

Oliver Böhm: *Java Software Engineering unter Linux*



Oliver Böhm

Java Software Engineering unter Linux



SuSE PRESS

Alle in diesem Buch enthaltenen Programme, Darstellungen und Informationen wurden nach bestem Wissen erstellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund ist das in dem vorliegenden Buch enthaltene Programm-Material mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren und die SuSE Linux AG übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials, oder Teilen davon, oder durch Rechtsverletzungen Dritter entsteht.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann verwendet werden dürften.

Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Die SuSE Linux AG richtet sich im Wesentlichen nach den Schreibweisen der Hersteller. Andere hier genannte Produkte können Warenzeichen des jeweiligen Herstellers sein.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Druck, Fotokopie, Microfilm oder einem anderen Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Böhm, Oliver:

Java Software Engineering unter Linux / Oliver Böhm. – Nürnberg : SuSE-PRESS, 2002

ISBN 3-935922-53-1

© 2002 SuSE Linux AG, Nürnberg (<http://www.suse.de>)

Umschlaggestaltung: Fritz Design GmbH, Erlangen

Gesamtlektorat: Nicolaus Millin

Fachlektorat: Harald Bertram, Michael Eicks, Michael Hager, Markus Meisters, Stefan Reinauer, Joerg Steffens

Satz: L^AT_EX

Druck: Kösel, Kempten

Printed in Germany on acid free paper.

Geleitwort

Linux ist schon seit geraumer Zeit die Entwicklungsplattform der Wahl für viele (C/C++-)Entwickler, da es von Haus aus bereits viele Programmiersprachen wie Pascal, Perl oder Python und viele Tools wie `grep`, `gcc`, `gdb`, `make`, `lex` & `yacc`, u v. a. m... mitbringt. Mit Java jedoch, als „einfache Programmiersprache für die Masse“, wurde eher unter MS-Windows-Betriebssystemen entwickelt, weil es hier sowohl die besseren Entwicklungsumgebungen als auch die stabilere Laufzeitumgebung gab.

Diese beiden Punkte sind mittlerweile behoben. Das JDK gibt es sowohl von SUN selbst als auch von Drittanbietern (wie beispielsweise IBM oder Blackdown) auch für Linux in hoher Qualität; und da viele der neueren IDEs in Java geschrieben sind, laufen sie auch problemlos unter Linux. Was noch fehlte, war ein schönes Buch, das man sich zu Gemüte führen kann, wenn man mit einer schönen Programmiersprache auf einer tollen Plattform sein Unwesen treiben möchte. Die Angebote und Möglichkeiten sind ja doch recht groß. Wie lange haben wir bei ArgoUML zum Beispiel „make“ eingesetzt, bis irgendwann das viel besser geeignete „Ant“ gefunden wurde. Alle diese (oder zumindest die meisten dieser) Möglichkeiten sind nun in diesem Buch zusammengefasst. Es wird sicher einen Stammplatz auf vielen Schreibtischen finden.

August 2002

Toby Baier
Projektleiter von ArgoUML

Vorwort

Warum dieses Buch? Lange Zeit habe ich unter Unix und Linux entwickelt, hauptsächlich in C und C++. Unix und auch Linux sind Betriebssysteme von Entwicklern für Entwickler. Die Linux-Entwicklung ist gekennzeichnet von Engagement und herausragender Programmierleistung.

Vor knapp drei Jahren bin ich dann ins Java-Lager gewechselt. Leider war diese Entscheidung mit einem Umstieg ins NT-Lager verbunden. Die Situation, die ich dabei vorfand, war: Entwicklung unter NT, aber Zielsystem Unix-Plattform (meist Solaris, manchmal auch Linux). Immerhin haben viele Entscheidungsträger inzwischen erkannt, dass NT als Server-Plattform eher mit „Nice Try“¹ zu übersetzen ist und dass hier Linux seine Stärken ausspielen kann.

Wenn Linux als Zielplattform eingesetzt wird, erhebt sich natürlich die Frage: Warum nicht auch als Entwicklungsplattform? Hier verlässt viele Verantwortliche der Mut, da es ja unter Windows so viele schöne Entwicklungsumgebungen wie Visual J++, Visual Cafe² oder Visual Age gibt, die unter Linux nicht vorhanden sind. Aber es gibt sie. Und es gibt keine Gründe (mehr), die gegen eine Entwicklung unter Linux sprechen. Neben den Bordmitteln, die Linux von Haus aus mitbringt, gibt es alle Tools, die für eine erfolgreiche SW-Entwicklung nötig sind, auch für Linux. Nur Mut, Linux-Entwickler sind motiviertere Entwickler.

Neben dem Ziel, Linux als Java-Plattform salonfähig zu machen, möchte ich aber vor allem einen Überblick über die Java-Szene und den SW-Entwicklungsprozess geben. SW-Entwicklung ist mehr als „nur“ Programmierung und das Zählen von LOCs (Lines Of Code), auch wenn dies manche Chefs und andere Fossilien noch nicht erkannt haben.

Die Codierung und Programmierung ist nur *ein* Teil der verschiedenen Entwicklungs-Phasen. Entsprechend reicht eine schöne IDE allein kaum aus, um erfolgreich zum Ziel zu gelangen. Welche Tools und Hilfsmittel es noch für die Entwicklung gibt, wird u. a. in diesem Buch vorgestellt.

Zielgruppe

Wie sich aus dem Vorwort schon erahnen lässt, richtet sich das Buch an

- ☐ Entscheidungsträger (warum Linux als Entwicklungsplattform geeignet ist)
- ☐ Entwickler (warum Java-Entwicklung mehr als eine gute IDE bedeutet)

¹Aufschrift auf einem OS/2-T-Shirt (s. <http://geekt.org/geekt/comment.cgi?newsid=1136>)

²jetzt WebGain

Ziele

Wie kann man mit Java professionell programmieren? Unter Linux? Nach einigen grundsätzlichen Überlegungen und der Wahl des richtigen „Java Development Kits“ (JDK) werden wir in Teil II feststellen, dass das JDK mit Sicherheit aus mehr als nur dem Compiler (`javac`) und dem Debugger (`jdb`) besteht. Wissen Sie, dass Ihr ausgeliefertes Java-Programm Rückschlüsse auf die Java-Sourcen zulässt? Dann dürfte das Kapitel „Decompiler und Obfuscator“ für Sie interessant sein.

„*A fool with a tool is still a fool*“ (Scott Ambler). Aus diesem Grund werden im dritten Teil nicht nur Tools für die verschiedenen Phasen der SW-Entwicklung vorgestellt, sondern anhand von Beispielen gezeigt, wie diese Tools sinnvoll eingesetzt werden. Sie werden feststellen, dass SW-Entwicklung weit mehr als nur aus der Codier-Phase besteht.

Java hat sich in kurzer Zeit von einer *coolen* Programmiersprache zu einer Basis für unternehmenskritischen Anwendungen hochkatapultiert, die hauptsächlich im Server-Bereich zum Einsatz kommt. Um hier mitsprechen und mitentwickeln zu können, werden die verschiedenen Techniken in „Enterprise-Java“ vorgestellt und durch Beispiele verdeutlicht. Aber auch das Testen kommt nicht zu kurz, ist es doch durch *eXtreme Programming* (XP) in der Vordergrund gerückt.

Reflexion ist ein Thema für den fortgeschritteneren Java-Entwickler, während JNI eher für den Entwickler im Embedded Bereich interessant ist. Aber nicht nur, wie uns das Beispiel aus diesem Kapitel lehrt.

Wer schon mal fremden Java-Code übernehmen musste, wird sich freuen, wenn dieser sauber dokumentiert und lesbar gestaltet ist. Leider sieht die Realität etwas anders aus. Deswegen finden sich im Teil VII einige Anregungen und Tools, um im Vorfeld dem Wildwuchs etwas Einhalt zu gebieten.

Wenn ein Programm fertig entwickelt ist, gibt es mehrere Möglichkeiten, dieses an den Mann oder die Frau zu bringen. Aber eigentlich ist ein Programm nie fertig, dafür sorgen schon die Programmierfehler, die sich vorzugsweise erst nach der Auslieferung zeigen. Hier können Trouble-Ticket- bzw. Bug-Tracking-Systeme helfen, die Flut an Fehler-Meldungen in die richtigen Bahnen zu lenken.

Die Erlernung von Java gehört *nicht* zu den Zielen dieses Buches. Wer Java-Anfänger ist oder seine Kenntnisse noch einmal auffrischen will, findet auf der beiliegenden CD-ROM die Java Schulungsunterlagen (mit Aufgaben und Musterlösungen) von Hubert Partl, die sehr gut zum Selbststudium geeignet sind.

Voraussetzungen

- ☐ Java-Kenntnisse
- ☐ Lust am Lesen
- ☐ Lust am Ausprobieren

Fachbegriffe

Bei der Übersetzung mancher Fachbegriffe habe ich mich schwer getan. Oft ist mir der deutsche Begriff überhaupt nicht geläufig und völlig fremd. In diesen Fällen habe ich den englischen Fachbegriff einfach übernommen. Eine Eindeutigung würde meiner Meinung nach den Leser eher verwirren.³ In Fällen, wo mir der deutsche Begriff ungewohnt vorkam, habe ich den englischen Begriff in Klammern dazugeschrieben. Sollte der eine oder andere englische Begriff zuviel auftauchen, hoffe ich, dass Sie mir verzeihen.

Warnung

Achtung, Windows-Umsteiger: Linux erschließt sich nicht von selbst. Man muss sich schon damit beschäftigen, wenn man den größtmöglichen Nutzen daraus ziehen will. Und nicht immer wird alles so funktionieren, wie Sie es sich vorstellen. Da hilft dann keine Neuinstallation oder Booten, sondern nur lesen, lesen, lesen... Aber die Chancen stehen danach gut, dass Sie Ihren Rechner beherrschen und nicht umgekehrt.

Wer Angst vor der englischen Sprache hat, sollte diese Angst ablegen oder auf dieses Buch verzichten. Viele Code-Beispiele verwenden englische Variablen und Kommentare. Die deutsche Sprache ist leider für die Programmierung ungeeignet – zu unhandlich, zu lange Wörter, zu viel Grammatik. Englisch ist viel besser für die Software-Entwicklung geeignet. Ich habe sechs Jahre Studium gebraucht, um dies einzusehen. Ich hoffe, Sie sind einsichtiger.

Danksagungen

Hier nun die obligatorische Danksagung an all jene Personen, die mir dieses Skript ermöglicht haben: An meine Frau *Cordula* und meine Kinder *Manuel*, *Philip* und *Carmen* für die moralische Unterstützung und ihr Verständnis, dass ich nicht immer ansprechbar war. Ich hoffe, dass ich nun wieder mehr Zeit für sie

³Beispiel: wenn eine „Exception“ geworfen wird, weiß jeder, was damit gemeint ist. Mit „Ausnahmen“ schmeißen die wenigsten Java-Programmierer um sich.

habe. Danken möchte ich meinem Schwiegervater, Herrn *Karl Muffler*, für sein Durchhaltevermögen, die Unterlagen auf Rechtschreibfehler und stilistische Entgleisungen durchzusehen, obwohl die ganzen Themen böhm'sche Dörfer für ihn waren.

Über einen ArgoUML-Beitrag in der Zeitschrift „Linux Enterprise“ habe ich *Toby Baier* kennengelernt. Seine Nachsicht mit meinem Artikel und seine freundliche Art sorgten dafür, dass ich mich intensiver mit ArgoUML (s. Kap. 9.4) befasste. Später habe ich ihn nach einem Vortrag über Open Source auf dem Java Forum Stuttgart persönlich kennengelernt, wobei ich ihn für das Vorwort gewinnen konnte.

Dank geht auch an *Artur Lojewski*, der sich während unserer gemeinsamen Zusammenarbeit als wandelndes Linux-Lexikon und verlässlicher Kollege erwiesen hat. Zeichneten sich neue und vielversprechende Projekte im OpenSource-Bereich ab, wusste er bereits lange vorher darüber Bescheid, ehe ich es wahrnahm. Ihm haben Sie das Kapitel zu Eclipse (Kap. 11.5) zu verdanken.

Auch wenn dieses Buch in der Freizeit entstanden ist, konnte ich manches aus der täglichen Arbeit bei der Fiducia AG in mein Buch übernehmen (z. B. Kap. 17, das im Rahmen einer EJB-Schulung entstanden ist). Hier möchte ich mich bei meinen Kollegen und Vorgesetzten für die Toleranz gegenüber meiner Linux-Insel inmitten einer OS/2- und Windows-dominierten Umgebung bedanken. Stellvertretend erwähnen möchte ich den Entwicklungsleiter *Wolfgang Clauss*, der Linux gegen interne Widerstände hoffähig gemacht hat und für einen Manager außerordentlich fortschrittliche Ideen besitzt und vertritt.

Herr *Nicolaus Millin* von SuSE PRESS sorgte dafür, dass ich Sie, den Leser, beim Schreiben nicht aus den Augen verlor, auch wenn es für mich einiges an zusätzlicher Arbeit bedeutet hat. Ihm und den „Probe“-Lesern, *Harald Bertram*, *Michael Eicks*, *Michael Hager*, *Markus Meisters*, *Stefan Reinauer*, *Joerg Steffens*, gebührt mein Dank für die konstruktive Kritik.

Zum Schluss möchte ich allen danken, die Linux zu dem gemacht haben, was es heute ist, ebenso den vielen engagierten Entwicklern in OpenSource-Projekten, ohne deren Enthusiasmus und Einsatz wir der Dominanz einzelner SW-Hersteller ausgeliefert wären. Ohne sie wäre die Computerwelt um einiges ärmer und dieses Buch nur halb so dick.

Und ganz zum Schluss sei all denen gedankt, die mein Buch weiterempfehlen werden.

Gerlingen, im August 2002

Oliver Böhm

Inhaltsverzeichnis

I	Einleitung	1
1	Glaubenskriege	5
1.1	Lernziele	5
1.2	Warum Linux?	5
1.3	Die Geschichte von Java	6
1.3.1	Java – das bessere C++	7
1.3.2	Garbage Collection	8
1.3.3	JDK 1.eins, zwei, drei, ...	8
1.3.4	Java-Plattformen	9
1.3.5	Was geht nicht mit Java?	9
1.4	Entscheidungshilfe für...	10
1.4.1	Entwickler	10
1.4.2	Manager	11
1.4.2.1	Konfiguration	11
1.4.2.2	Sicherheit und Viren	12
1.4.2.3	Herstellerunabhängigkeit	12
1.4.2.4	Besserer Nachwuchs	13
1.4.2.5	Vielseitigere Mitarbeiter	13
1.4.2.6	Motiviertere Mitarbeiter	13
1.4.3	Kühle Rechner	14
1.4.3.1	Wirtschaftlichkeit	14

1.4.3.2	Software		14
1.4.3.3	TCO und Administration		14
1.4.4	Träumer		15
1.5	Gegenargumente, die keine sind		15
1.5.1	Kein Support		15
1.5.2	Schwierige Installation?		16
1.6	Warum Windows?		17
1.6.1	Technische Gründe		17
1.6.1.1	Fehlende HW-Unterstützung		17
1.6.1.2	Fehlende SW		17
1.6.1.3	Zielmarkt		17
1.6.2	Wirtschaftliche Gründe		18
1.6.2.1	Altlasten		18
1.6.2.2	Schulung		18
1.6.3	Politische Gründe		19
1.6.3.1	Management-Entscheidung		19
1.6.3.2	Abhängigkeiten		19
1.7	Weitere Informationsquellen		19
1.8	Zusammenfassung		20
2	Java-Installation		23
2.1	Lernziele		23
2.2	Begriffsbestimmung		23
2.3	Anforderungen an die...		25
2.3.1	...Hardware		25
2.3.2	...Software		26
2.4	Welche JVM?		26
2.4.1	JDK 1.2		26
2.4.2	IBM-JDK 1.1.8		27
2.4.3	IBM-JDK 1.3		27
2.4.4	Sun-JDK 1.3		28
2.4.5	Sun-JDK 1.4		28
2.4.6	JIT-Compiler		30

2.4.7	HotSpot-Technologie	30
2.4.8	Benchmarking	31
2.4.8.1	CaffeineMark	31
2.4.9	Kaffe OpenVM	34
2.4.10	Japhar	37
2.4.11	GNU Classpath	37
2.4.12	Open Runtime Platform (ORP)	38
2.5	Welcher Compiler?	40
2.5.1	jikes	40
2.5.1.1	Jikes-Optionen	41
2.5.1.2	Abhängigkeiten	42
2.5.1.3	Fazit	43
2.5.1.4	Jikes Debugger (JD)	43
2.5.2	GNU-Java-Compiler (GCJ)	44
2.5.2.1	Installation	44
2.5.2.2	Experimentelle Installation	45
2.5.2.3	Beispiel	45
2.5.2.4	gcj-Optionen	45
2.5.2.5	Übersetzung von Class-Dateien	46
2.5.2.6	Einschränkungen	46
2.6	Welche Bibliotheken	47
2.6.1	Optional Packages	47
2.6.2	Enterprise Technologien	49
2.6.3	JavaMail	51
2.6.4	Weitere Bibliotheken	53
2.7	Weitere Informationsquellen	53
2.8	Zusammenfassung	54
3	XML – eine kurze Einführung	55
3.1	Lernziele	55
3.2	Hype or Hope	55
3.2.1	HTML – der kleine Bruder	55
3.2.2	SGML – der große Urahn	57

