaining
 - backward, 32
 - forward, 32
- ChangeEvent, 139
- ChangeListener, 139
- char, 145–147
- charAt(), 99
- Christerson, M., 583
- CICS, 313, 589
- Class
 - anonym
 - Beispiel, 202, 203, 205
 - CSS, 388
 - inner, 195–216, 371
 - Beispiel, 198, 209, 211
 - local
 - Beispiel, 200
- class, 45, 145
- class, 356
- Class Diagram, 89, 102, 116, 120, 135, 143, 158, 171, 179, 187, 299, 305, 426, 457, 460
- classid, 131
- CLASSPATH, 88
 - AIX, 575
 - NT, 577
- CLASSPATH, 345
- Clocksini, W.F., 582
- clone, 144, 228
- Cloning, 228–233
- CLOS, 35, 36, 583, 589
- close(), 111
- COBOL, 35
- Code
 - Konvention, 355
- codebase, 131
- codetype, 131
- Color, 139
- com.ibm.ejs.ns.jndi.CNInitialContextFactory, 326
- commit(), 233
- Common Business Objects, 378–379
- Common Gateway Interface, 105
- Common Object Request Broker Architecture, 266
- Compiler
 - Pizza, 589
- Compilieren
 - implizit, 101
- Component Model, 312
- Composition, 298
- Computer Science, 581
- const, 145
- Constraint, 58
- Constraints, 238
- Constructor, 217
- Container, 139
- Context, 326
- continue, 145
- Control Center
 - Borland Together, 89, 102, 116, 120, 135, 140, 143, 158, 171, 179, 187, 299, 305, 426, 457, 460
- CORBA, 62, 266, 590
 - COS, 22
 - IIOP, 23
- Core Business Processes, 378–379
- Corel
 - CAPTURE, 589
 - PHOTO-PAINT, 589
- COS
 - CORBA, 22
- CounterApplet, 134
- create(), 322, 323
- CreateException, 326
- createRaisedBevelBorder(), 139
- Crupi, John, 581
- CSCW, 581
 - .cshrc, 341
- CSS, 384–397, 590
 - class, 388
 - id, 389
- CSS1, 584

- CTM, 590
- CVS, 338–342, 590
 - Repository, 340
 - SSH2 plug in, 342
- da, 250
- Dahl, O.-J., 582
- D’Anjou, Jim, 585
- data, 131
- Data-directed Programming, 31
- DataInputStream, 105
- Date, 81, 531
- Daten-gesteuerte Programmierung
 - Wurzel, 31
- Datenbank-Managementsystem, 35
- Datenabstraktion, 400
- Datentyp, 45
- DBMS, 22, 35, 590
- Debian
 - Linux, 341
- default, 150, 151
- default, 145
- Deklaration, 357
- Dekrementieren, 95
- DelegatingMethodAccessorImpl, 345
- Denkmodell, 27
- Deployment Descriptors, 328
- <description>, 348
- Descriptors
 - Deployment, 328
- detach(), 298
- disableassertions, 250
- Diskriminator, 58
- DISPATCH-Funktion, 31
- DISPOSE_ON_CLOSE, 137
- do, 145
- DocType(), 566
- Document, 288, 297
- Document(), 566
- DOM, 283, 590
- DOS
 - Windows, 577
- Double, 99
- double, 98, 145–147
- doubleValue(), 99
- DSTC, 62
- DTD, 328, 590
 - EJB, 588
- DTP, 385, 590
- DVI, 590
- dvips, 589
- E, 78
- E-Government, 586
- ea, 250
- Eckel, Bruce, 582
- Eclipse, 79
- Eddy, F., 585
- Editor
 - GNU Emacs, 87
 - jEdit, 86
- Effizienz, 35
- EIS, 20, 22, 590
- EJB, 22, 319–329, 584, 588, 590
 - Architektur, 321
 - Beispiel, 321–329
 - Definition, 319
 - <ejb-jar>, 328
 - ejb.raum, 322–324, 326
 - ejbActivate(), 324
 - ejbCreate(), 324
 - ejbLoad(), 324
 - ejbPassivate(), 324
 - ejbPostCreate(), 324
 - ejbRemove(), 324
 - ejbStore(), 324
- Element, 288, 297
- Element(), 566
- else, 145
- else if, 98
- Emacs, 74, 87, 590
 - GNU, 589
- Embley, David W., 582
- enableassertions, 250
- English, Jason, 583
- Enterprise Architect, 505
- Enterprise JavaBeans, 22, 584
- Entity Beans, 319
- Entscheidungstabelle
 - Beispiel, 98

- equals, 144
- equals (), 409
- ERP, 20, 590
- ET
 - Beispiel, 98
- Event handler, 167
- Event Handling, 114
- Event Model, 167–182
- extends, 145
- Externalizable, 185
- extssh, 340
- extssh2, 340

- FahrzeugApp, 85–95
- Fairbrother, Scott, 585
- Fakultät, 422
- false, 145
- Field, 217
- FileHandler, 352
- FileWriter (), 566
- FINAL, 582
- final, 98, 145
- finalize, 144
- finally, 145
- findByPrimaryKey (), 322, 323
- FinderException, 326
- Flanagan, David, 582
- float, 145–147
- FlowLayout, 139
- flush (), 111
- Foo.java, 81–85
- for, 145
- for () {}, 81
- Forbrig, Peter, 582
- FORTRAN, 30, 35
- forward chaining, 32
- Fowler, Martin, 582
- Freeman, Adam, 583
- FTP, 74, 111, 590
- Funktion, 47
- future, 145

- Gabriel, Richard P., 583
- Gamma, Erich, 344
- Ganzes-Teile-Beziehung, 55, 57

- Garbage Collection, 114
- gc (), 88
- Geheimnisprinzip, 400
- Generalsierung, 58
- generic, 145
- get (), 288
- getAttribute (), 298
- getAttributeValue (), 288, 298
- getByName (), 111
- getChild (), 297
- getChildren (), 288, 297, 570
- getClass, 144
- getContentPane (), 137–139
- getInputStream (), 105
- getInputStream (), 111
- getIntValue (), 298
- getOutputStream (), 111
- getPrettyFormat (), 566
- getResource (), 193
- getResourceAsStream (), 193
- getRootElement (), 288, 297, 570

- Getter, 314
- getValue (), 139
- GJ, 590
- GNU, 74, 590
 - Emacs, 87, 589
- gnumake, 348
- Goldberg, Adele, 583
- Gosling, James, 19, 581, 583
- goto, 145
- Granulat, 400
- Graphical User Interface, 167
- GridLayout (), 137
- Groupware, 582
- Grune, Dick, 341
- GUI, 167, 590
- GUI Object
 - Listener, 178

- Harkey, Dan, 584
- hashCode, 144
- HashMap, 425
- hasNext (), 193
- Hau, Michael, 583

- Heap Allocation, 252
- height, 131
- HelloWorld, 76–81
- Herrmann, Stephan, 583
- Hirplastizität, 4
- Hoff, van Arthur, 583
- Hohlfeld, Sven, 8
- Hors, Le Arnaud, 583
- HotSpot, 20
 - Memory Options, 252
- HTML, 20, 582
 - 4.0, 583
- HTML-Syntax
 - rekursive Definition, 129
- HTTP, 22, 590
- HTTPS, 22, 111, 590

- IANA, 590
- IBM, 73, 590
 - San Francisco Projekt, 583
 - VisualAge for Java, 73
- Id
 - CSS, 389
- IDE, 21, 73, 79, 590
- IEEEremainder(), 78
- if, 145
- IIOP, 590
 - CORBA, 23
- iiop:///, 326
- Implementierungsvererbung, 122
- implements, 145
- import, 111, 145
- import, 356
- Ince, Darrel, 583
- InetAddress, 111
- Inheritance, 58, 59
- init(), 138, 139
- InitialContext, 326
- INITIAL_CONTEXT_FACTORY, 326
- Inkrementieren, 95
- inner, 145
- InputStream, 111, 193
- InputStreamReader, 99, 111
- instanceof, 145, 153
- Instanziierung, 400