3.2.3	Was ist XML?	57
3.3	<ein_Beispiel/>	58
3.3.1	Attribute	60
3.3.2	Weitere Elemente	60
3.3.3	Anwendungsgebiete	61
3.3.4	XHTML	62
3.3.5	Der Stammbaum im Überblick	63
3.4	XSL-Stylesheets	63
3.4.1	Stylesheet-Einbindung	63
3.4.2	XSL-Beispiel	64
3.4.3	XSL-Transformation (XSLT)	68
3.5	DTD	69
3.5.1	Eine Buch-DTD	70
3.5.2	DTD-Einbindung	72
3.5.3	Dokument-Typen	73
3.6	Java und XML	73
3.6.1	XML-Parser	73
3.6.2	XML-API	74
3.6.3	Xerces	75
3.6.4	Java Architecture for XML Binding (JAXB)	76
3.7	XML-Editoren	76
3.7.1	Amaya	77
3.7.2	XXE	78
3.7.3	Weitere XML-Editoren	79
3.8	Aussichten	79
3.9	Weitere Informationsquellen	80
3.10	Zusammenfassung	80

II Java Development Kit 81

4 JDK & JVM 85

4.1	Lernziele	85
-----	-----------	----

4.2	Der Class-Loader	85
4.2.1	Umgebungsvariablen	86
4.3	Zeichensätze	87
4.3.1	Konfiguration	87
4.3.2	Zeichensatz-Suche	89
4.3.3	Font-Path	90
4.4	JIT-Compiler	90
4.5	Security	90
4.6	Threads	91
4.6.1	Green Threads versus Native Threads	91
4.6.2	Nebenwirkungen	91
4.6.3	Prozess	91
4.7	Java-Optionen	92
4.7.1	Standard-Optionen	92
4.7.2	Weitere Optionen	94
4.8	Umgebungsvariablen...	95
4.8.1	...für Java	95
4.8.2	...für Linux	96
4.9	JNI-Bibliotheken	97
4.10	Java Runtime Environment (JRE)	97
4.11	javac – der Java-Compiler	97
4.11.1	Standard-Optionen	98
4.11.2	Cross-Compiler-Optionen	99
4.11.3	Nicht-Standard-Optionen	99
4.11.4	VM-Optionen	99
4.12	jdb – der Java Debugger	100
4.12.1	Normaler Start	100
4.12.2	Andock-Manöver	100
4.12.3	Applets	100
4.12.4	jdb-Optionen	101
4.12.5	jdb-Kommandos	101
4.12.5.1	Allgemeine Kommandos	101

4.12.5.2	Haltepunkte (Breakpoints)		102
4.12.5.3	Anzeige		102
4.12.5.4	Exceptions		103
4.12.5.5	Threads		103
4.12.5.6	Sourcen		103
4.12.5.7	Ressourcen		104
4.12.6	Java Platform Debugging Architecture (JDBA)		104
4.13	Weitere Informationsquellen		105
4.14	Zusammenfassung		105
5	Java-Tools		107
5.1	Lernziele		107
5.2	Java-Archivierer (jar)		107
5.2.1	Aufruf		108
5.2.2	Standard-Optionen		109
5.2.3	zusätzliche Optionen		110
5.2.4	Manifest-Datei		110
5.2.4.1	Bedeutung		110
5.2.4.2	Aufbau		111
5.2.4.3	Hauptabschnitt		111
5.2.4.4	Weitere Abschnitte		112
5.2.4.5	Paket-Versionierung		112
5.2.5	Midnight Commander		112
5.3	Disassembler (javap)		113
5.3.1	Optionen		114
5.3.2	VM-Optionen		114
5.3.3	HelloWorld durchleuchtet		115
5.4	Zeichen-Konvertierung (native2ascii)		116
5.5	Die Seriennummer (serialver)		116
5.6	Extension Checker (extcheck)		117
5.7	Weitere Informationsquellen		118
5.8	Zusammenfassung		118

6	Sicherheit	119
6.1	Lernziele	119
6.2	Das Sicherheitsmodell	119
6.2.1	Sandkastenspiele	119
6.2.2	Java2-Sicherheits-Modell	121
6.2.3	Der Security-Manager	122
6.3	Ein signiertes Applet	122
6.3.1	CounterApplet	122
6.3.2	AccessControlException	124
6.3.3	Signierung	124
6.3.4	Der nächste Anlauf	125
6.4	Die Policy-Datei	126
6.4.1	Das Format der Policy-Datei	126
6.4.2	grant-Einträge	127
6.4.3	Properties in Policy-Dateien	130
6.4.4	Erstellen der Policy-Datei	130
6.4.5	policytool	131
6.5	Security-Manager für Java-Programme	131
6.6	keytool	132
6.6.1	Export/Import von Zertifikaten	132
6.6.2	Syntax	133
6.6.3	Kommandos	133
6.6.4	Optionen	134
6.7	jarsigner	135
6.7.1	Optionen	136
6.8	Weitere Informationsquellen	136
6.9	Zusammenfassung	137
7	Decompiler und Obfuscator	139
7.1	Lernziele	139
7.2	Java Decompiler (jad)	139
7.2.1	Installation	140
7.2.2	Optionen	140

7.2.3	Beispiel	141
7.3	Obfuscator	144
7.3.1	RetroGuard	145
7.3.2	Vorbereitung	145
7.3.3	Die Verschleierung	145
7.3.4	Schutz vor Manipulationen	147
7.3.5	Weitere Obfuscatoren	147
7.3.6	Probleme	148
7.4	Weitere Informationsquellen	148
7.5	Zusammenfassung	148

III Entwicklung 149

8 Versionierung 153

8.1	Lernziele	153
8.2	Grundlagen	153
8.2.1	Wozu überhaupt?	153
8.2.2	Aufgaben eines Versionierungssystems	154
8.2.3	Release-Verwaltung	155
8.2.4	Parallel-Entwicklung	155
8.2.4.1	Lock-Strategien	156
8.2.5	Kleinere Projekte	156
8.3	RCS	157
8.3.1	Die erste Version	157
8.3.2	Die nächste Version	158
8.3.3	Schlüsselwörter	159
8.3.4	Kennzeichnungspflicht	161
8.3.5	The Next Generation	162
8.3.6	Seitensprünge	162
8.3.7	RCS-Kommandos	163
8.4	CVS	166
8.4.1	Installation	166

8.4.2	Bedeutung	166
8.4.2.1	Vorteile	167
8.4.2.2	Nachteile	167
8.4.3	CVS-Kommandos	167
8.4.3.1	Überblick	167
8.4.3.2	Optionen	169
8.4.3.3	Umgebungsvariablen	170
8.4.3.4	Hilfe	170
8.4.4	Projekt aufsetzen	171
8.4.4.1	Das Repository	172
8.4.4.2	Import	172
8.4.5	Der erste Checkout	174
8.4.6	Der erste Checkin	175
8.4.7	Elemente hinzufügen	176
8.4.8	Binäre Dateien	177
8.4.9	Dateien löschen	177
8.4.10	Dateien umbenennen	178
8.4.11	CVS im Team	179
8.4.11.1	Update	179
8.4.11.2	Konflikte	180
8.4.11.3	cvs release	182
8.4.12	Etikettierung	182
8.4.13	Verzweigungen (branches)	184
8.4.13.1	Sticky Tags	184
8.4.13.2	Die Wiedervereinigung (merge)	186
8.4.14	Einige Tipps	188
8.4.14.1	Stabile Versionen	188
8.4.14.2	Regelmäßige Updates	188
8.4.14.3	Lokale Änderungen verwerfen	188
8.5	CVS im Netz	189
8.5.1	Lokale Netze	189
8.5.2	Größere Netze	190

8.5.2.1	Weitere Mitarbeiter	.	.	.	.	.	.	191
8.5.2.2	CVS und SSH	.	.	.	.	.	.	192
8.6	CVS-GUI	.	.	.	.	.	.	194
8.6.1	tkCVS	.	.	.	.	.	.	194
8.6.2	jCVS	.	.	.	.	.	.	195
8.6.2.1	Installation	.	.	.	.	.	.	196
8.6.2.2	Ein erster Test	.	.	.	.	.	.	196
8.6.2.3	Projekt-Öffnung	.	.	.	.	.	.	197
8.6.2.4	jCVS und SSH	.	.	.	.	.	.	198
8.6.3	Cervisia	.	.	.	.	.	.	198
8.6.3.1	Erfahrungen	.	.	.	.	.	.	198
8.6.4	pharmacy	.	.	.	.	.	.	200
8.6.5	Weitere Frontends	.	.	.	.	.	.	201
8.7	Weitere Informationsquellen	.	.	.	.	.	.	201
8.8	Zusammenfassung	.	.	.	.	.	.	202
9	Analyse und Design							203
9.1	Lernziele	.	.	.	.	.	.	203
9.2	Der SW-Lifecycle	.	.	.	.	.	.	203
9.2.1	Analyse	.	.	.	.	.	.	204
9.2.2	Design	.	.	.	.	.	.	205
9.2.3	Implementierung und Testen	.	.	.	.	.	.	205
9.3	UML (Unified Modelling Language)	.	.	.	.	.	.	205
9.3.1	Diagrammtypen	.	.	.	.	.	.	206
9.3.2	UML-Tools	.	.	.	.	.	.	207
9.4	Eine Fallstudie mit ArgoUML	.	.	.	.	.	.	207
9.4.1	Ausgangslage	.	.	.	.	.	.	207
9.4.2	Überblick über ArgoUML	.	.	.	.	.	.	208
9.4.3	Anwendungsfälle (Use Cases)	.	.	.	.	.	.	209
9.4.4	Klassendiagramme (Class Diagrams)	.	.	.	.	.	.	211
9.4.4.1	Fein-Design	.	.	.	.	.	.	214
9.4.4.2	Code-Generierung	.	.	.	.	.	.	214
9.4.5	Paketdiagramme	.	.	.	.	.	.	215

9.4.6	Interaktionsdiagramme	.	.	.	.	.	.	217
9.4.6.1	Sequenz-Diagramm	.	.	.	.	.	.	217
9.4.6.2	Kollaborations-Diagramm	.	.	.	.	.	.	221
9.4.7	Zustandsdiagramme	.	.	.	.	.	.	222
9.4.8	Aktivitätsdiagramme	.	.	.	.	.	.	226
9.4.9	Verteilungsdiagramme	.	.	.	.	.	.	230
9.4.10	Verbindlichkeiten	.	.	.	.	.	.	232
9.4.11	Ist UML nur was für Weicheier?	.	.	.	.	.	.	232
9.5	Entwickeln mit ArgoUML	.	.	.	.	.	.	233
9.5.1	Der Navigations-Bereich	.	.	.	.	.	.	233
9.5.2	Der Diagramm-Bereich	.	.	.	.	.	.	233
9.5.2.1	Rapid Buttons	.	.	.	.	.	.	234
9.5.2.2	Elemente entfernen und löschen	.	.	.	.	.	.	234
9.5.3	Cognitiver Support	.	.	.	.	.	.	235
9.5.4	Detail-Bereich	.	.	.	.	.	.	236
9.5.5	Reverse Engineering	.	.	.	.	.	.	236
9.5.6	Datei-Formate	.	.	.	.	.	.	237
9.5.6.1	Ablage	.	.	.	.	.	.	237
9.5.6.2	Graphik-Formate	.	.	.	.	.	.	237
9.5.7	Fazit	.	.	.	.	.	.	238
9.6	Freie UML-Tools	.	.	.	.	.	.	238
9.6.1	Fujaba	.	.	.	.	.	.	239
9.6.1.1	Story Driven Modeling (SDM)	.	.	.	.	.	.	240
9.6.1.2	Klassendiagramme	.	.	.	.	.	.	241
9.6.1.3	Reverse Engineering	.	.	.	.	.	.	242
9.6.1.4	... And Back Again?	.	.	.	.	.	.	243
9.6.1.5	Die Ablage	.	.	.	.	.	.	244
9.6.1.6	Fazit	.	.	.	.	.	.	244
9.6.2	kUML	.	.	.	.	.	.	245
9.7	Kommerzielle UML-Tools	.	.	.	.	.	.	245
9.7.1	Together	.	.	.	.	.	.	245
9.7.1.1	Plattformen	.	.	.	.	.	.	246

	9.7.1.2	Installation	.	.	.	.	.	.	246
	9.7.1.3	Reverse Engineering	.	.	.	.	.	.	246
	9.7.1.4	Erweiterungen	.	.	.	.	.	.	247
	9.7.1.5	Erfahrungswerte	.	.	.	.	.	.	248
	9.7.2	Weitere UML-Tools	.	.	.	.	.	.	248
	9.7.2.1	Rational Rose	.	.	.	.	.	.	249
	9.7.2.2	Innovator	.	.	.	.	.	.	249
9.8	Weitere Informationsquellen		.	.	.	.	.	.	250
9.9	Zusammenfassung		.	.	.	.	.	.	251
10 Codierung									253
10.1	Lernziele		.	.	.	.	.	.	253
10.2	Welcher Editor		.	.	.	.	.	.	253
	10.2.1	vi	.	.	.	.	.	.	253
	10.2.2	NEdit	.	.	.	.	.	.	255
10.3	Debugger		.	.	.	.	.	.	255
	10.3.1	jdb	.	.	.	.	.	.	256
	10.3.2	Data Display Debugger (DDD)	.	.	.	.	.	.	256
	10.3.2.1	Der Start	.	.	.	.	.	.	257
	10.3.2.2	Erfahrungen	.	.	.	.	.	.	258
10.4	(X)Emacs		.	.	.	.	.	.	259
	10.4.1	Historisches	.	.	.	.	.	.	260
	10.4.2	Emacs ist eine Religion	.	.	.	.	.	.	260
	10.4.3	Emacs ist ein Betriebssystem	.	.	.	.	.	.	261
	10.4.4	Eignung für Java	.	.	.	.	.	.	262
	10.4.5	JDE	.	.	.	.	.	.	263
	10.4.5.1	Installation	.	.	.	.	.	.	264
	10.4.5.2	Konfiguration	.	.	.	.	.	.	265
	10.4.5.3	Bean Shell	.	.	.	.	.	.	266
	10.4.5.4	Erfahrungswerte	.	.	.	.	.	.	266
10.5	Zusammenfassung		.	.	.	.	.	.	267
11 IDE									269
11.1	Lernziele		.	.	.	.	.	.	269

11.2	NetBeans	269
11.2.1	Umfang	270
11.2.2	Installation	270
11.2.3	Update	271
11.2.4	Einstieg	271
11.2.5	Erfahrungswerte	273
11.3	Sun ONE Studio	273
11.3.1	Installation	274
11.3.2	Deinstallation	274
11.3.3	Erfahrungswerte	274
11.3.4	Aussichten	275
11.4	JBuilder	275
11.4.1	Voraussetzungen	275
11.4.2	Editionen	276
11.4.3	Eigenschaften	276
11.4.4	Installation	277
11.4.5	Austausch der VM	279
11.4.6	On Tour	279
11.4.7	Das erste Projekt	280
11.4.8	GUI-Erstellung	282
11.4.9	Der Debugger	283
11.4.10	UML	286
11.4.11	Javadoc-Generierung	287
11.4.12	Erfahrung	288
11.5	Die Eclipse Plattform	288
11.5.1	Einführung	288
11.5.2	Installation	289
11.5.3	Hello Eclipse	290
11.5.3.1	Der Debugger	291
11.5.4	Plug-Ins	293
11.5.5	Workspaces	294
11.5.6	Die Workbench und die UI-Toolkits	294

11.5.6.1	Das Standard-Widget-Toolkit (SWT)	294
11.5.6.2	JFace	295
11.5.6.3	Die Workbench	295
11.5.6.4	Integration von Werkzeugen	297
11.5.6.5	Team Support	297
11.5.7	Eclipse für Fortgeschrittene	297
11.5.8	Das Hilfe-System	298
11.5.9	Eigene Erfahrungen	298
11.6	Weitere Informationsquellen	299
11.7	Zusammenfassung	299
12	Der Build-Prozess	301
12.1	Lernziele	301
12.2	Wozu?	301
12.2.1	Daily Build	302
12.2.2	Smoke Test	302
12.3	make	302
12.3.1	Ein einfaches Makefile	303
12.3.2	Regel-Werk	303
12.3.3	Abhängigkeiten	303
12.3.4	Ein einfaches Beispiel	304
12.3.5	Pseudo-Targets	306
12.3.6	Makros	306
12.3.6.1	Syntax	306
12.3.6.2	Makro-Übergabe	307
12.3.6.3	Umgebungs-Variablen	307
12.3.6.4	Prioritäts-Regeln	307
12.3.6.5	Substitutions-Regeln	307
12.3.6.6	Interne Makros	308
12.3.7	Suffix-Regeln	310
12.3.7.1	Was sind Suffix-Regeln?	310
12.3.7.2	Null-Suffix-Regeln	311
12.3.7.3	Eingebaute Suffix-Regeln	311

12.3.8	Kommandos	311
12.3.8.1	Wildcards (Jokerzeichen)	311
12.3.8.2	Zeilenende	312
12.3.8.3	Skript-Programmierung	312
12.3.8.4	Fehlercode	313
12.3.8.5	Ausgabe unterdrücken	313
12.3.9	Projekt-Management	313
12.3.9.1	Dummy-Targets	314
12.3.9.2	Rekursives make	314
12.3.10	Richtlinien	315
12.3.10.1	Allgemeine Konventionen	315
12.3.10.2	Kommandos	315
12.3.10.3	Installation	316
12.3.10.4	Standard Targets	318
12.3.11	make-Problematik	318
12.4	jmk	320
12.4.1	Installation	321
12.4.2	Beispiel	321
12.5	Weitere Informationsquellen	321
12.6	Zusammenfassung	321
13	Ant	323
13.1	Lernziele	323
13.2	Ant – ein Make-Ersatz?	324
13.2.1	Eigenschaften	324
13.2.2	Installation	325
13.3	Ein einfaches Beispiel	326
13.4	Aufbau einer Build-Datei	327
13.5	Targets	328
13.6	Properties	329
13.6.1	Property-Dateien	330
13.6.2	Built-in Properties	331
13.6.3	System-Properties	332

13.6.4	Spezielle Properties		332
13.7	Path, Classpath und Referenzen		333
13.7.1	Filesets		333
13.7.2	Pattern und Patternsets		334
13.8	Ant-Tasks		336
13.8.1	Kompilation		336
13.8.2	Löscharbeiten		338
13.8.3	Javadoc-Generierung		339
13.8.4	Test-Unterstützung		342
13.8.5	Archive erstellen		346
13.9	Task-Konzepte		348
13.9.1	Allgemeine Attribute		348
13.9.2	Token-Filter		349
13.9.3	Kommandozeilen-Argumente		349
13.10	Ant-Optionen		350
13.11	Ausgabe		351
13.12	build.xml aufgeteilt		352
13.12.1	Aufteilung per XML		352
13.12.2	Systemabhängige Properties		353
13.13	Ant-Tipps und -Richtlinien		353
13.13.1	Ant-Hilfen		354
13.13.2	Verzeichnis-Struktur		354
13.13.3	Targets		355
13.13.4	Plattform-Unabhängigkeit		356
13.14	Praxis-Erfahrungen		356
13.15	Weitere Informationsquellen		356
13.16	Zusammenfassung		357
IV	Enterprise-Java		359
14	RMI		363
14.1	Lernziele		363

14.2	Was kann RMI?	363
14.2.1	Das RMI-Prinzip	364
14.2.2	Stub-/Skeleton-Generator <code>rmic</code>	364
14.2.3	Einschränkungen	364
14.3	Romeo und Julia	365
14.3.1	1. Akt: Erstellung eines Remote-Objekts	365
14.3.2	2. Akt: Registrierung	366
14.3.3	3. Akt: Stub und Skeleton treten auf	367
14.3.4	4. Akt: Der Zugriff	368
14.4	Vergleich zu CORBA	371
14.5	Fehlerquellen	371
14.6	Weitere Informationsquellen	372
14.7	Zusammenfassung	372
15	JDBC	373
15.1	Lernziele	373
15.2	Datenbanken für Linux	373
15.2.1	MySQL	374
15.2.2	Aufsetzen einer MySQL-Datenbank	375
15.2.3	phpMyAdmin	376
15.3	Relationale Datenbanken	377
15.3.1	Tabellen	377
15.3.2	Beziehungen	378
15.3.3	Begriffe	378
15.3.4	Die wichtigsten SQL-Befehle	378
15.3.4.1	Anlegen und Löschen	378
15.3.4.2	Ansicht	379
15.3.4.3	Update	379
15.4	Was ist JDBC?	380
15.4.1	Die JDBC-Teile	380
15.4.2	JDBC-Treiber	380
15.4.3	JDBC-Konformität	382
15.4.4	SQL-Stufe	382

15.5 Grundstruktur	382
15.5.1 Treiber laden	383
15.5.2 Verbindung zur Datenbank	383
15.5.3 Anweisung erzeugen	384
15.5.4 Anweisung ausführen	384
15.5.5 Ergebnis lesen	384
15.5.6 Datenbank schließen	385
15.6 Datentypen	385
15.7 Weitere JDBC-Befehle	385
15.7.1 Merksregel	385
15.7.2 Prepared Statement	386
15.7.3 CallableStatement	387
15.8 Fehlerbehandlung	387
15.8.1 SQLException	387
15.8.2 SQLWarning	387
15.9 Meta-Daten	388
15.10 Transaktionen	388
15.11 JDBC-Tuning	389
15.12 Weitere Informationsquellen	391
15.13 Zusammenfassung	391
16 Java im Web	393
16.1 Lernziele	393
16.2 Das World Wide Web	394
16.2.1 WWWie alles begann	394
16.2.2 Die Dynamisierung des Webs	395
16.2.3 HTTP-Sessions	396
16.2.3.1 Cookies	396
16.2.3.2 URL-Rewriting	397
16.3 Client-seitiges Java	397
16.3.1 Applets	397
16.3.2 Hello-World-Applet	397
16.3.3 Appletviewer	398

16.4	Servlets	399
16.4.1	Das Servlet-Prinzip	399
16.4.2	Hello-World-Servlet	400
16.5	Servlet-Implementierungen	401
16.5.1	Tomcat	401
16.5.2	Installation	402
16.5.3	Ein erster Test	403
16.5.4	Das erste Servlet	403
16.5.5	WAR-Archiv	405
16.6	Servlets und Formulare	406
16.6.1	Parameter-Übergabe	406
16.6.2	Get-Request	406
16.6.3	Post-Request	408
16.6.4	Session-Realisierung	408
16.6.5	Weitere Einsatzgebiete	409
16.6.6	Vor- und Nachteile	409
16.6.6.1	Vorteile	409
16.6.6.2	Nachteile	409
16.6.7	Weiterführende Literatur	410
16.7	JavaServer Pages (JSP)	410
16.7.1	JSP-Implementierungen	410
16.7.2	Das JSP-Prinzip	410
16.7.3	Ein erstes JSP-Beispiel	411
16.7.4	System-JSP	412
16.7.5	Die JSP-Elemente	414
16.7.6	JSP-Direktiven	414
16.7.6.1	Include-Direktive	414
16.7.6.2	Page-Direktive	415
16.7.6.3	Taglib-Direktive	416
16.7.7	JSP-Tags	416
16.7.7.1	useBean	417
16.7.7.2	setProperty/getProperty	417

16.7.7.3	include		417
16.7.7.4	forward		418
16.7.7.5	plugin		418
16.7.8	Skript-Code		418
16.7.9	Implizit vorhandene Objekte		418
16.7.10	Weiterführende Literatur		419
16.8	JSP- und Servlet-Architekturen		419
16.8.1	Einfache Anwendung		419
16.8.1.1	Vorteile		419
16.8.1.2	Nachteile		419
16.8.2	Komplexere Anwendung		420
16.8.2.1	Vorteile		420
16.8.2.2	Nachteile		420
16.8.3	Servlets oder JSP?		420
16.8.4	Session-Realisierung mit Servlets und JSPs		421
16.8.5	Erfahrungswerte		422
16.9	Weitere Informationsquellen		423
16.10	Zusammenfassung		424
17	EJB		425
17.1	Lernziele		425
17.2	Was sind EJBs?		425
17.2.1	Standalone-Anwendungen		426
17.2.2	2-Tier-Architekturen		426
17.2.3	3- und n-Tier Architektur		427
17.2.4	Was bieten EJBs		428
17.2.5	Das EJB-Prinzip		429
17.2.6	Bean-Typen		430
17.2.6.1	Stateless Session Beans		430
17.2.6.2	Stateful Session Beans		431
17.2.6.3	Entity Bean		432
17.2.7	Deployment		433
17.2.8	Rollenspiel		434

17.3	Kochrezept	435
17.3.1	Bean-Klassen und Interfaces	435
17.3.2	Hello-Stateless-Session-Bean	436
17.3.2.1	Remote-Interface (Hello.java)	436
17.3.2.2	Home-Interface (HelloHome.java)	436
17.3.2.3	Bean-Klasse (HelloBean.java)	436
17.3.2.4	Deployment	438
17.3.2.5	Der Client	439
17.3.3	Flip-Flop-Stateful-Session-Bean	440
17.3.3.1	Remote-Interface (FlipFlop.java)	440
17.3.3.2	Home Interface (FlipFlopHome.java)	440
17.3.3.3	Bean-Klasse (FlipFlopBean.java)	440
17.3.3.4	Deployment	442
17.3.3.5	Der FlipFlop-Client	442
17.3.4	Account-Bean (CMP)	443
17.3.4.1	Remote-Interface (Account.java)	443
17.3.4.2	Primary Key (AccountPK.java)	443
17.3.4.3	Home-Interface (AccountHome.java)	444
17.3.4.4	Bean-Klasse (AccountBean.java)	445
17.3.4.5	Deployment	446
17.3.5	Der Account-Client	447
17.3.6	Account Bean (BMP)	449
17.3.6.1	Bean-Klasse (AccountBean.java)	450
17.3.6.2	Deployment	453
17.3.6.3	Client	454
17.3.7	Andere Datenbanken	454
17.4	Transaktionen	454
17.4.1	Steuerung	456
17.4.2	JTA und JTS	456
17.4.3	Implizite Transaktionssteuerung	457
17.4.4	Synchronisation	458
17.5	EJB 2.0	458

17.5.1	Neuerungen	459
17.5.2	Neue Schnittstellen	459
17.5.3	Persistenz	459
17.5.4	Message Driven Beans	460
17.6	Application Server	460
17.7	Jboss	461
17.7.1	Auspacken...	461
17.7.2	Einschalten...	461
17.7.3	Fertig?	462
17.7.4	Besonderheiten	463
17.7.4.1	Auto-Deploy	463
17.7.4.2	CLASSPATH	463
17.8	Kreuzworträtsel	463
17.9	Weitere Informationsquellen	466
17.10	Zusammenfassung	467

V Testen 469

18 Fehlersuche 473

18.1	Lernziele	473
18.2	Logging	473
18.2.1	Warum Logging?	474
18.2.2	Wann Logging?	474
18.2.2.1	Anforderungen	475
18.2.3	System.out.println	475
18.2.4	Beispiel	475
18.3	log4j	476
18.3.1	Installation	477
18.3.2	Ausgabe-Level	478
18.3.3	Konfigurierbarkeit	479
18.3.4	Appender	479
18.3.5	Beispiel	479

18.3.6	Log-Konfigurierung		480
18.3.7	Weitere Aussichten		482
18.4	Techniken zur Fehlervermeidung		482
18.4.1	Das Vertrags-Modell		482
18.4.2	iContract		484
18.5	Asserts in Java		485
18.5.1	Eigene Assert-Klasse		485
18.5.2	Asserts in JDK 1.4		486
18.5.3	Assert-Optionen		488
18.5.4	Ausblenden der assert-Anweisung		489
18.6	Weitere Informationsquellen		489
18.7	Zusammenfassung		490
19	Test-Phase		493
19.1	Lernziele		493
19.2	To Test Or Not To Test...		493
19.3	JUnit		495
19.3.1	Installation		495
19.3.2	Dokumentation		496
19.3.3	Test-Gerüst		496
19.3.4	Prüfpunkte		497
19.3.5	Test-Reihenfolge		500
19.3.6	Ein einfacher Testfall		501
19.3.7	Test-Sammlung		503
19.3.8	ClassCastExceptions		504
19.3.9	JUnit Best Practices		505
19.3.10	Testen macht Spass		506
19.4	Test-Automatisierung		506
19.4.1	expect		506
19.4.2	Ein einfaches Beispiel		507
19.5	Weitere Informationsquellen		509
19.6	Zusammenfassung		510

20 Performance	511
20.1 Lernziele	511
20.2 Warum ist Java so langsam?	511
20.2.1 Objekt-Referenzen	512
20.2.2 Was ist Performance?	513
20.2.3 Taktische Überlegungen	514
20.2.4 Interfaces und Abstraktion	515
20.3 Java-Tuning	516
20.3.1 Beispiel-Programm	516
20.3.2 Environment- und Tool-Strategien	518
20.4 Native Compile	519
20.4.1 Vorbereitung	519
20.4.2 Das Ergebnis	520
20.5 Weitere Massnahmen	521
20.5.1 Prozess-Verlagerung	521
20.5.2 Caching	525
20.5.3 Objekt-Pools	527
20.5.4 Größere Granularität	528
20.5.5 Inlining	530
20.5.6 Java-Collections und Maps	530
20.5.7 Strings	533
20.6 Ein-/ Ausgabe	533
20.6.1 Serialisierung	533
20.6.2 Pufferung	535
20.7 Tuning-Kit	535
20.7.1 Profiler	535
20.7.2 hprof	536
20.7.2.1 Ein Beispiel	536
20.7.3 ProfileViewer	539
20.7.3.1 Installation	539
20.7.3.2 Probelauf	540
20.7.4 PerfAnal	541

20.7.5	HPjmeter	542
20.7.6	OptimizeIt™	543
20.7.6.1	Installation	543
20.7.6.2	Dokumentation	544
20.7.6.3	Heap-Analyse	544
20.7.6.4	VM-Info	545
20.7.6.5	CPU-Verbrauch	546
20.7.6.6	Fazit	547
20.8	Kernel-Performance	547
20.8.1	As time goes by...	547
20.8.2	System Trace (strace)	549
20.8.3	Verwendung	551
20.9	Jopt	552
20.9.1	Installation	552
20.9.2	Optimierung mit Jopt	553
20.9.3	Einsatzgebiete	554
20.10	Weitere Informationsquellen	554
20.11	Zusammenfassung	554

VI Java für Fortgeschrittene 557

21	Interface und Abstrakte Klassen	561
21.1	Lernziele	561
21.2	Was ist ein Interface?	561
21.2.1	Definition	562
21.2.2	Implementierung	562
21.2.3	Anwendung	563
21.2.4	Marker-Interface	565
21.2.5	Konstanten	566
21.3	Abstrakte Klassen	566
21.3.1	Mehrfachvererbung	568
21.4	Weitere Informationsquellen	570
21.5	Zusammenfassung	570

22 Reflexion	571
22.1 Lernziele	571
22.2 Einleitung	571
22.3 Eine einfache Dumper-Klasse	572
22.4 Basis eines Persistenz-Frameworks	574
22.5 Weitere Einsatzmöglichkeiten	574
22.6 Weitere Informationsquellen	575
22.7 Zusammenfassung	576
23 Java Native Interface (JNI)	577
23.1 Lernziele	577
23.2 Gefangen in der VM?	577
23.3 Ausbruch aus der VM?	578
23.3.1 Performance	578
23.3.2 Plattform-spezifischer Code	579
23.3.3 Anbindung vorhandener Software	579
23.4 Die Geschichte von JNI	580
23.5 Der Zugriff auf C-Funktionen	581
23.5.1 javah	581
23.5.2 Beispiel	581
23.5.2.1 1. Schritt: „Native“-Deklaration	582
23.5.2.2 2. Schritt: Java-Kompilation	582
23.5.2.3 3. Schritt: Header-Datei-Generierung	582
23.5.2.4 4. Schritt: C-Implementierung	584
23.5.2.5 5. Schritt: Generierung einer Shared Library	584
23.5.2.6 6. Schritt: Bibliothek laden	585
23.6 JNI Datentypen	586
23.6.1 Einfache Datentypen	586
23.6.2 Höhere JNI-Datentypen	587
23.6.3 Strings	588
23.6.4 Typ-Signaturen	588
23.7 JNI-Funktionen	589
23.7.1 Versions-Information	589

23.7.2	Klassen-Operationen	589
23.7.3	Exceptions	590
23.7.4	Globale und Lokale Referenzen	591
23.7.5	Objekt-Operationen	592
23.7.6	Zugriff auf Objekt-Felder	593
23.7.7	Methoden-Aufruf	594
23.7.8	Zugriff auf statische Felder	596
23.7.9	Aufruf statischer Methoden	597
23.7.10	String-Operationen	598
23.7.11	Array-Operationen	600
23.7.12	Registrierung von C/C++-Funktionen	603
23.7.13	Monitor-Operationen	604
23.7.14	JVM-Interface	605
23.7.15	Beispiele	605
23.7.15.1	Ein einfaches Beispiel	605
23.7.15.2	Ausnahme-Beispiel	606
23.8	javah	607
23.9	Bibliotheken installieren	607
23.10	Debuggen	607
23.11	Gemischte Lösung	609
23.12	JNI – Nicht Immer	611
23.13	Weitere Informationsquellen	611
23.14	Zusammenfassung	611
VII	Verbesserung der Qualität	613
24	Java-Dokumentation	617
24.1	Lernziele	617
24.2	javadoc	617
24.2.1	Warum Dokumentierung?	617
24.2.2	Javadoc-Kommentare	618
24.2.3	@tags	619

24.2.4	Die Tags im Einzelnen	620
24.2.5	Reihenfolge der Tags	623
24.2.6	HTML in Javadoc-Kommentaren	624
24.3	Aufruf von <code>javadoc</code>	625
24.3.1	Standard-Optionen	625
24.3.2	Optionen des Standard-Doclets	627
24.4	Doclets	629
24.4.1	Eigene Doclets	629
24.4.2	Ein einfaches Beispiel	629
24.4.3	Der <code>@bug</code> -Tag	630
24.4.4	Erweiterung des Standard-Doclets	631
24.4.5	Weitere Änderungen	634
24.5	Sonstige Dokumentation	635
24.5.1	$\text{T}_{\text{E}}\text{X}$ / $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$	635
24.5.2	StarOffice	637
24.6	Weitere Informationsquellen	638
24.7	Zusammenfassung	638
25	Programmier-Richtlinien	641
25.1	Lernziele	641
25.2	Richtlinien – für wen?	641
25.3	Allgemeine Überlegungen	642
25.3.1	Ziel von Richtlinien	642
25.3.2	Grundlage	643
25.3.3	Abgrenzung	643
25.4	Die 10 goldenen Java-Regeln	644
25.5	Erläuterungen und Beispiele	644
25.5.1	Abweichungen vom Standard	645
25.5.2	Aussagekräftige Namen	645
25.5.2.1	Allgemein	645
25.5.2.2	Hilfs-Variablen	647
25.5.2.3	Setter- / Getter-Methoden	648
25.5.3	Redundanzen	648