- int, 145–147
- Integrität
 - referenzielle, 51
- Interface, 102, 115, 116, 299, 305
- interface, 145
- interface, 356
- Internetzugriff, 105
- invoke(), 225
- invoke(), 345
- IOException, 99, 114
- IP, 590
- ipconfig /all, 578
- ISO 8879, 383
- Iteration, 412

- J2EE, 20, 590
- J2ME, 590
- J2SE, 19, 588, 590
 - Komponenten, 21
- JAAS, 586
- Jacobs, Ian, 583
- Jacobsen, Ivar, 6
- Jacobson, Ivar, 583
- Jähnichen, Stefan, 583
- jam, 348
- JApplet, 138, 139
- JAR, 190, 328, 348, 590
 - Mainfest, 349
 - Manifest, 192, 194
- Java, 30, 36
 - 1.1, 586
 - Spezifikation, 583
- Applet, 127
- Applikation, 20, 127
- Database Connectivity, 22
- Definition, 19
- Eclipse, 79
- Einsatzgrund, 586
- Enterprise Beans, 22, 584
- Erweiterung, 589
- Historiebericht, 583, 588
- HotSpot, 20
- Introduction, 584
- klassische Beschreibung, 581
- Message Service, 22

- Naming and Directory Interface, 22
- Quellcodedokumentation, 527
- road to, 585
- Runtime, 20
- SDK, 20
- Threads, 584
- Transaction API, 23
- Werkzeuge, 73
- java, 74
- Java Archiv, 190, 328
- Java Server Pages, 20
- java.awt.*, 21, 139, 316
- java.awt.event.*, 137, 139
- java.awt.GridLayout, 137
- java.beans.*, 316
- java.io.*, 21, 105, 111
- java.io.BufferedReader, 566
- java.io.BufferedWriter, 288, 566
- java.io.File, 288, 570
- java.io.FileNotFoundException, 566
- java.io.FileReader, 566
- java.io.FileWriter, 288, 566
- java.io.InputStream, 193
- java.io.IOException, 114, 193, 352, 566, 570
- java.io.ObjectInputStream, 258, 262
- java.io.ObjectOutputStream, 258, 262
- java.net, 105
- java.net.*, 105, 111
- java.net.ServerSocket, 262
- java.net.Socket, 258, 262
- java.net.URL, 193
- java.rmi.RemoteException, 322, 326
- java.security.*, 21
- java.sql.*, 21
- java.swing.*, 21
- java.util.List, 288, 570
- java.util.logging.FileHandler, 352
- java.util.logging.Level, 352
- java.util.logging.Logger, 352
- java.util.Properties, 326
- java.util.Scanner, 193
- java.util.Timer, 114
- java.util.TimerTask, 114
- JavaBeans, 313–319
- javac, 74
- Javadoc, 82
- javadoc, 81, 361, 527
- javadoc, 74
- javah, 74
- javap, 74
- javax.ejb.CreateException, 322, 323, 326
- javax.ejb.EJBHome, 322
- javax.ejb.EJBLocalHome, 323
- javax.ejb.EJBLocalObject, 323
- javax.ejb.EJBObject, 322
- javax.ejb.EntityBean, 324
- javax.ejb.EntityContext, 324
- javax.ejb.FinderException, 322, 323, 326
- javax.naming.*, 21
- javax.naming.Context, 326
- javax.naming.InitialContext, 326
- javax.naming.NamingException, 326
- javax.rmi.CORBA.*, 21
- javax.rmi.PortableRemoteObject, 326
- javax.swing.*, 137, 139
- javax.swing.event.*, 139
- JBoss, 586, 587
- JButton, 137, 139
- JCraft, Inc., 342
- jdb, 74
- JDBC, 22, 588, 590
- JDK, 20, 590
- AIX, 575