25.5.3.1	Keine redundanten Benennungen	. . .	648
25.5.3.2	Kein redundanter Code	. . .	649
25.5.4	Kommentierung	. . .	649
25.5.4.1	Was sind gute Kommentare?	. . .	649
25.5.4.2	Javadoc-Kommentare	. . .	650
25.5.4.3	Implementierungs-Kommentare	. . .	653
25.5.5	Schreibweisen	. . .	654
25.5.5.1	Klassennamen in Großbuchstaben	. . .	654
25.5.5.2	Variablen in Kleinbuchstaben	. . .	654
25.5.5.3	Methoden in Kleinbuchstaben	. . .	655
25.5.6	Konstanten komplett groß	. . .	655
25.5.7	Unterstriche, optische Trennung	. . .	655
25.5.8	Packages	. . .	656
25.5.8.1	Namenskonvention	. . .	656
25.5.8.2	root-Package	. . .	656
25.5.8.3	Spezielle Packages	. . .	656
25.5.8.4	import	. . .	656
25.5.9	Formatierung	. . .	656
25.5.9.1	Einrückung	. . .	657
25.5.9.2	Zeilenumbruch	. . .	657
25.5.9.3	Leerzeichen	. . .	658
25.5.10	this	. . .	659
25.6	Weitere Empfehlungen	. . .	660
25.6.1	Metriken und qualitative Kennzahlen	. . .	660
25.6.2	Variablen- und Methoden-Deklaration	. . .	660
25.6.3	Deutsch oder Englisch	. . .	660
25.6.4	Umlaute	. . .	662
25.6.5	Länderspezifische Strings	. . .	663
25.6.6	Erfolgreicher Code	. . .	664
25.7	Einhaltung	. . .	664
25.7.1	Die Anti-Regeln	. . .	664
25.8	Design-Fragen	. . .	668

Inhaltsverzeichnis

25.8.1	Wiederverwendung		668
25.8.2	Interface oder Klasse?		669
25.8.3	Die akademische Antwort		669
25.8.3.1	Die Impl-Klasse		669
25.8.4	Die Antwort aus der Praxis		671
25.8.5	Konstanten via Interface		671
25.9	EJB-Richtlinien		672
25.9.1	Namenskonvention		672
25.9.1.1	Bean-Klassen		672
25.9.1.2	Finder-Methoden		672
25.9.2	Package-Struktur		672
25.9.3	Granularität		672
25.9.4	Kommunikation		673
25.9.5	Deployment		673
25.10	Weitere Informationsquellen		673
25.11	Zusammenfassung		674
26	Q-Tools		675
26.1	Lernziele		675
26.2	Code-Reading		676
26.2.1	Umfang		676
26.2.2	Was bringt Code-Reading?		677
26.2.3	Was kann man damit vermeiden?		677
26.2.4	Was gibt es sonst noch dazu zu sagen?		677
26.2.5	Die XP-Lösung		678
26.2.6	Die Tool-Lösung		678
26.3	QStudio™Java		678
26.3.1	Installation		679
26.3.2	Projekt aufsetzen		679
26.3.3	JBUILDER-Integration		681
26.3.4	Fazit		681
26.4	Jlint		682
26.4.1	Installation		682

26.4.2	Testlauf		682
26.4.3	IDE-Einbindung		683
26.4.4	Kategorien		684
26.4.5	Erfahrungen		685
26.5	Code-Formatierung		685
26.5.1	Wozu?		685
26.5.2	JavaBeautifier		686
26.5.3	JxBeauty		687
26.5.4	Jindent		688
26.5.4.1	Aufruf		689
26.5.4.2	Anpassungen		689
26.5.4.3	Javadoc-Generierung		689
26.5.4.4	Geschützte Bereiche		690
26.5.4.5	Fazit		690
26.6	Zusammenfassung		692

VIII SW-Verteilung 693

27 Open Source Software (OSS) 697

27.1	Lernziele		697
27.2	Die Anfänge		697
27.2.1	Hacker und Cracker		698
27.2.2	Kommerzialisierung		698
27.3	Die Kathedrale und der Basar		699
27.4	Wann lohnt sich OSS?		702
27.4.1	Vorteile		704
27.4.2	Voraussetzungen		706
27.4.3	Closed Source		707
27.4.4	Marktmodelle für OSS		707
27.5	OSS-Lizenzen		709
27.5.1	OpenSource Definition		709
27.5.2	GNU General Public Licence (GPL)		710

27.5.3	Lesser GPL (LGPL)		711
27.5.4	BSD-Lizenz		711
27.5.5	Artistic Licence		711
27.5.6	NPL / MPL		712
27.5.7	Weitere Lizenz-Modelle		712
27.5.8	Sun Community Source Licence (SCSL)		712
27.6	Eigene OSS-Projekte		713
27.6.1	Projekt aufsetzen		713
27.6.2	Benutzer-Registrierung		714
27.6.3	Projekt-Registrierung		714
27.6.4	Administration		714
27.6.5	Sourcen anlegen		715
27.7	Projekt-Pflege		716
27.7.1	Die Web-Seite		716
27.7.2	Weitere Aktivitäten		717
27.8	Weitere Informationsquellen		717
27.9	Zusammenfassung		718
28	Installationen		719
28.1	Lernziele		719
28.2	IBM Java Installations-Toolkit		720
28.2.1	Erstellen einer Installations-Klasse		720
28.2.2	Ein erster Test		721
28.2.3	Anpassungen		721
28.2.4	Deinstallation		722
28.2.5	Weitere Merkmale		722
28.2.6	Einschränkungen		722
28.3	InstallShield		723
28.3.1	Installation		723
28.3.2	Dokumentation		724
28.3.3	Projekt-Erstellung		725
28.3.3.1	Begriffe		725
28.3.3.2	Ein einfaches Projekt		725

28.3.4	Test-Installation	728
28.3.5	Deinstallation	728
28.3.6	Installations-Optionen	729
28.3.7	Projekt-Datei	729
28.3.8	Unbegrenzte Möglichkeiten	729
28.3.9	Was nicht im Handbuch steht...	730
28.3.9.1	Auf der Suche nach der VM	730
28.3.9.2	Installations-Verzeichnis	730
28.3.9.3	Kommandozeilen-Build	731
28.3.10	Gesamteindruck	732
28.4	RPM	732
28.4.1	Spec-Datei	732
28.4.1.1	Der Header	733
28.4.1.2	Die Bauanleitung (Prep-Abschnitt)	734
28.4.1.3	Build-Abschnitt	735
28.4.1.4	Install-Abschnitt	735
28.4.1.5	Aufräumarbeiten	735
28.4.1.6	Pre-/Post-(De-)Installations-Skripte	736
28.4.1.7	Files-Abschnitt	736
28.4.1.8	Changelog	737
28.4.2	Einpacken	737
28.4.2.1	Vorbereitung	737
28.4.2.2	Paket-Erstellung	738
28.4.3	Probe-Installation	739
28.5	Die WebStart-Technologie	740
28.5.1	Funktionsweise	740
28.5.2	Was kann Java WebStart	741
28.5.3	Installation	741
28.5.4	Launch-Time	741
28.5.5	Eigene Programme	743
28.5.5.1	Die JNLP-Datei	744
28.5.5.2	Sonstige Dateien	745

28.5.5.3	Web-Server konfigurieren	.	.	.	.	.	.	.	745
28.5.5.4	Web-Browser konfigurieren	.	.	.	.	.	.	.	745
28.5.5.5	Ein bisschen JavaScript	.	.	.	.	.	.	.	745
28.5.6	Sicherheit	.	.	.	.	.	.	.	747
28.5.6.1	Test-Zertifikat	.	.	.	.	.	.	.	747
28.5.6.2	Signierung	.	.	.	.	.	.	.	748
28.5.7	Der Application-Manager	.	.	.	.	.	.	.	748
28.5.8	Die JNLP-Datei	.	.	.	.	.	.	.	749
28.5.8.1	JNLP-Elemente	.	.	.	.	.	.	.	749
28.5.8.2	Informations-Elemente	.	.	.	.	.	.	.	749
28.5.8.3	Security-Elemente	.	.	.	.	.	.	.	750
28.5.8.4	Resource-Elemente	.	.	.	.	.	.	.	750
28.5.8.5	Property-Element	.	.	.	.	.	.	.	751
28.5.8.6	application-desc-Element	.	.	.	.	.	.	.	751
28.5.8.7	applet-desc-Element	.	.	.	.	.	.	.	751
28.5.9	Das JNLP-API	.	.	.	.	.	.	.	751
28.5.10	Fazit	.	.	.	.	.	.	.	752
28.6	Weitere Informationsquellen	.	.	.	.	.	.	.	753
28.7	Zusammenfassung	.	.	.	.	.	.	.	753
29	Ausgeliefert!								755
29.1	Lernziele	.	.	.	.	.	.	.	755
29.2	Blind Date	.	.	.	.	.	.	.	755
29.2.1	Test-Versionen	.	.	.	.	.	.	.	757
29.2.2	Service-Packs	.	.	.	.	.	.	.	757
29.2.3	Bug-Fixe	.	.	.	.	.	.	.	758
29.3	Trouble-Ticket-Systeme	.	.	.	.	.	.	.	758
29.3.1	The Day After...	.	.	.	.	.	.	.	758
29.3.2	Wozu Trouble-Ticket-Systeme	.	.	.	.	.	.	.	758
29.3.3	Anforderungen	.	.	.	.	.	.	.	759
29.3.4	GNATS	.	.	.	.	.	.	.	760
29.3.5	Request Tracker (RT)	.	.	.	.	.	.	.	761
29.3.6	OpenTrack (OT)	.	.	.	.	.	.	.	761

29.3.7	WebTTS		762
29.3.8	Debian Bug TS		762
29.3.9	Req / ReqNG		763
29.3.10	Bugzilla		763
29.3.11	JitterBug		764
29.3.12	Double Choco Latte (DCL)		765
29.4	DCL@Work		765
29.4.1	Installation		766
29.4.2	Accounts		771
29.4.3	Der erste Fehlerreport		771
29.4.4	Ticket abschließen		773
29.4.5	Planung		774
29.4.6	Eindrücke		774
29.5	Weitere Informationsquellen		774
29.6	Zusammenfassung		775
IX	Ende		777
30	Schlussbemerkung		779
30.1	Wie wird es weitergehen...		779
30.1.1	... mit Java?		779
30.1.2	... mit Linux?		780
30.1.3	... mit Ihrem Chef?		781
30.2	Linux lebt!		781
X	Anhang		783
A	Lösung des Kreuzwort-Rätsels		785
B	UML-Diagramme		789
C	Listings		793
C.1	Applets		793

C.1.1	CounterApplet	793
C.1.2	CounterApplet.html	795
C.1.3	counter.policy	795
C.2	RMI-Beispiel	795
C.2.1	flirt.server.Darling	795
C.2.2	flirt.server.Juliet	796
C.2.3	flirt.client.Romeo	797
C.3	Enterprise Java Beans	798
C.3.1	HelloBean-Beispiel	798
C.3.1.1	Remote-Interface (Hello.java)	798
C.3.1.2	Home-Interface (HelloHome.java)	798
C.3.1.3	HelloBean.java	799
C.3.1.4	ejb-jar.xml	799
C.3.1.5	weblogic-ejb-jar.xml	800
C.3.2	AccountBean-Beispiel	801
C.3.2.1	Remote-Interface (Account.java)	801
C.3.2.2	Home-Interface (AccountHome.java)	801
C.3.2.3	AccountBean.java	801
C.3.2.4	Primärschlüssel (AccountPK.java)	802
C.3.2.5	ejb-jar.xml	803
C.3.2.6	weblogic-ejb-jar.xml	804
C.3.2.7	weblogic-cmp-rdbms-jar.xml	804
C.4	Zauberspruch-Generator	805
C.4.1	wizard.Spell	805
C.4.2	wizard.server.SpellProvider	807
C.4.3	wizard.server.MagicSpellProvider	807
C.4.4	wizard.client.SpellPot	808
C.5	Crypt-Beispiel	809
C.5.1	Java-Sourcen	809
C.5.1.1	enigma.Crypt	809
C.5.1.2	enigma.RandomByte	815
C.5.2	C-Sourcen	816

C.5.2.1	enigma_Crypt.h	.	.	.	.	.	816
C.5.2.2	enigma_Crypt.c	.	.	.	.	.	816
D	Die beiliegende CD-ROM						819
D.1	Inhalt	.	.	.	.	.	819
D.1.1	Kommerzielle Tools	.	.	.	.	.	819
D.1.2	Open Source Software	.	.	.	.	.	820
D.1.3	Utilities	.	.	.	.	.	821
D.1.4	Beispiel-Projekte	.	.	.	.	.	821
D.1.5	Java-Bibliotheken	.	.	.	.	.	822
D.1.6	Java-Dokumentation	.	.	.	.	.	823
D.2	Spezielle Dateien	.	.	.	.	.	823
E	Toolübersicht						825
E.1	Development	.	.	.	.	.	825
E.1.1	Development/Languages/Java	.	.	.	.	.	825
E.1.2	Development/Libraries/Java	.	.	.	.	.	826
E.1.3	Development/Tools/Building	.	.	.	.	.	826
E.1.4	Development/Tools/Debuggers	.	.	.	.	.	826
E.1.5	Development/Tools/IDE	.	.	.	.	.	826
E.1.6	Development/Tools/Version Control	.	.	.	.	.	827
E.2	Productivity	.	.	.	.	.	827
E.2.1	Productivity/Databases/Clients	.	.	.	.	.	827
E.2.2	Productivity/Databases/Servers	.	.	.	.	.	827
E.2.3	Productivity/Databases/Tools	.	.	.	.	.	828
E.2.4	Productivity/Editors	.	.	.	.	.	828
E.2.5	Productivity/Networking/SSH	.	.	.	.	.	828
E.2.6	Productivity/Networking/Frontends	.	.	.	.	.	828
E.2.7	Productivity/Networking/Web/Servers	.	.	.	.	.	829
E.2.8	Productivity/Office/Suite	.	.	.	.	.	829
E.2.9	Productivity/Publishing/XML	.	.	.	.	.	829
E.3	System	.	.	.	.	.	829
E.3.1	System/Packages	.	.	.	.	.	829

F	Glossar	831
F.1	Allgemein	831
F.2	Abkürzungen	836

Teil I

Einleitung

Im ersten Teil wollen wir die Grundlagen schaffen, um mit Java und diesem Buch erfolgreich unter Linux entwickeln zu können:

❑ **Glaubenskriege:**

Wollten Sie immer schon einmal unter Linux entwickeln, aber Ihr Chef lässt sie nicht? Hier finden Sie einige gute Gründe, die vielleicht auch Ihren Chef überzeugen dürften.

❑ **Java-Installation:**

Bevor es mit der Entwicklung unter Java losgehen kann, muss es irgendwie auf den Rechner kommen. Wie – das wird in diesem Abschnitt beschrieben.

❑ **XML – eine kurze Einführung:**

Was hat XML mit Java zu tun? Eigentlich nichts. Oder doch? In diesem Abschnitt erhalten Sie eine Kurzeinführung in XML, da XML an einigen Stellen im Buch auftauchen wird.

Nach diesem Einstieg werden wir im nächsten Teil dann den Umgang mit Java und dem JDK vertiefen.

Kapitel 2

Java-Installation

In the computer industry, there are three kind of lies:
lies, damn lies, and benchmarks.
(aus: *The Jargon File*)

2.1 Lernziele

Auch wenn Java von Sun Microsystems entwickelt und vorangetrieben wurde, hat man inzwischen die Auswahl zwischen einer Reihe von Alternativen. Bevor Sie sich gleich in die Installation stürzen, verschaffen wir uns erst einen Überblick über

- ❑ wichtige Begriffe und Abkürzungen,
- ❑ die verschiedenen (Java Virtual Machines) JVMs und
- ❑ die verschiedenen Bibliotheken und Erweiterungen.

2.2 Begriffsbestimmung

Wenn man mit Java arbeitet, wird man oft mit folgenden Begriffen konfrontiert:

JDK das *Java Development Kit* enthält alles, was man zur Entwicklung braucht: die Java Virtual Machine (JVM), den Java-Compiler (`javac`) und den Java-Debugger (`jdb`).

JRE das *Java Runtime Environment* enthält nur die Komponenten, die zum Starten eines übersetzten Java-Programms notwendig sind.

JVM die *Java Virtual Machine*, die den Byte-Code eines übersetzten Java-Programms ausführt.

VM manchmal wird JVM auch nur mit VM (*Virtual Machine*) abgekürzt.

JNI Das *Java Native Interface* ist die Schnittstelle, über die man C / C++-Funktionen aufrufen kann.

CLASSPATH – ist vergleichbar mit dem Suchpfad der Shell. Beim Starten einer Java-Anwendung wird nach dem Aufruf einer Klasse diese im *CLASSPATH* gesucht. Wird sie nicht gefunden, kommt es zu einer „ClassNotFoundException“. Gerade als Neuling oder am Anfang eines Projekts hat man häufig damit zu kämpfen, bis man den richtigen CLASSPATH beisammen hat – also nicht gleich verzweifeln, Sie sind nicht allein damit.

Die einzelnen Pfadangaben im CLASSPATH werden unter Linux durch einen Doppelpunkt (:) voneinander getrennt.

Beispiel:

```
export CLASSPATH=./lib/java/log4j.jar:/lib/java/junit3.5/junit.jar
```

Applet – damit werden kleine Java-Anwendungen bezeichnet, die nur innerhalb eines Web-Browsers oder eines Applet-Viewers ablauffähig sind.

Standard Extension / Optionale Pakete Dies sind von Sun spezifizierte Erweiterungen, die nicht Bestandteil des JDKs sind und separat bezogen werden müssen. Ein Beispiel dafür ist die Java3D-Spezifikation für die Unterstützung von 3D-Objekten oder JavaMail zum Versenden von E-Mails (s. Kap. 2.6.3).

Nicht genug der Begriffsvielfalt – es wurden von Sun noch einige weitere neue Begriffe ins Spiel gebracht, z. B.:

Java 2 Plattform – Dieser Begriff wurde mit JDK 1.2 eingeführt und spezifiziert die Umgebung für Applets und Java-Anwendungen. Etwas verwirrend an dieser Namensgebung war die gewollte oder ungewollte Kopplung von Java 2 mit JDK 1.2, wobei das JDK nur einen Teil in der Java 2 Plattform-Definition ausmacht. Die offizielle Bezeichnung des zur Drucklegung des Buches aktuellen Releases ist „Java 2 Plattform Version 1.4“.

Java Foundation Classes (JFC) – Dies ist nur ein Marketingname, unter dem einige alte Klassenbibliotheken wie AWT, Swing oder Java2D zusammengefasst wurden.

Java 2 Runtime Environment (J2RE) – ein neuer Name der JRE, der mit der Java2-Plattform eingeführt wurde.

Java 2 Software Development Kit (J2SDK) – Damit wird nun im wesentlichen das JDK beschrieben. Wenn im Folgenden von JDK 1.4 die Rede ist, ist damit genau genommen das „Java 2 Software Development Kit Version 1.4“ gemeint.

Java 2 Standard Edition (J2SE) – Dies ist der Standardumfang der Java2-Plattform für den Einsatz von Java auf der Client-Seite (im Desktop- und Browser-Bereich).

Java 2 Enterprise Edition (J2EE) = J2SE + einige Erweiterungen für den Server-Bereich (wie z. B. Servlets¹, JSP², EJB³)

Java 2 Micro Edition (J2ME) = J2SE minus „Ballast“, d. h., dies ist eine abgespeckte J2SE, die hauptsächlich für den Einsatz im Embedded Bereich (Handies, Waschmaschinen, Kleinstcomputer, ...) vorgesehen ist.

2.3 Anforderungen an die...

2.3.1 ... Hardware

Es ist schwierig, eine allgemeine Abschätzung über die benötigte Hardware abzugeben, da das von verschiedenen Faktoren, wie eingesetzte Java-Plattform (Micro-, Standard- oder Enterprise-Edition) und benötigte Oberfläche (Kommandozeile, GUI) abhängt. Als ich mit Java und den Recherchen zu diesem Buch anfang, besaß ich einen Pentium I mit 133 MHz und 96 MB Hauptspeicher. Vieles im Java-Bereich war damit quälend langsam. Erst als ich zu einer (gebrauchten) Workstation mit 2 · 450 MHz und 512 MB Hauptspeicher aufgestiegen bin, kam richtig Freude auf.

Wenn Sie nur mal in Java reinschnuppern wollen, würde ich es unter einem

- ❑ Pentium II mit 233 MHz
- ❑ 128 MB Hauptspeicher

gar nicht erst versuchen. Für eine ernsthafte Java-Entwicklung sollte es dann schon das Doppelte sein, d. h.

- ❑ Pentium III mit 400 MHz
- ❑ 256 MB Hauptspeicher

Vor allem am Hauptspeicher sollten Sie nicht sparen – je mehr, desto besser. Es gibt einige Java-basierte Entwicklungs-Tools, die es Ihnen danken werden.

¹s. Kap. 16.4

²s. Kap. 16.7

³s. Kap. 17

2.3.2 ... Software

Auch wenn in der Vergangenheit meist RedHat als Voraussetzung für den Java-Einsatz unter Linux angegeben wurde,⁴ heißt das nicht, dass Java nicht auf den anderen Distributionen läuft. Persönlich setze ich z. Zt. SuSE ein und habe damit keine Schwierigkeiten, was die verschiedenen Java-Umgebungen betrifft. Auch Java unter Calderas OpenLinux-Version, das ich leihweise für eine Präsentation verwendet habe, lief problemlos.⁵

Meist ist Java bereits im Umfang der verschiedenen Distributionen mit enthalten,⁶ sodass Sie hier relativ frei in der Auswahl Ihrer Lieblings-Distribution sind.

2.4 Welche JVM?

Verglichen mit der Windows-Welt hinkte anfangs die Entwicklung der einzelnen Java Development Kits (JDK) etwas hinterher, trotz der offiziellen Unterstützung durch Sun. So ist es vor allem den Entwicklern um Blackdown.org⁷ zu verdanken, dass nach langer Zeit des Wartens JDK 1.2 endlich in seiner endgültigen Fassung verfügbar war. Dies hat sich inzwischen geändert, da Linux von IBM und auch von Sun zur strategischen Plattform erhoben wurde.

Aber auch abseits der Mutter Sun gibt es verschiedene Implementierungen der JVM und Java-Compiler. Java ist dankenswerterweise im Wesentlichen eine Spezifikation, und jeder kann und darf selbst seine JVM schreiben, wenn er mit der VM von Sun nicht zufrieden ist.

2.4.1 JDK 1.2

Inzwischen ist die 1.2er-Version bei den meisten Distributionen mit enthalten. Falls dies nicht der Fall ist, ist der erste Anwahlpunkt für den Linux-Indianer neben

- ❑ <http://www.javasoft.com> auch
- ❑ <http://www.blackdown.org>.

Die offizielle Version von Sun entpuppt sich bei näherem Hinsehen als die Blackdown-Version. Allerdings hat Sun einige kleinere Erweiterungen vorgenommen;

⁴so gibt Sun für JDK 1.4 als offiziell unterstützte Plattform RedHat 7.1 und 6.2 an (<http://java.sun.com/j2se/1.4/jre/install-linux.html>)

⁵Hierbei zählen sich die Bemühungen der verschiedenen Distributoren aus, mit Hilfe der LSB (Linux Standard Base, s. <http://www.linuxbase.org/>) eine gemeinsame Basis für die unterschiedlichen Distributionen zu schaffen.

⁶So wird bei SuSE 8.0 sowohl das JDK 1.3.1 von Sun als auch das JDK 1.3 von IBM mitgeliefert.

⁷<http://www.blackdown.org>

z.B. gibt es beim Drucken aus Java-Programmen heraus mit der Sun-VM meist weniger Probleme als bei der Blackdown-Variante.

Wer die aktuelle Version von Blackdown.org herunterladen will, braucht etwas Geduld (oder eine schnelle Internetverbindung): es sind ca. 20 MB, die eine 64KBit-Leitung für eine 3/4 Stunde auslasten können.

Die Installation selber ist denkbar einfach: als `root` nach `/usr/local` gehen und die heruntergeladene Tar-Datei mit „`tar xvfI ...`“ auspacken.⁸

2.4.2 IBM-JDK 1.1.8

Das JDK 1.1.8 von IBM war sehr lang die schnellste virtuelle Maschine auf dem Markt und zudem für Linux frei erhältlich. Es ist daher vor allem für betagtere Rechner und Programme, die mit Performance-Problemen zu kämpfen haben, sicherlich die erste Wahl, zumal es auch durch Stabilität glänzt. Einziger Wermutstropfen ist die Beschränkung auf JDK 1.1, was nicht mehr ganz zeitgemäß ist.

2.4.3 IBM-JDK 1.3

Lange Zeit war IBM bei JDK 1.1 stehengeblieben (nicht nur bei der VM), während die Entwicklergemeinde schon lange mit JDK 1.2 arbeitete und auf eine Antwort von IBM wartete. Offensichtlich war man sich dort dieses Problems bewusst, denn IBM tat zum Erstaunen vieler einen großen Schritt nach vorn, indem es die 1.2er-Version kurzerhand übersprang und die 1.3er-Version gleich für neun Plattformen (darunter natürlich auch Linux) ankündigte.

Inzwischen ist das JDK 1.3 von IBM verfügbar, und es hat nichts von seiner Stabilität und Schnelligkeit eingebüßt. Wenn man von der Graphik-Leistung einmal absieht (die im Server-Bereich sowieso unerheblich ist), hat die VM sogar in allen Bereichen um ca. 50 % zugelegt. Damit dürfte diese VM sicherlich bei vielen Anwendungen, vor allem im Server-Bereich, zur ersten Wahl gehören.

Im Desktop-Bereich muss man allerdings Kompromisse eingehen. So gibt es Probleme beim Doppel-Klick mit der Maus – er funktioniert meistens nicht. Versuchen Sie einmal, im Dialogfenster zur Dateiauswahl einen Ordner ohne Doppelklick zu öffnen. Zur Ehrenrettung von IBM muss gesagt werden, dass dieses Fehlverhalten auch mit dem Sun-JDK unter Windows NT zu beobachten war, wenn auch äußerst selten. Dies legt die Vermutung nahe, dass es sich dabei um eine Unsauberkeit der Swing-Bibliothek handeln könnte.

⁸Manche tar-Versionen, vor allem auf anderen Unixen, kennen die Option „-I“ nicht. In diesem Fall kann die Datei mit „`bunzip2 -c ... | tar xvf -`“ ausgepackt werden.

2.4.4 Sun-JDK 1.3

Inzwischen wurde Linux zur offiziell unterstützten Java-Plattform erhoben. Damit stand zeitgleich mit der Windows- und Solaris-Version auch eine Linux-Version zur Verfügung. Im Falle von JDK 1.3 kann diese direkt von Sun als ausführbares RPM- oder Tar-Skript (Größe ca. 25 MB) heruntergeladen werden. Startet man dieses Skript, so erhält man (nach Bestätigung der Lizenz-Bedingungen) eine RPM- bzw. Tar-Datei, die man dann per `rpm` bzw. `tar` installieren kann.

Installiert man es als RPM-Paket, geschieht dies unter `/usr/java`. Eventuell muss man noch `glibc` aktualisieren (oder die Option „`--nodeps`“ verwenden), damit die Installation erfolgreich verläuft.

Die Sun-VM ist zwar nicht so schnell wie die IBM-VM, dafür läuft sie im Desktop-Bereich stabiler. Ärgerlich ist, dass beim Starten einer Swing-Anwendung über fehlende Fonts gemeckert wird, aber ansonsten funktioniert alles wie erwartet, auch der Doppelklick. Nicht umsonst ist die Sun-VM die „offizielle“ VM, auf die von den meisten Herstellern bei Problemen mit Java-Programmen gern hingewiesen wird.

2.4.5 Sun-JDK 1.4

Offiziell heisst diese Version J2SE v1.4, aber viele Entwicklern sagen einfach „JDK 1.4“ dazu. Die Installation verläuft ähnlich wie in der Vorgänger-Version, lediglich die Größe ist inzwischen bei 40 MB angelangt. Damit das Skript ausgeführt werden kann, muss man die Ausführungsberechtigung (`chmod +x`) setzen oder den Interpreter mit der heruntergeladenen Datei füttern:

```
linux: # cd /usr/java ↵
linux:/usr/java # sh j2sdk-1_4_0-linux-i386.bin ↵
Sun Microsystems, Inc.
Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE
TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE
MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO
...
Do you agree to the above license terms? [yes or no]
yes ↵
Unpacking...
Checksumming...
0
0
Extracting...
UnZipSFX 5.40 of 28 November 1998, by Info-ZIP (Zip-Bugs@lists.wku.edu).
creating: j2sdk1.4.0/
```

```

creating: j2sdk1.4.0/jre/
creating: j2sdk1.4.0/jre/bin/
inflating: j2sdk1.4.0/jre/bin/java
inflating: j2sdk1.4.0/jre/bin/keytool
inflating: j2sdk1.4.0/jre/bin/policytool
...
Done.
linux:/usr/java #

```

Hier wurde das ausführbar Tar-Shell-Skript heruntergeladen und gestartet. Nach der Bestätigung der Lizenz-Vereinbarung, die man über die Leertaste durchblättern kann, wird das Archiv im aktuellen Verzeichnis ausgepackt. Nach der Installation sollte man die `JAVA_HOME`-Umgebungsvariable setzen und den Java-Interpreter in den Suchpfad mit aufnehmen:

```

user@linux: > export JAVA_HOME=/usr/java/j2sdk1.4.0
user@linux: > export PATH=$JAVA_HOME/bin:$PATH
user@linux: > export MANPATH=$JAVA_HOME/man:$MANPATH
user@linux: > java -version
java version "1.4.0"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0-b92)
Java HotSpot(TM) Client VM (build 1.4.0-b92, mixed mode)

```

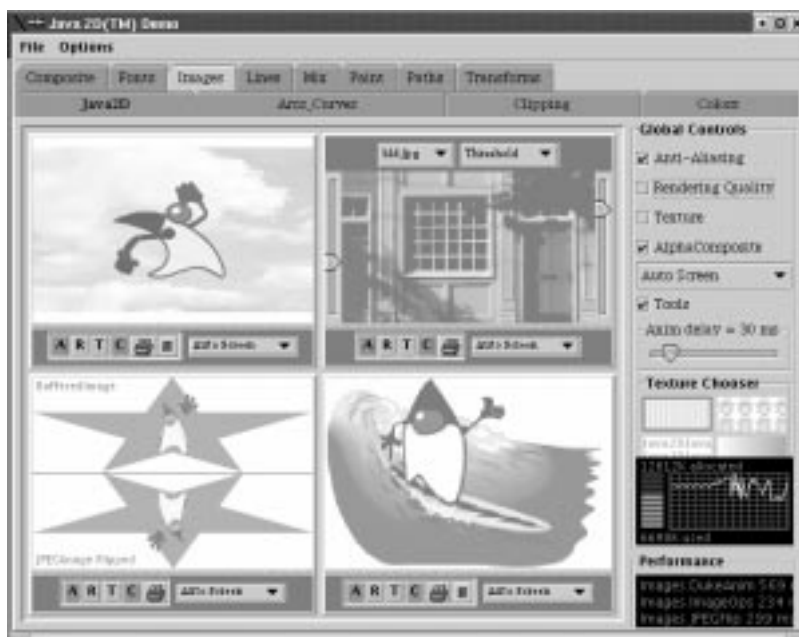


Abbildung 2.1: Java2D-Demo

Zum Ausprobieren der neuen Version empfiehlt es sich, den Java-Interpreter mit der Option „-version“ aufzurufen (s. o.). Ist die Umgebung richtig aufgesetzt, sollte er sich mit der Version „1.4.0“ melden.

Für einen weiteren Schnupperkurs in die frisch installierte Java-Maschine lohnt sich ein Blick in das `demo`-Verzeichnis. Hier findet sich unter „jfc“ (Java Foundation Classes) einige interessante Beispiele mit jeweils einer `README.txt`-Datei (s. Abb. 2.1):

```
user@linux: > cd $JAVA_HOME/demo/jfc/Java2D ↵
user@linux: /usr/java/j2sdk1.4.0/demo/jfc/Java2D > java -jar Java2Demo.jar ↵
```

2.4.6 JIT-Compiler

Die Plattformunabhängigkeit von Java-Programmen wird dadurch erreicht, dass Java-Sourcen in den Java-Bytecode übersetzt werden, der dann interpretiert wird. Dadurch haben Java-Programme oft mit Performance-Problemen zu kämpfen. Just-In-Time-(JIT-)Compiler versuchen nun, den Byte-Code, der vom Interpreter in Maschinencode übersetzt und ausgeführt wird, zu puffern, damit er beim nächsten Durchlauf nicht nochmals übersetzt werden muss. Dies bringt vor allem bei Schleifen einen enormen Geschwindigkeitsgewinn.

Ab JDK 1.2 wird von Sun ein JIT-Compiler mitgeliefert (`sunwjit`). Daneben gibt es auch freie Implementierungen wie `tya`⁹, `shuJIT`¹⁰, Metrowerks JIT für den PowerPC¹¹ oder CACAO für den Alpha-Prozessor¹². Ferner bietet OpenJIT¹³ verschiedene Eingriffsmöglichkeiten in den Übersetzungsprozess.

2.4.7 HotSpot-Technologie

Einen Schritt weiter geht die HotSpot-Technologie¹⁴ von Sun. Sie versucht noch zusätzlich, im Hintergrund die zeitkritischen Teile, die „Hot Spots“, zu optimieren, um einen weiteren Geschwindigkeitsschub zu erreichen. Dazu werden während der Laufzeit Informationen gesammelt, um die lohnenden Code-Stellen für die Optimierung zu finden. Vor allem bei Langläufern, wie sie typischerweise auf dem Server zu finden sind, kann diese Technik ihre Stärken ausspielen, sodass sich hier der zusätzliche Mehraufwand sehr schnell bezahlt macht.

In der Praxis funktioniert die HotSpot-Technologie ganz gut, solange man nicht auf die Idee kommt, Programme zu debuggen. Hier scheint der HotSpot-Com-

⁹[ftp://gonzalez.cyberus.ca/pub/{\Linux}/java](http://gonzalez.cyberus.ca/pub/{\Linux}/java)

¹⁰<http://www.shudo.net/jit/>

¹¹<http://business.tyler.wm.edu/mklinux/>

¹²<http://www.complang.tuwien.ac.at/java/cacao/>

¹³<http://www.openjit.org/>

¹⁴<http://java.sun.com/products/hotspot>

piler seine Optimierungen ständig zu verwerfen, sodass Programme im Debugger sehr zäh ablaufen. In diesem Fall die HotSpot-Technologie über die Option „-classic“ beim Aufruf der JVM deaktivieren.