- NT, 577
- JDOM, 297, 586
- jEdit, 86, 589
- JFrame, 137
- JIT, 70, 590
- JLabel, 137
- JLabel.RIGHT, 137
- JMI, 590
- JMS, 22, 588, 590
- JMX, 586
- JNDI, 22, 588, 590
- JNI, 265, 590
- join, 166
- Jonsson, P., 583
- Joy, Bill, 19
- JPanel, 137
- JPEG, 252, 590
- JProgressBar, 139
- JRE, 21, 590
- JSP, 20, 588, 590
- JTA, 23, 591
- JUnit, 342–348, 588
 - junit-4.3.1-src.jar, 345
 - junit-4.3.1.jar, 345
 - junit.framework.TestCase, 344
 - junit.framework.TestSuite, 344
 - junit.swingui.TestRunner, 344
 - junit.textui.TestRunner, 344
 - junit4.3.1, 345
- JVM, 252, 591
- JXTA, 591

- Kapselung, 400
- Kehn, Dan, 585
- Kellerman, John, 585
- Ken, Arnold, 581
- KeyListener, 176
- Kiczales, Gregor, 584
- Kim, Won, 584
- Klasse, 35, 45
 - abstrakte, 45, 119
 - Name, 61
- Klassenhierarchie, 140
- Klassenkonzept, 34
- Klassenvariable
 - Beispiel, 204
- Kodierung
 - Tipps, 361–378
- Kommentar, 356
 - am Anfang, 356
- Kommentierung, 356
- Komponente, 583
 - Modell, 312
- Komponentenbegriff, 401
- Komposition, 50, 56, 57, 102, 299, 305, 401
- Komposition, 298
- Konsole
 - Eingabe, 95, 98
- Konstruktor, 85, 420
 - Name, 61
- Kontrollstruktur, 32
- Konvention
 - Code, 355
- Konvertierung
 - Objekt, 182
 - Zeit, 38
- Konzept
 - Klasse, 34
 - Prototyp, 34
- Kritik
 - Objektorientierung, 582
- Kurtz, Barry D., 582

- Larman, Craig, 584
- L^AT_EX, 589
- LDAP, 22, 591
- left, 131
- length, 81
- Lie, Hakon Wium, 584
- Lieberman, H., 584
- Light-weight Directory Access Protocol, 22
- Link, 50
- Linux
 - Debian, 341
- LISP, 31, 581, 591
 - Common, 582
- List, 288, 297
- Liste, 434, 438

- Listener, 167, 314
 - Typen, 177
- Literalkonstante, 409
- Lochovsky, Frederick H., 584
- Logger, 352
- Logging API, 351–355
- Long, 99
- long, 145–147
- Long Running Transaction, 233
- lookup (), 235
- Lorenson, W., 585

- Main-Class:, 192, 194
- Mainfest
 - JAR, 349
- make, 348
- Malks, Crupi, 581
- Manes, Anne, 313
- Manifest, 192, 194, 315
- Math.E, 78
- Math.IEEEremainder(), 78
- Math.PI, 78
- Math.pow(), 78
- Math.saw(x), 363
- Math.sqrt(), 78
- McCarty, Pat, 585
- MDA, 62, 591
- Message-driven Beans, 319
- Mellish, C.S., 582
- Member, 45
- Menge, 38
- Merkmal, 48
 - Name, 62
- Mertens, Peter, 583
- Message Service, 22
- Metaklasse, 45
- Metaobject
 - Protocol, 584
- Method, 217
- Methode, 35, 47
 - Name, 61
- Microsoft
 - Visio, 589
 - Windows NT, 73
- middle, 131