Neben der HotSpot-Technologie hat Sun noch weitere Optimierungen vorgenommen, um das leidige Performance-Problem zu mildern:

- ❑ Objekt-Referenzen werden nicht mehr über einen „Handle“¹⁵ verwaltet, sondern direkt als Zeiger. Dies spart eine zusätzliche Indirektion beim Zugriff.
- ❑ Die Thread-Synchronisation wurde optimiert.
- ❑ Ein gemeinsamer Stack zwischen Java und Maschinen-Code reduziert die Kosten beim Umschalten zwischen den verschiedenen Sprach-Kontexten.
- ❑ Aggressiveres Inlining und geringer Overhead beim Methoden-Aufruf reduzieren die Kosten für den Aufruf einer Methode. Dies ist oft ein Flaschenhals bei OO-Programmen, da sie traditionell von vielen kleinen Methoden leben.
- ❑ Durch neues Heap-Management mit schnellerer Speicher-Zuteilung, besserer Garbage Collection (GC) und Verminderung von „unpassenden“ GC-Aktivitäten ist der Garbage Collector insgesamt schneller und besser geworden.

2.4.8 Benchmarking

Um ein Gefühl für die unterschiedlichen JVMs zu bekommen, habe ich nach einigen Benchmarks Ausschau gehalten. Es gibt verschiedene Benchmarks, die ihre Schwerpunkte auf die unterschiedlichen Einsatzgebiete von Java-Programmen legen (Otto-Normal-Programm, Server-Programme, ...). Da mein Testlabor anfangs nur einen alten, betagten Pentium I mit 133 MHz aufweisen konnte, war ich bei der Suche etwas eingeschränkt und bin dabei auf den

2.4.8.1 CaffeineMark

gestoßen (s. Abb. 2.2). Obwohl er schon etwas älter ist, hat mir dieser Test deswegen so gut gefallen, weil als Referenzsystem ein

1. Pentium I (133 MHz) mit
2. Windows 95 und
3. Suns JDK 1.1

¹⁵Die Objekte werden über eine Tabelle verwaltet. Der „Handle“ ist eine Zahl (nämlich der Tabellen-Index), über die das Objekt erreichbar ist.

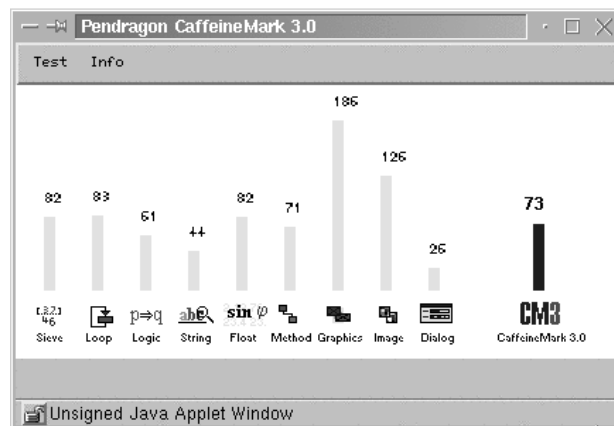


Abbildung 2.2: CaffeineMarks im Netscape 4.7

dient. Als Referenzwert liefert es 100 Caffeine-Marks. Damit lassen sich die unterschiedlichen JDKs nicht nur untereinander vergleichen, sondern auch mit der Windows-Welt.

Beim Starten des CaffeineMark-Tests erscheint ein Fenster, in dem die Resultate graphisch angezeigt werden. Jeder der 9 Tests dauert etwa 5 Sekunden (insgesamt also ca. 45 Sekunden).

Sieve das klassische Sieb des Eratosthenes zum Auffinden von Primzahlen. Dabei ist in erster Linie die Integer-Leistung gefragt.

Loop wie gut optimiert die VM Programm-Schleifen?

Logic testet die Geschwindigkeit, wie schnell die VM zur Entscheidungsfindung kommt.

Method wie gut werden Methoden-Aufrufe abgehandelt? Dies wird durch rekursive Methoden-Aufrufe getestet.

Float simuliert eine 3D-Rotation um einen Punkt. Hierbei kommt es hauptsächlich auf die Gleitkomma-Leistung an.

Graphics zeichnet zufällig Rechtecke und Linien in ein Ausgabefenster und testet damit die Graphik-Leistung.

Image zeichnet mehrmals eine Sequenz von drei Graphiken.

Dialog schreibt eine Reihe von Werten in Label-Objekte und Edit-Boxen.

CaffeineMark dies ist das geometrische Mittel aus den Einzel-Tests, d. h. die 9. Wurzel aus dem Produkt der Einzelergebnisse.

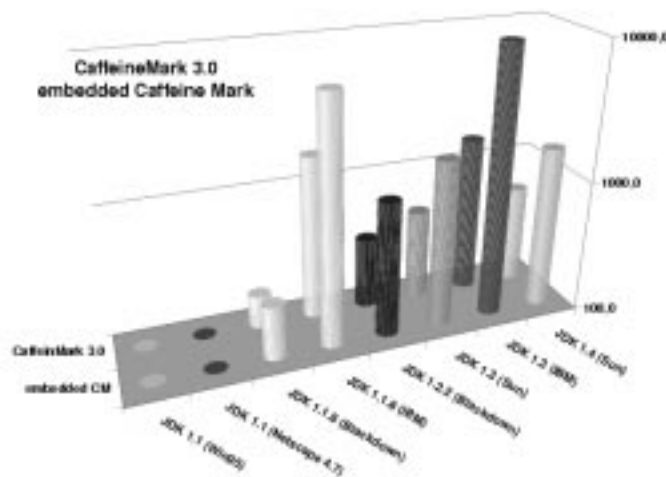


Abbildung 2.3: Der Caffeine-Benchmark

Der CaffeineMark3-Benchmark lässt sich auch aus einem Java-fähigen HTML-Browser heraus starten (Abb. 2.3). Dann erhält man als Ergebnis die Güte der JVM-Implementierung des Browsers (s. Abbildung 2.2 auf Seite 32). Wie man sieht, hat hier die Windows-Welt die Nase deutlich vorn.

Man kann den Benchmark aber auch mit Hilfe des Applet-Viewers direkt starten. Als Ergebnis erhält man dann die Maßzahl der installierten JVM (s. Tab. 2.1).

Beim embedded CaffeineMark geht die Graphik-Leistung nicht mit ein. Wie der Name schon vermuten lässt, eignet sich diese Maßzahl vor allem für den embedded Bereich und den Server-Bereich, wo keine GUI oder sonstigen graphischen Ausgaben vonnöten sind.

Bei der JDK 1.2-Version wurde einiges für Optimierung und Performance-Steigerung getan. Aber auch die Blackdown-Version 1.1.8 überrascht mit durchweg besseren Resultaten als die Windows-Referenz (sieht man vom Einbruch beim Dialog-Test ab, mit dem alle JDK-Versionen zu kämpfen haben¹⁶).

Die Performance-Steigerung von JKD 1.2 auf 1.3 fällt dagegen moderater aus. JDK 1.4, das in Tabelle 2.1 fehlt, liegt auf gleichem Niveau wie JDK 1.3. Dies deutet daraufhin, dass man hinsichtlich weiterer Performance-Sprünge keine Wunder mehr erwarten darf.

¹⁶Das liegt vermutlich daran, dass der Graphik-Test auf AWT beruht.

Tabelle 2.1: CaffeineMarks (CM3.0)

	JDK 1.1 Win 95	JDK 1.1.8 Blackdown	JDK 1.1.8 IBM	JDK 1.2.2 Blackdown	JDK 1.3 Sun	JDK 1.3 IBM
Sieve	100	239	1366	832	628	1578
Loop	100	299	9199	1315	2418	10608
Logic	100	284	196316	2239	1711	204019
String	100	300	2601	479	3714	5658
Float	100	191	2875	718	1932	7355
Method	100	240	3794	1474	1628	6266
Graphics	100	200	398	121	92	194
Image	100	134	195	22	30	15
Dialog	100	23	25	22	18	19
CM 3.0	100	177	1726	346	485	1541
embedded	100	256	6420	1037	1766	9808

Deutlicher Sieger nach Punkten ist die IBM-Implementierung. Sie wird damit ihrem Ruf als schnellste VM gerecht. Vor allem die herausragenden Logic-Werte lassen darauf schließen, dass hier der JIT-Compiler seine Stärken ausspielen konnte. Die Werte waren derart gut, dass als CaffeineMark lediglich ein N/A (Überlauf) angezeigt wurde und von Hand nachgerechnet werden musste.

Wer also mit Performance-Problemen zu kämpfen hat oder einen älteren Rechner sein eigen nennt, ist mit dem JDK von IBM gut bedient. Wer dagegen lieber auf das Original von Sun (das ja eigentlich von Blackdown portiert wurde) vertraut, fährt mit JDK 1.3 oder 1.4 auch nicht schlecht, da bei Problemen gern auf diese Version als Referenz verwiesen wird.

2.4.9 Kaffe OpenVM

*Kaffe*¹⁷ ist eine komplett eigenständige PersonalJava 3.0 Implementierung, die unter der GPL (GNU General Public License) zur Verfügung steht, sowohl in Source-Form als auch schon vorcompiliert für eine Reihe von Prozessoren (z. B. i386, Sparc, Alpha, Motorola 68K, PowerPC, MIPS, PA-RISC, StrongARM, MIPS) und Plattformen (z. B. Linux, FreeBSD, HP-UX, Solaris, OSF/1, AmigoOS, AIX, Irix). Sie kommt zwar in der Performance nicht an die IBM-JVM heran, ist aber trotzdem deutlich schneller als die Blackdown-Implementierung (und auf der Sun sogar schneller als das Sun-JDK). *Kaffe* gibt es von der Firma Transvirtual¹⁸ auch in einer kommerziellen Version.

¹⁷<http://www.kaffe.org>

¹⁸<http://www.transvirtual.com/kaffe.htm>

Auch wenn Kaffe JDK 1.2 noch nicht vollständig unterstützt wird, ist neben AWT 1.1 bereits JavaSofts Swing-Bibliothek verfügbar. Der große Vorteil von Kaffe ist seine Kompaktheit und Konfigurierbarkeit. Die komplette JVM inklusiv aller benötigten Bibliotheken macht ca. 900 KB (164 KB JVM, 743 KB Bibliotheken) aus. Jedoch kann man die JVM bis auf 54 KB abspecken, wenn man nur die Core VM und den Interpreter benutzt und weniger wichtige Elemente wie Threading oder Verifier weglässt. Analog verhält es sich mit den Java-Bibliotheken. Benutzt man nur die Core-Bibliothek, kommt man hier mit 138 KB aus. Zusammen mit der Möglichkeit, Klassen aus dem RAM, ROM oder Flash nachzuladen, eignet sich die Kaffe OpenVM sehr gut für den Embedded Bereich oder für PDAs¹⁹. Und tatsächlich ist Kaffe auch für eine Reihe von exotischen Betriebssystemen, wie sie in diesem Markt anzutreffen sind, verfügbar (z. B. LynxOS, pSOS, VxWorks, Windows CE). Auch bei vielen Linux-Distributionen wie SuSE oder RedHat ist sie mit von der Partie.

Will man Kaffe selber installieren, kann es man als 3,5 MB große Tar-Datei von <http://www.kaffe.org> herunterladen und anschließend in einem beliebigen Verzeichnis auspacken:

```
user@linux: > tar xzvf kaffe-1.0.6.tar.gz ↵
kaffe-1.0.6/
kaffe-1.0.6/CVS/
kaffe-1.0.6/CVS/Root
kaffe-1.0.6/CVS/Repository
kaffe-1.0.6/CVS/Entries
kaffe-1.0.6/ChangeLog
kaffe-1.0.6/AUTHORS
kaffe-1.0.6/FAQ/
...
```

Die weitere Installation ist in beiliegenden README-Datei erklärt:

1. `./configure`
2. `make`
3. `make install`

Normalerweise sollte die Übersetzung der Sourcen problemlos verlaufen. Sollte `make` sich allerdings mit folgender Fehlermeldung verabschieden:

```
...
System.c:415: warning: implicit declaration of function 'time'
System.c:418: warning: implicit declaration of function 'localtime'
System.c:418: invalid type argument of '->'
```

¹⁹Personal Digital Assistant, z. B. PalmPilot, Apple Newton, ...

```
make[3]: *** [System.lo] Error 1
make[3]: Leaving directory '/tmp/kaffe-1.0.6/libraries/clib/native'
...
```

so fügen Sie in die Datei `System.c` (im Verzeichnis `libraries/clib/native`) am Anfang eine `include`-Anweisung ein:

```
#include <time.h>
```

Nach „`make install`“ werden die ausführbare Programme unter `/usr/local/bin` installiert, das üblicherweise zum Standard-Suchpfad gehört. Ist man Kaffe überdrüssig geworden, kann man sich dieser VM mit „`make uninstall`“ wieder entledigen.

Bei vielen Distributionen ist Kaffe schon mit von der Partie, sodass man sich die Installation etwas vereinfachen kann, indem man auf die Administrations-Werkzeuge der jeweiligen Distribution zurückgreift. So ist bei SuSE diese VM unter `/usr/lib/kaffe` zu finden:

```
user@linux: > /usr/lib/kaffe/bin/kaffe -version ↩
Kaffe Virtual Machine
Copyright (c) 1996-2000
Transvirtual Technologies, Inc. All rights reserved
Engine: Just-in-time v3  Version: 1.0.6  Java Version: 1.1
```

Wenn man auf Swing verzichten kann und mit Platz geizen muss, ist Kaffe eine interessante Alternative. Mit Dokumentation und allen Tools kommt man mit ca. 3,5 MB aus, hat dabei aber alles, was man für die Entwicklung braucht. Zum Ausprobieren des Appletviewers kann man das Hello-World-Applet auf der beiliegenden CD verwenden:



Abbildung 2.4: Appletviewer von Kaffe (hinten) versus Sun-JDK (vorne)



Abbildung 2.5: Japhar-Logo

```
user@linux: > cd /cdrom/samples/applet/hello ↵
user@linux: > /usr/lib/kaffe/bin/appletviewer HelloWorld.html
```

In Abbildung 2.4 ist im Vordergrund der Appletviewer des normalen Sun-JDKs (hier: 1.4) zu sehen, während im Hintergrund der Appletviewer von Kaffe zu bewundern ist. Die Buchstaben werden hier zwar nicht so schön geschliffen wie bei Sun-Variante angezeigt, aber das Tool erfüllt seinen Zweck. Von der Leistungsfähigkeit liegt Kaffe im Bereich von JDK 1.3 bzw 1.4.

2.4.10 Japhar

Auch Japhar²⁰ (Abb. 2.5) ist eine OpenSource-Implementierung einer JVM, die ohne Hilfe der Sun-Sourcen entwickelt wurde. Im Unterschied zu Kaffe ist Japhar noch in einem frühen Entwicklungsstadium, das nur die JVM mitbringt und auf die Kern-Bibliotheken (Core Libraries) von Sun oder anderen Herstellern (z. B. GNU Classpath, s. nächster Abschnitt) angewiesen ist. Im Gegensatz zu Kaffe handelt es sich bei Japhar um eine rein nicht-kommerzielle Lösung, die unter der LGPL steht.

2.4.11 GNU Classpath

Dies ist keine JVM, sondern eine Sammlung wichtiger JDK-Bibliotheksteile mit dem Ziel, eine komplette Java-Kern-Bibliothek (Core Library) zusammenzustellen, die frei modifiziert, angepasst und weiterverbreitet werden darf.²¹

GNU Classpath ist ein relativ junges Projekt und steht unter der LGPL. Damit wird die Entwicklung und Verbreitung freier JVMs (wie z. B. Japhar, s. voriger Abschnitt) um einiges erleichtert, da man nicht mehr auf Suns Java-Bibliotheken angewiesen ist.

Nach dem Herunterladen und Auspacken der Tar-Datei werden die Sourcen mit `configure` und `make` übersetzt:

²⁰<http://www.japhar.org>

²¹<http://www.fsf.org/software/classpath/classpath.html>

```
user@linux: > ./configure ↵
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
...
user@linux: > make install ↵
make[1]: Entering directory '/tmp/classpath-0.03/lib'
top_builddir=.. /bin/sh ./gen-classlist.sh standard
/usr/bin/jikes -nowarn +F -classpath ../../vm/current:: -d . @classes
...
```

Stürzt die Compilation beim Aufruf des Jikes-Compilers²² (s. nächsten Abschnitt) mit einem *Segmentation Fault* ab, haben Sie vermutlich eine zu alte Version auf Ihrem Rechner liegen. So kommt SuSE 7.3 noch mit Version 1.14 daher, während aber mindestens Jikes 1.15 vorausgesetzt wird. In diesem Fall überspringen Sie diesen Abschnitt und nehmen sich das Kapitel 2.5.1 „jikes“ zuerst vor. Danach können Sie ihr Glück wieder mit `./configure` versuchen.

War die Compilation erfolgreich, wird der GNU-Classpath über `make install` (als root) installiert

```
linux: # make install ↵
Making install in lib
make[1]: Entering directory '/tmp/classpath-0.03/lib'
top_builddir=.. /bin/sh ./gen-classlist.sh standard
...
```

Die Dateien werden unter `/usr/local/classpath` installiert und können mit `make uninstall` wieder entfernt werden, wenn man jikes wieder loswerden will.