- Middleware
 - message-oriented, 22
- Miller, Joaquin, 584
- Modell
 - Begriff, 28
 - Komponenten, 312
- Modularität, 400
- MOF, 591
- MOM, 22, 591
- Monson-Haefel, Richard, 584
- Moon, David, 582
- MOP, 591
- Multiplizität, 52
- Multithreaded Environment, 114
- Multithreading, 157–167
- Muster, 581
- Muster-gesteuerter Prozeduraufruf
 - Wurzel, 31
- MyNetProg, 105–114
- MyProgressBar, 140, 142
- Myrhaug, KB., 582

- Nachrichtenaustausch, 32
- Name
 - Klasse, 61
 - Konstruktor, 61
 - Merkmal, 62
 - Methode, 61
 - Paket, 61
 - Stereotyp, 62
 - Variable, 61
 - Zusicherung, 62
- NamingException, 326
- NaN, 364, 591
- native, 145, 147
- NativeMethodAccessorImpl, 345
- Netscape, 74
- Neurogenese, 4
- new, 145
- nmake, 348
- Not a Number, 364
- Notation, 6–7
 - allgemeine Regeln, 356
- notify, 144, 166
- notifyAll, 144

- NT, 591
- null, 111, 145
- Nygaard, K., 582
- OAK, 591
- Object
 - Listener, 178
- <object>, 131
- Object Management Group, 6
- ObjectInputStream, 258, 262
- ObjectOutputStream, 258, 262
- Objekt
 - Begriff, 27
 - Konvertierung, 182
 - Menge, 38
 - Struktur, 38
- Objektidentität, 400
- Objektorientierung
 - Anwendungsfelder, 35, 37
 - Databases, 584
 - Definition, 32
 - Kritik, 582
 - Modellierung, 582, 585
 - Software Engineering, 583
 - strikt, 401
 - Systemanalyse, 582
 - Umsetzung, 400
 - Wurzeln, 29
- OCL, 591
- ODBMS, 233–249
- ODMG, 233
- Oechsle, Rainer, 584
- Oestereich, Bernd, 43, 63, 584
- Övergaard, G., 583
- OGSA, 591
- OGSI, 591
- OMG, 6, 62, 591
- OOP, 591
- openConnection(), 105
- openStream(), 193
- Operator, 153
- operator, 145
- OPM, 62
- OQL, 591
- Orfali, Robert, 584
- org.jdom.Attribute, 570
- org.jdom.DocType, 288, 566
- org.jdom.Document, 288, 566, 570
- org.jdom.Element, 288, 566, 570
- org.jdom.input.SAXBuilder, 288, 570
- org.jdom.JDOMException, 570
- org.jdom.output.Format, 288, 566
- org.jdom.output.XMLOutputter, 288, 566
- Ourosoff, Nick, 584
- outer, 145
- OutputStream, 111
- OutputStreamWriter, 111
- P2P, 591
- pack(), 137
- Package, 88
- package, 77, 145, 258, 262
- package, 356
- Package Explorer, 339
- Paket, 49, 77, 88
 - Name, 61
- Paradigma, 27
- Parameter, 81
- parseDouble(), 99
- parseInt(), 99
- parseLong(), 99
- Partl, Hubert, 584
- Pascal, 35
- Pattern, 581
- pattern matching, 32
- Persistenz, 182–194, 315
- PGP, 591
- PI, 78
- Pizza Compiler, 589
- Plasitizität
 - Hirn, 4
- POET, 233
 - Beispiel, 463
- SDK
 - Version 6.1.8.76, 544

- Pointer, 71
- Polymorphismus, 30
 - Compilierung, 31
 - Laufzeit, 31
 - Multiargument, 35
 - Zeitpunkt, 31
- Pooley, Rob, 585
- POP, 591
- POP3, 111, 591
- Portabilität, 67
- PortableRemoteObject, 326
- pow(), 78
- Pragmatik, 144–149
- Premarlani, W., 585
- Primitive Typen, 401, 584
- print(), 111
- PrintWriter, 111
- Priorität, 153
- private, 145, 150, 151
- Programmierung
 - daten-gesteuerte
 - Wurzel, 31
 - <project>, 348
 - PROLOG, 32, 581, 582, 591
 - Properties, 326
 - <property>, 348
 - PropertyChangeSupport, 316
 - protected, 145, 150, 151
 - Prototyp
 - Konzept, 34
 - PROVIDER_URL, 326
 - Prozedur, 47
 - Prozeduraufruf
 - muster-gesteuerter
 - Wurzel, 31
- PS, 591
- ptjavac, 242
- public, 77, 145, 150, 151
- Purdue University, 338
- Queue, 434, 438
- Rüstzeit
 - terminierbare, 38
- Raggett, David, 583
- Rambaugh, James, 6
- RAS, 591
- Rational, 43
- RCS, 338, 591
- readInt(), 258, 262
- readLine(), 99, 111
- readObject(), 258, 262
- Refactoring, 342, 582
- Reflection, 216–228
- Regel-Orientierung, 39
- Reichweite, 410
 - Begrenzung, 371–375
- Rekursion, 422
- Relation, 50
- RemoteException, 326
- remove(), 297
- removeAll(), 297
- removeAttribute(), 298
- removeChildren(), 297
- removePropertyChangeListener(), 316
- removeVetoableChangeListener(), 316
- Repository, 341
- Requirement, 363
- Resolution, 238
- resolve symbol, 531
- rest, 145
- return, 99, 145
- return, 358
- right, 131
- RISC, 73, 591
- Rivieres, Jim des, 584
- RMI, 23, 265–282, 591
 - URL, 266
- rmic, 266
- rmiregistry, 268
- Robustheit, 26
- RPC, 265, 591
- RRZN, 585
- Rumbaugh, J., 585
- run(), 262
- run(), 114, 344
- Runtime, 20

- SAX, 283, 591
- SAXBuilder, 288, 297
- SAXBuilder(), 570
- Scanner, 193
- SCCS, 338, 591
- Schader, Martin, 585
- schedule(), 114
- Scheme, 581
- Schmidt-Thieme, Lars, 585
- Schnittstelle, 400
- Schriftart
 - Typewriter, 6
- SDK, 20, 21, 591
- SELF, 586
- Semantik, 144–149
- Serializable, 184
- Serialization, 182, 186
- serialVersionUID, 185
- ServerSocket, 262
- Servlet, 20, 588
- Session Beans, 319
- setAttribute(), 298
- setBorder(), 139
- setDefaultCloseOperation(), 137
- setDocType(), 566
- setEncoding(), 566
- setEntityContext(), 324
- setForeground(), 139
- setFormat(), 566
- setLayout(), 137, 139
- setOrientation(), 139
- setRootElement(), 566
- setString(), 139
- setStringPainted(), 139
- Setter, 314
- setValue(), 139
- setVisible(), 137
- SGML, 383, 591
- Shaio, Sami, 583
- Shavor, Sherry, 585
- Shell, 348
- short, 145–147
- Short Running Transaction, 233
- showStatus(), 139
- Sicherheit, 67
- Siedersleben, Johannes, 582
- Simula 67, 582
- Skel, 273
- Skeleton, 257
- Slot, 45
- Smalltalk, 36, 583
- Smiley, 7
- Smith, Randall B., 586
- SMTP, 111, 591
- Socket, 111, 258
- Software Engineering, 585
- Softwareentwicklung
 - Arbeitstechniken, 581
- Sommerville, Ian, 585
- source, 250
- Sparx Systems, 505
- Speicher
 - Heap, 252
- Spezialisierung, 58
- Sprachen
 - imperativ-geprägte, 35
 - objekt-orientierte
 - Ausrichtung, 32
- SQL, 592
- sqrt(), 78
- SSH, 342, 591
- SSL, 22, 592
- standby, 131
- Starbuck, Orca, 583
- stateChanged(), 139
- static, 77, 114, 144, 145
- static{}, 91
- Steele, Guy, 583
- Stereotyp, 49
 - Name, 62
- String, 99
- String.valueOf(), 137
- Stroustrup, Bjarne, 585
- Struktur, 38
- Stub, 257
- stub, 273
- Style Sheet, 384–397
- Suffix, 356
- suite(), 344