2.4.12 Open Runtime Platform (ORP)

Die Open Runtime Platform ist eine VM, die auf den GNU Classpath aufbaut. Über <http://orp.sourceforge.net/> kann sie als Tar-Datei heruntergeladen werden. Nach dem Auspacken wird sie über `make` übersetzt:

```
user@linux: > tar xzf orp-1.0.9.tgz ↵
user@linux: > cd orp-1.0.9 ↵
user@linux:~/orp-1.0.9 > make ↵
cd mains/orp; make -f Makefile dbg;
make[1]: Entering directory '/tmp/orp-1.0.9/mains/orp'
...
```

²²Der normale Compiler aus dem Sun-JDK kann leider nicht für die Übersetzung verwendet werden, da er die Klassen aus `java.lang.Object` nicht übersetzt

Sollten Sie bei der Kompilation die Fehlermeldung „cannot find lglib...“ erhalten, fehlt vermutlich das Paket „glib-devel“, das die statische Bibliotheken und Header-Dateien der glib-Bibliothek enthält. Bei SuSE ist dieses Paket unter „Development/Libraries“ zu finden.

Nach der Kompilation gibt es eine Debug-Version und eine optimierte Version, die unter `mains/orp/Linux/dbg` und `mains/orp/Linux/opt` zu finden sind. Um nicht jedesmal den kompletten Pfad eingeben zu müssen, nehmen wir `orp` in den Suchpfad der Shell mit auf:

```
linux: # export PATH=$PATH:$PWD/mains/orp/Linux/opt ↵
linux: # orp -version ↵
Open Runtime Platform, version 1.0.9
```

Zur Kontrolle, ob der Suchpfad richtig aufgesetzt ist, rufen wir `orp` mit der Option „-version“ auf. Um `orp` zusammen mit dem GNU-Classpath auszuprobieren, erstellen wir ein kleines Hello-World-Beispiel:

```
public class Hello {
    public static void main (String args[]) {
        System.out.println("Hello World!");
    }
}
```

Für die Kompilation dieses Einsteiger-Beispiels können wir `jikes` hernehmen, den wir im nächsten Kapitel noch näher unter die Lupe nehmen wollen:

```
user@linux: > jikes -classpath /usr/local/classpath/share/classpath Hello.java ↵
```

Als Classpath geben wir dabei den GNU Classpath an, da `jikes` keine eigene JDK-Bibliotheken mitbringt. Danach lässt sich dieses einfache Programm mit `orp` aufrufen:

```
user@linux: > orp -classpath /usr/local/classpath/share/classpath:. Hello ↵
```

Allerdings lässt sich nicht jedes Java-Programm starten, da der GNU Classpath noch nicht komplett ist. So führte der Versuch, den CaffeineMark-Benchmark zu starten zu einem

```
No Java handlers found
Uncaught exception:
java.lang.UnsatisfiedLinkError
    at gnu/java/awt/peer/gtk/GtkMainThread.run
    at java/lang/Thread.run
```

Aber es ist ein Anfang und einfache Programme laufen mit der Kombination ORP / GNU Classpath. Wenn man sich einen Überblick über den aktuellen Stand

des GNU Classpaths verschaffen will, hilft neben dem Blick auf die Homepage²³ auch ein Blick in die TODO-Liste, die mit GNU Classpath ausgeliefert wird, für ORP lohnt ein Blick in die mitgelieferte „`readme.txt`“-Datei.

2.5 Welcher Compiler?

Auch wenn die diversen JDK bereits einen Compiler enthalten, gibt es einige interessante Compiler, die als Ersatz oder sinnvolle Ergänzung herhalten können.

2.5.1 jikes

`jikes`²⁴ ist ein Java-Compiler aus den IBM-Forschungslabors, der dankenswerterweise ab der Version 1.12 als OpenSource zur freien Verfügung steht. `jikes` ist vor allem auf Geschwindigkeit des Übersetzungsvorgangs getrimmt und in C++ geschrieben. Neben einem erheblich schnelleren Übersetzungsvorgang (ca. 8 – 10-mal schneller als der Sun-Compiler) zeichnet er sich auch durch bessere Fehlermeldungen als das Sun-Original aus. Dafür ist der erzeugte Code nicht optimiert und kann auch nicht optimiert werden (trotz vorhandener „-O“-Option). Somit wird man `jikes` hauptsächlich zur Entwicklung (und auf älteren Rechnern) einsetzen, wo es auf kurze Turnaround-Zeiten und aussagekräftige Fehlermeldungen ankommt und noch nicht so sehr auf die Geschwindigkeit des generierten Codes.

Die Installation sollte ohne größere Probleme ablaufen: komprimierte Tar-Datei auspacken „./configure“ anstoßen, mittels „make“ übersetzen und mit „make install“ installieren – das war’s.

```
linux: # rpm -i jikes-1.15-glibc-2.2.i386.rpm ↩
user cabbey does not exist - using root
...
```

Alternativ kann `jikes` auch als RPM-Datei heruntergeladen und installiert werden (die obige Fehlermeldung kann ignoriert werden). Ansonsten ist er bei den meisten Distributionen schon mit dabei.

Für die Installation über `rpm` werden neben den verschiedenen Architekturen zwei Versionen bereitgestellt: eine für die „glibc-2.1“ und eine für die „glibc-2.2“²⁵. Falls Sie nicht wissen, welche Bibliothek auf Ihrem Linux-System vorhanden ist, können sie es mit:

```
linux: # rpm -q glibc ↩
glibc-2.2.4-21
```

²³<http://www.fsf.org/software/classpath/classpath.html>

²⁴<http://www.jikes.org>

²⁵glibc ist eine System-Bibliothek, die von den meisten C-Programmen benötigt wird.

abfragen. In unserem Fall brauchen wir die „glibc-2.2“-Version.

Um mit `jikes` erfolgreich arbeiten zu können, sollte das aktuelle Verzeichnis („.“) im `CLASSPATH` enthalten sein, da der Compiler sonst die Class-Dateien darin nicht findet. Daneben kennt `jikes` noch `JIKESPATH`, in dem die Jar- oder Zip-Datei der JDK-Bibliotheken stehen sollte (üblicherweise `classes.zip` oder `rt.jar`), z.B.:

```
export JIKESPATH=/usr/java/j2sdk1.4.0/jre/lib/rt.jar:.
```

Ohne diese Angabe kann es vorkommen, dass `jikes` die üblichen JDK-Packages wie z.B. `java.lang` oder `java.util` nicht findet, da er keine Bibliotheken mitbringt, sondern auf ein vorhandenes JDK angewiesen ist.

Eine weitere Beschleunigung des Edit-/Compile-Zyklus wird neben der schnellen Übersetzung auch durch den inkrementellen Build-Mechanismus erreicht. Gestartet wird der Compiler über die Option „++“. Nach der Übersetzung der Sourcen beendet sich der Compiler nicht, sondern wartet, bis die -Taste gedrückt wird. In der Zwischenzeit kann man seinen Code erweitern und bereinigen, ehe man `jikes` wieder mit einem Tastendruck weiterlaufen lässt. Dabei überprüft `jikes` die Abhängigkeiten und übersetzt nur die Java-Dateien, die tatsächlich nötig sind. Dies macht `Makefiles` für die Übersetzung überflüssig.

Apropos `Makefile`: über die Option „+M“ kann man `jikes` veranlassen, die Abhängigkeiten in `Makefiles` abzulegen.

2.5.1.1 Jikes-Optionen

`jikes` besitzt neben den Standard-Optionen eine Reihe weiterer Optionen, die ihn von anderen Compilern unterscheidet:

- **-help** gibt eine Übersicht und Kurzbeschreibung der möglichen Optionen aus.
- **-d dir** legt die Klassen unter dem angegebenen Verzeichnis ab.
- **-depend, -xdepend** übersetzt *alle* Klassen; ansonsten werden nur die benötigten Klassen übersetzt, bei denen sich etwas geändert hat.
- **-g** legt Debug-Informationen im Code ab.
- **-nowarn** unterdrückt Warnungen; dies kann hilfreich sein, wenn mit dem `jikes`-Compiler generierte Klassen übersetzt werden sollen, die sich nicht warnungs-frei übersetzen lassen.
- **-o** verhindert nur, dass die Zeilennummerierung im Code abgelegt wird.
- **-verbose** zeigt die Dateinamen an, die gerade gelesen und geschrieben werden.

-Xstdout gibt das Ausgabe-Listing auf der Konsole aus.

++ inkrementeller Mode

+B überprüft nur die Java-Syntax, ohne Code zu generieren.

+D, **+E** gibt die Fehlermeldungen im „Emacs“-Mode aus. Dabei reicht in Emacs (oder auch in anderen, ähnlichen IDEs) ein Klick auf die Fehlermeldung aus, und der Editor zeigt die fehlerhafte Codestelle an.

Bei **+D** wird im Gegensatz zu **+E** die Ausgabe nicht gepuffert, was bei einem (unwahrscheinlichen) Absturz von `jikes` hilfreich sein kann, um noch die letzte Fehlermeldung zu sehen.

+DR=Dateiname generiert einen Abhängigkeitsreport in die angegebene Datei.

+F, **+U** überprüft die Abhängigkeiten ausschließlich (**+F**) bzw. einschließlich (**+U**) Zip- und Jar-Dateien

+Kname=Datentyp ersetzt den Typ *name* durch *Datentyp*, wobei *Datentyp* nur ein primitiver Datentyp sein darf. So ersetzt „+Kreal=double“ überall den Datentyp „real“ durch den tatsächlichen Datentyp „double“.

+M erzeugt Abhängigkeiten im `Makefile`-Format

2.5.1.2 Abhängigkeiten

Normalerweise nimmt Jikes eine einfache Überprüfung der Abhängigkeiten vor, die in den meisten Fällen auch wie vorgesehen funktioniert. Für eine genauere Abhängigkeitsanalyse gibt es die Optionen „**+F**“ bzw. „**+U**“.

Im inkrementellen Modus (Option „**++**“) übersetzt Jikes die übergebenen Klassen und wartet dann auf weitere Befehle. Mit  werden die geänderten Klassen erneut übersetzt. Mit der `q`-Taste (für „quit“) kann der inkrementelle Modus beendet werden. Auf diese Weise kann man sehr kurze Turnaround-Zeiten erreichen: Editieren –  – Editieren –  – `q`:

```
user@linux: > jikes ++ Hello.java 
Incremental: Enter to continue or q + Enter to quit: Enter

ok...

Found 1 syntax error in "Hello.java":

    31.          System.out.println("Hello " + System.getProperty("user.name"));
           ^-----^
```

```

*** Syntax: ")" inserted to complete StatementExpression

Incremental: Enter to continue or q + Enter to quit: Enter

ok...

Incremental: Enter to continue or q + Enter to quit: q
user@linux: >

```

Sobald wir die Datei „Hello.java“ editiert und verändert haben, brauchen wir lediglich die -Taste in dem Fenster zu betätigen, in dem wir jikes gestartet haben. Als Beispiel haben wir hierbei das HelloWorld-Beispiel aus dem vorigen Abschnitt reaktiviert, das wir jetzt etwas personalisiert wollen: es soll nicht lediglich ‘Hello World‘ ausgegeben werden, sondern das Programm soll uns mit unserem (Benutzer-)Namen ansprechen:

```

public class Hello {
    public static void main (String args[]) {
        System.out.println("Hello " + System.getProperty("user.name"));
    }
}

```

2.5.1.3 Fazit

Insgesamt ist Jikes ein wertvoller und schneller Compiler. Nicht umsonst wird er bei vielen Distributionen (SuSE, Debian, FreeBSD, Mandrake-Linux, Red Hat und Linux PPC) standardmäßig mit ausgeliefert (wenn auch leider nicht immer in der aktuellen Version). Wenn Jikes beim Übersetzungsvorgang Fehler meldet, die andere Compiler „durchgehen“ lassen, so handelt es sich nicht um Fehler von Jikes, sondern es liegt in den allermeisten Fällen daran, dass Jikes die Spezifikation sehr viel strenger einhält als andere Compiler.

2.5.1.4 Jikes Debugger (JD)

Von AlphaWorks²⁶ gibt es auch den JikesDebugger²⁷, ein graphischer Debugger, der in Java geschrieben ist. Leider ist der Jikes Debugger auf JDK 1.1 beschränkt. Wie es zur Zeit aussieht, wird es keine Version für JDK 1.2 oder höher geben.

²⁶<http://www.alphaworks.ibm.com/>

²⁷<http://www.alphaworks.ibm.com/tech/JikesDebugger>

2.5.2 GNU-Java-Compiler (GCJ)

Einen interessanten Ansatz verfolgt `gcj`²⁸, der Java-Compiler aus der GNU-Compiler-Familie, der Java-Quellen direkt in Maschinencode übersetzen kann. Aber nicht nur das. Er kann:

- ❑ Java-Quellen direkt in Maschinencode übersetzen
- ❑ Java-Quellen in Bytecode übersetzen (.java → .class)
- ❑ Java-Bytecode in Maschinencode übersetzen
- ❑ Maschinencode für andere Systeme erzeugen (Cross-Compiler)

Dies macht ihn nicht nur für schwächere Rechner interessant, sondern auch für den Einsatz im Embedded Bereich, da er auch mit dem GNU-C-Compiler (`gcc`) und GNU-Debugger (`gdb`) zusammenarbeitet, die hier sehr oft anzutreffen sind.

2.5.2.1 Installation

Normalerweise ist der `gcj` bei der GNU Compiler Collection und den meisten Distributionen mit dabei. Sollten Sie allerdings noch die Version 2.95 besitzen, können Sie zwar `gcj` aufrufen, aber von der Verwendung wird auf der `gcj`-Homepage abgeraten. Um festzustellen, ob und welche Version installiert ist, rufen Sie `gcj` (oder auch `gcc`) mit der Option „`--version`“ auf.

```
user@linux: > gcj --version ↵
2.95.3
user@linux: >
```

In diesem Fall sollten Sie auf die Version 3.0 aktualisieren. Nach dem Herunterladen und Auspacken der GNU-Compiler-Collection (ca. 20 MB) sind es die üblichen Schritte, um die Programme zu installieren:

```
user@linux: > ./configure ↵
...
user@linux: > make ↵
...
user@linux: > make install ↵
...
```

Alternativ kann der GNU-Compiler auch über <http://rpmfind.net> als RPM-Paket gesucht und installiert werden. Dies dürfte vor allem auf älteren Rechnern erheblich schneller gehen, da die Compilation über `make` doch etwas länger dauert.²⁹

²⁸<http://gcc.gnu.org/java/>

²⁹ca. 40 Minuten auf einem Pentium-III (850 MHz)

2.5.2.2 Experimentelle Installation

Manche Distributionen vertrauen noch auf den GCC 2.95, bieten aber die Version 3.0 als „experimental“ an (bei SuSE 8.0 ist diese Version unter `/opt/experimental` zu finden). In diesem Fall müssen Sie dafür sorgen, dass sich diese Version ganz vorne im Suchpfad befindet:

```
user@linux: > export PATH=/opt/experimental/bin:$PATH
user@linux: > export LD_LIBRARY_PATH=/opt/experimental/lib:$LD_LIBRARY_PATH
user@linux: > gcj --version
3.0.4
```

Wenn Sie später beim Aufruf des übersetzten Java-Programms eine Fehlermeldung:

```
error while loading shared libraries: libgcc_s.so.1...
```

erhalten, haben Sie den Suchpfad für die Bibliothek (`LD_LIBRARY_PATH`) nicht richtig aufgesetzt.

2.5.2.3 Beispiel

Um den `gcj` auszuprobieren, greifen wir wieder auf das altbewährte HelloWorld-Beispiel zurück:

```
user@linux: > gcj --main=Hello -o Hello Hello.java
```

Damit haben wir das HelloWorld-Beispiel direkt in ausführbaren (nativen) Code übersetzt, der sich ohne Java-Interpreter starten lässt:

```
user@linux: > ./Hello
Hello world!
```

Die Ausführungsgeschwindigkeit spielt bei diesem Beispiel noch keine Rolle. Was aber spürbar schneller geht, ist der Programmstart, da keine Java-Interpreter gestartet werden muss. Dafür haben wir die Plattformunabhängigkeit verloren, da dieser Code nur unter Linux lauffähig ist.

2.5.2.4 gcj-Optionen

Der `gcj` erwartet als Eingabe Java-, Class-, Jar- oder Zip-Dateien. Neben den normalen Optionen (s. „man gcc“) lässt er sich über folgende Optionen steuern:

-help gibt eine Kurz-Übersicht der möglichen Optionen aus.

-C übersetzt Java-Quellen in Byte-Code (Class-Dateien).

--output-class-dir=*dir*, -d *dir* spezifiziert das Ausgabe-Verzeichnis der übersetzten Class-Dateien.

-bounds-check schaltet die Überprüfung der Feld-Grenzen ein (Standard-Einstellung).

-no-bounds-check schaltet die Überprüfung der Feld-Grenzen aus.

--main=*Klasse* spezifiziert den Einstiegspunkt, den `main`-Aufruf, der übersetzten Dateien. Ohne diese Angabe erhalten Sie bei der Übersetzung folgende Fehlermeldung:

```
user@linux: > gcj -o Hello Hello.java ↩
/usr/lib/crt1.o: In function '_start':
/usr/lib/crt1.o(.text+0x18): undefined reference to 'main'
collect2: ld returned 1 exit status
```

-Idir hängt das angegebene Verzeichnis an den Classpath³⁰ an.

--classpath=*path*, -classpath *path* verwendet den angegebenen Pfad als Classpath (überschreibt den voreingestellten Classpath).

--CLASSPATH=*path*, -CLASSPATH *path* verwendet den angegebenen Pfad als Classpath (wird an den voreingestellten Classpath angehängt).

2.5.2.5 Übersetzung von Class-Dateien

Wie schon erwähnt, verarbeitet `gcj` auch bereits Java-Dateien, die als Bytecode vorliegen:

```
user@linux: > gcj --main=Hello -o Hello Hello.class ↩
user@linux: > ./Hello
Hello World!
```

Damit können auch Java-Klassen, von denen kein Quellcode vorhanden ist, übersetzt werden.