- Sun, 585
 - Forte for Java, 73
- Sun Microsystems, 72
- super, 145, 416
- Sussman, Gerald Jay, 581
- Sussman, Julie, 581
- Swing, 344
- switch, 145
- switch, 358
- synchronized, 145, 147, 166
- Syntax, 144–149
- System
 - komplexes, 27
- System.gc(), 88
- System.in.read(), 114
- System.out.println(), 98, 111

- TakeFive Software GmbH, 586
- <target>, 348
- TCP, 592
- TCP/IP, 74
- Teamwork
 - Non-Equals, 581
- TestCase, 344
- TestRunner, 344
- TestSuite, 344
- TEX, 589
- this, 91, 145, 413, 420
- this(), 91
- Thomas, Anne, 313
- Thread, 114, 157–167
 - Beispiel, 443
 - join, 166
 - sleep, 166
- Thread, 262
- thread, 68
- throw, 145
- Throwable, 258, 262
- throws, 114, 145, 258, 262, 376
- Tichy, Walter, 338
- Timer, 114
- TimerTask, 114
- Together
 - Borland Control Center, 89, 102, 116, 120, 135, 140, 143, 158, 171, 179, 187, 299, 305, 426, 457, 460
 - top, 131
 - toString, 144
 - toString(), 81
 - Transaktion, 233
 - long running, 233
 - short running, 233
 - transient, 145, 147, 185, 237
 - Traunmüller, Roland, 586
 - true, 145
 - try, 145, 375
 - Tyma, Paul, 586
 - Typ, 45
 - type, 131

- U2P, 62
- Umgebungsvariable, 576
- UML, 582, 584, 592
 - Anwendung, 585
 - Klassensymbol, 46
 - Patterns, 584
 - Praxis, 585
 - Rational, 585
 - Version2, 584
- UML 2.x, 63
- Ungar, David, 586
- Unicode, 356
- unsetEntityContext(), 324
- URI, 131, 592
- URL, 105, 592
- URL, 193
- URLConnection, 105

- valueOf(), 99
- Vanderburg, Glenn, 586
- var, 145
- Variable, 45, 47, 48
 - interne, 35
 - Name, 61
- Vererbung, 58, 59, 61, 400
 - Implementierung, 122
 - Slot, 432

- Verhalten, 122
- Vererbungsgraph, 35
- Verhaltensvererbung, 122
- Verifier
 - Bytecode, 70
- VERTICAL, 139
- Verwaltungsautomation, 581
- VetoableChangeSupport, 316
- Visio
 - Microsoft, 589
- VisualAge for Java, 73
- void, 77, 145
- volatile, 145, 148

- W3C, 592
- wait, 144, 166
- while, 111, 145
- White, John L., 583
- width, 131
- Wiederverwendung, 400
- Wiesner, Stephan, 8
- Wilcox, Pauline, 585
- Windows
 - DOS, 577
 - IP-Konfiguration, 578
- Windows NT, 73
- Windows XP, 73
- Wirkungsbereich, 401
- Woodfield, Scot N., 582
- WSDL, 592

- XHTML, 6, 382–384, 592
- XMI, 592
- XML, 282, 592
- XMLOutputter(), 566
- Xms, 255
- Xmx, 255
- XPS, 39

- Zeichenkette, 409
- Zeiger, 71
- Zeit
 - Konvertierung, 38
- Zugriffskontrolle
 - Liste, 72
- Zusicherung, 48
 - Name, 62
 - Wert, 249

```
      ' '
      ( )
      ( )
      ( . . )
      ( @ ) -----+-----+
      ( )             | Programmieren |
                       | bleibt       |
                       | schwierig!  |
                       +-----+-----+
      //( ) \\
      //( ) \\
      vv ( ) vv
      ( )
      _//~\\_
      ( ) ( )
```

* * *

Java-Coach