2.5.2.6 Einschränkungen

Wenn man Java nur als Programmiersprache und nicht als Weltanschauung betrachtet, bietet der GNU-Java-Compiler einen interessanten Ansatz. Durch die Einbindung in die GNU Compiler Collection (GCC) ist die Anbindung anderer Sprachen wie C oder C++ einfacher möglich als über das JNI-Interface (s. Kap. 23), das aber auch unterstützt wird.

³⁰Auf die Bedeutung und das Setzen des Classpaths werden wir später in Kapitel 4 noch näher eingehen.

Der `gcj` steht in vielen Dingen noch am Anfang. So wird AWT nur rudimentär unterstützt, Swing noch gar nicht. Aber es gibt schon einige Projekte, die diesen Compiler einsetzen (s. <http://gcc.gnu.org/java/done.html>). Der Performance-Gewinn, der sich damit erzielen lässt, wird zwar durch den Verlust der Plattform-Unabhängigkeit erkauft, aber nicht immer wird diese Unabhängigkeit benötigt.

In Kapitel 20.4 werden wir nochmals auf den `gcj` zurückkommen, um ihn an einem etwas größeren Beispiel auszuprobieren. Wir werden dort sehen, welche Einsparungen in der Ausführungszeit möglich sind.

2.6 Welche Bibliotheken

Mit dem JDK selbst wird bereits eine umfangreiche Bibliothek³¹ mitgeliefert, die das Programmierer-Leben vereinfachen. Fast alle Packages beginnen mit `java`, einige (vor allem in JDK 1.2) auch mit `javax`. Diese waren ursprünglich als Erweiterung vorgesehen, haben aber ihren Weg in die Core-Bibliothek gefunden. Um nun nicht die ganzen Programmierer zu verärgern, weil sie ihre `import`-Anweisungen umbenennen müssen, hat man diese Namensgebung beibehalten. Neben dem JDK gibt es eine Reihe weiterer Bibliotheken von Sun Microsystems, die sich in zwei Kategorien aufteilen lassen:

Optional Packages sind Bibliotheken für bestimmte Einsatzbereiche inkl. Implementierung und Test-Suite, die von Sun bezogen und eingesetzt werden können.

Enterprise Technologien zielen auf den Server-Bereich. Oftmals handelt es sich dabei lediglich um eine Spezifikation, die die Schnittstellen (mit Hilfe von Interfaces) beschreibt.

Falls Sie in der nun folgenden Übersicht mit dem einen oder anderen Fachbegriff nichts anfangen können, ignorieren Sie ihn einfach. Sie soll Ihnen zum einen ein Gefühl vermitteln, was es standardmäßig alles gibt, zum anderen ist sie für späteres Nachschlagen gedacht, wenn Sie eine bestimmte Bibliothek suchen.

2.6.1 Optional Packages

Die „Optional Packages“ (früher als „Standard Extensions“ bezeichnet) können von Sun Microsystems³² heruntergeladen werden.

³¹die sogenannte „Core Class Library“

³²<http://java.sun.com/products/0V/stdExt.html>

Tabelle 2.2: Optional Packages

Bibliothek	Kurzbeschreibung
JAF (JavaBeans Activation Framework)	regelt das Zusammenspiel von JavaBeans
JNDI (Java Naming and Directory Interface)	JNDI ist als einheitliche Schnittstelle zu verschiedenen Namensdiensten wie LDAP (Lightweight Directory Access Protocol), DNS (Domain Name System), NIS (Network Information System), NDS (Novell Directory Services) oder anderen gedacht.
JavaMail	Dieses API bietet Unterstützung zum Abschicken von Mails. Als Protokolle werden neben SMTP (Simple Mail Transfer Protocol) auch POP3 (Post Office Protocol) verwendet. Die Unterstützung anderer Protokolle (MAPI, NNTP, Lotus Notes, ...) überlässt Sun anderen Herstellern.
InfoBus	Der „Informations Bus“ dient zum Austausch von Daten zwischen JavaBeans.
Java3D	Die Einsatzgebiete dieser 3D-API sind hauptsächlich Spiele und 3D-Anwendungen. Es ist abhängig von der zugrunde liegenden Graphikfähigkeit des Betriebssystems und baut auf der OpenGL-Schnittstelle auf. Derzeit ist es für Solaris und Windows verfügbar, es gibt aber auch einen Linux-Port von Blackdown.
JMF (Java Media Framework)	Hierbei handelt es sich um ein Framework, um Multimedia-Daten (Video, Audio) abspielen und manipulieren zu können.
JAI (Java Advanced Imaging)	Kompatibel mit dem Java2D-API (Bestandteil des JDKs) bietet es Erweiterungen für die Bildbearbeitung.
JavaServlet	Servlets sind zur Generierung von dynamischen Webseiten und als Ablösung von CGI gedacht.
JCE (Java Cryptography Extension)	Diese Java-Bibliothek bietet Unterstützung von Verschlüsselungsalgorithmen mit öffentlichen und privaten Schlüsseln. Implementiert sind einige verbreitete kryptographische Algorithmen (DES, Blowfish, Diffie-Hellman); für die Einbindung anderer Algorithmen gibt es entsprechende Schnittstellen.

Tabelle 2.2 – Fortsetzung

Bibliothek	Kurzbeschreibung
JavaHelp	JavaHelp unterstützt den Entwickler bei der Erstellung von Hilfe-Systemen. Die Ausgabe ist HTML-basiert und damit auf allen Plattformen mit HTML-Unterstützung lauffähig.
RMI-IIOP (RMI over IIOP)	Hierbei wird das RMI ³³ -Protokoll auf das IIOP ³⁴ abgebildet, das das zugrunde liegende Protokoll von CORBA (die „Common Object Request Broker Architecture“ der OMG ³⁵) ist.
JAAS (Java Authentication and Authorization Service)	Hierbei handelt es sich um ein Framework zur Benutzer-Authorisierung. Es enthält eine Java-Implementierung der PAM ³⁶ -Architektur und unterstützt Benutzer-, Gruppen- und Rollen-basierte Zugriffskontrolle.
JSSE (Java Secure Socket Extension)	enthält Support für SSL (Secure Socket Layer) und TLS (Transport Layer Security)
JavaComm (Java Communications API)	API, um die serielle und parallele Schnittstelle ansprechen zu können
JMX (Java Management API)	JMX ist eine Spezifikation für das Management von Netzwerken, die meist über SNMP (Simple Network Management Protocol) administriert werden (z. Zt. nur als Interface ohne Implementierung verfügbar).

2.6.2 Enterprise Technologien

Mit den *Enterprise Technologien* zielt Sun auf den Server-Markt. Teilweise sind die hier aufgeführten Bibliotheken schon bei den *Optional Packages* (s. voriger Abschnitt) enthalten, teilweise werden sie mit dem J2EE-Umfang (s. <http://java.sun.com/j2ee/>) ausgeliefert.

³³Remote Method Invocation

³⁴Internet Inter-Orb Protocol

³⁵Object Management Group

³⁶Pluggable Authentication Module

Tabelle 2.3: Enterprise Technologien

Bibliothek	Kurzbeschreibung
EJB (Enterprise Java Beans)	Nicht zu verwechseln mit den Java Beans (mit denen sie nichts gemein haben), erlauben es die Enterprise Java Beans dem Entwickler, sich auf die eigentliche Business-Logik in einer verteilten Client-Server-Architektur zu konzentrieren. Wir werden später noch in einem eigenen Kapitel (s. Kap. 17 auf Seite 425) darauf eingehen. URL: http://java.sun.com/products/ejb/index.html
JSP (Java Server Pages)	Ähnlich wie PHP ³⁷ (Hypertext Preprocessor) oder ASP (Active Server Pages) von Microsoft erlaubt JSP die direkte Einbettung von (Java-)Code in einer HTML-Seite. Im Gegensatz zu ASP läuft es nicht nur innerhalb der Microsoft-Welt, sondern auf sämtlichen Java-Plattformen. URL: http://java.sun.com/products/jsp/index.html
Java Servlet	als Alternative zu CGI-Lösungen gedacht (s. „Optional Packages“) URL: http://java.sun.com/products/servlet/index.html
JNDI (Java Naming and Directory Interface)	Schnittstelle für Namensdienste (s. „Optional Packages“) URL: http://java.sun.com/products/jndi/index.html
Java IDL	IDL (Interface Definition Language) ist eine sprachunabhängige Beschreibung von Datenstrukturen, die für die Schnittstellenbeschreibung eines CORBA-Programms verwendet wird. Diese Beschreibung lässt sich auf verschiedenen Sprachen abbilden (z. B. C++, Smalltalk, COBOL), mit dem IDL-to-Java-Compiler (idlj) von Sun auch auf Java. URL: http://java.sun.com/products/idl/index.html

³⁷<http://www.php.net/>

Tabelle 2.3 – Fortsetzung

Bibliothek	Kurzbeschreibung
JDBC (Java Database Connectivity)	JDBC ist Bestandteil von JDK 1.2 und stellt eine Datenbank-unabhängige Schnittstelle für relationale Datenbanken (basierend auf SQL 92) zur Verfügung. URL: http://java.sun.com/products/jdbc/index.html
JMS (Java Message Service)	z. Zt. nur als Schnittstelle zum Austausch von „Messages“ verfügbar; Grundidee dabei ist, dass der Absender nicht warten muss, bis die Nachricht angekommen ist, sondern dies JMS überlassen kann. URL: http://java.sun.com/products/jms/index.html
JTA (Java Transaction API)	zur Unterstützung von verteilten Transaktionen URL: http://java.sun.com/j2ee/transactions.html
JTS (Java Transaction Service)	ähnlich wie JTS dient es zur Unterstützung von verteilten Transaktionen URL: http://java.sun.com/j2ee/transactions.html
JavaMail	zum Verschicken von Mails (s. „Optional Packages“) URL: http://java.sun.com/products/javamail/index.html
RMI/IIOP	RMI over IIOP (s. „Optional Packages“) URL: http://java.sun.com/j2ee/corba/index.html

2.6.3 JavaMail

Als Beispiel für ein „Optionales Paket“ wollen wir uns die JavaMail 1.2 API von <http://java.sun.com/products/javamail/> herunterladen. Nach Bestätigen der Lizenz-Bedingungen kann das Paket als Zip-Datei (ca. 2 MB) heruntergeladen und installiert werden:

```
user@linux: > unzip javamail-1_2.zip
Archive:  javamail-1_2.zip
  creating: javamail-1.2/
  creating: javamail-1.2/docs/
  creating: javamail-1.2/docs/javadocs/
  ...
```

Nach dem Auspacken empfiehlt es sich, die „README.txt“-Datei durchzulesen, die einen Kurzüberblick über die Dokumentation und Dateistruktur gibt. Hier wird auch erläutert, wie die Umgebung bzw. Classpath aufgesetzt werden sollte:

```
user@linux: > cd javamail-1.2
user@linux:javamail-1.2 > export CLASSPATH=$CLASSPATH:$PWD/mail.jar
```

Einige Beispiel-Programme befinden sich im demo-Verzeichnis, von denen wir „msgsend.java“ übersetzen wollen:

```
user@linux: > cd demo
user@linux:demo > javac msgsend.java
```

Um mit diesem Java-Programm auch tatsächlich Mails verschicken zu können, wird zusätzlich noch das JavaBeans Activation Framework (JAF)³⁸ benötigt, um nicht eine Fehlermeldung, wie „java.lang.NoClassDefFoundError: javax/activation/DataSource“, zu erhalten. Die Installation sieht ähnlich wie bei JavaMail aus:

1. Auspacken

```
user@linux: > unzip jaf1_0_1.zip
```

2. Umgebung (CLASSPATH) setzen:

```
user@linux: > cd jaf-1.0.1
user@linux: > export CLASSPATH=$CLASSPATH:$PWD/activation.jar
```

Nach diesem Intermezzo gehen wir wieder in das demo-Verzeichnis von JavaMail zurück und starten das msgsend-Beispiel:

```
user@linux: > java msgsend
To: oliver@localhost
Subject: Test von "msgsend"
Diese Mail wurde Ihnen präsentiert von:
MSGSEND
⌘
Mail was sent successfully.
```

Das Programm funktioniert ähnlich wie das mail-Programm, d. h., die Mail wird mit der Tastenkombination [⌘] abgeschlossen. Voraussetzung, dass die Mail bei Ihnen ankommt ist, dass sie a) auch „oliver“ heißen (ansonsten einfach Ihren Account einsetzen) und b) Ihr Rechner SMTP spricht. Falls keine Mail ankommt, schauen Sie nach, ob sie Sendmail³⁹ konfiguriert haben.

Um auszuprobieren, ob die Mail tatsächlich angekommen ist, können Sie einen Mail-Client wie z. B. kmail des KDE-Desktop verwenden (s. Abb. 2.6).

³⁸<http://java.sun.com/products/javabeans/glasgow/jaf.html>

³⁹Sendmail wird bei vielen Distributionen als Standard-MTA (Mail Transport Agents) eingesetzt.



Abbildung 2.6: über JavaMail verschickte Mail

2.6.4 Weitere Bibliotheken

Neben diesen aufgeführten Bibliotheken gibt es eine ganze Reihe von weiteren kommerziellen und frei verfügbaren Bibliotheken, deren Aufzählung den Rahmen dieses Buches sprengen würde (und auch nur unvollständig sein könnte bei der Dynamik im Java-Markt).

2.7 Weitere Informationsquellen

Neben Javasoft (<http://java.sun.com/>) ist vor allem Blackdown.org (<http://www.blackdown.org/>) die erste Anlaufstelle für Fragen rund um „Java Linux“. Aber es gibt noch weitere Seiten im Internet:

- ❑ <http://www.volano.com/report/index.html>
Wer außer CaffeineMark (s. Kap. 2.4.8.1 auf Seite 31) weitere Benchmarks zu Java sucht, findet mit dem „Volano Report“ einen Benchmark, der vor allem die Server-Leistung testet.
- ❑ <http://java.sun.com/products/hotspot/>
Wer wissen will, wie die HotSpot-Technologie funktioniert, kann es in diesem White Paper nachlesen.
- ❑ http://java.sun.com/products/OV_stdExt.html
Auf dieser Seite sind die offiziell freigegebenen „Optional Packages“ mit einer kurzen Beschreibung und weiterführenden Links aufgelistet.

2.8 Zusammenfassung

In diesem Kapitel haben wir die Voraussetzung geschaffen, um Java-Programme übersetzen und starten zu können.

- ❑ Zur Java-Entwicklung benötigt man das Java Development Kit (JDK).
- ❑ Das Sun-JDK basiert auf der Portierung von Blackdown.org.
- ❑ Das JDK von IBM zeichnet sich vor allem durch seine Schnelligkeit aus.
- ❑ `jikes` ist ein verdammt schneller und genauer Compiler.
- ❑ Mit dem GNU-Compiler `gcj` lässt sich neben Byte-Code auch Maschinen-Code erzeugen.
- ❑ Die *Optional Packages*, ehemals *Standard Extensions*, sind zusätzliche Java-Bibliotheken für verschiedene Einsatzbereiche.
- ❑ Bibliotheken und Schnittstellen für den Server-Bereich werden unter „*Enterprise Technologien*“ zusammengefasst.

8.4 CVS

Während man mit RCS nur einzelne Dateien versionieren kann, lassen sich mit CVS ganze Datei-Bäume und -Wälder verwalten. Dabei baut CVS auf RCS auf, sodass die gleichen Schlüsselwörter weiterverwendet werden können.

8.4.1 Installation

Genau wie RCS gehört CVS bei allen gängigen Distributionen zur Grundausstattung und kann für lokale Projekte ohne weitere Konfigurationsarbeiten eingesetzt werden. Um festzustellen, ob CVS tatsächlich auf Ihrem Rechner vorhanden ist, rufen Sie es einfach mit der Option „`--help`“ auf:

```
user@linux: > cvs --help ↩
Usage: cvs [cvs-options] command [command-options-and-arguments]
  where cvs-options are -q, -n, etc.
    (specify --help-options for a list of options)
  where command is add, admin, etc.
    (specify --help-commands for a list of commands
     or --help-synonyms for a list of command synonyms)
  where command-options-and-arguments depend on the specific command
    (specify -H followed by a command name for command-specific help)
  Specify --help to receive this message

The Concurrent Versions System (CVS) is a tool for version control.
For CVS updates and additional information, see
  the CVS home page at http://www.cvshome.org/ or
  Pascal Molli's CVS site at http://www.loria.fr/~molli/cvs-index.html
```

Sollte dieser Hilfe-Text ausbleiben und Sie stattdessen ein „command not found“ erhalten, müssen Sie es nachträglich installieren. Sie finden es als RPM-Paket unter der Rubrik „Development / Version Control“ bzw. bei SuSE unter der Serie „d“. Alternativ können Sie das Paket auch als Tar-Ball von <http://www.cvshome.org/> oder als RPM-Paket über <http://rpmfind.net/> herunterladen und auspacken.

8.4.2 Bedeutung

Auch wenn CVS sehr stark Kommandozeilen-orientiert ist, ist es dennoch weit verbreitet. Vor allem im OpenSource-Bereich kommt es sehr häufig zum Einsatz. So findet ein Großteil der Linux-Entwicklung unter CVS statt, aber auch viele Firmen setzen darauf, nicht zuletzt weil es auch für Windows- und Mac-Plattformen verfügbar ist, wenn auch nur als CVS-Client.⁵

⁵Es gibt inzwischen auch einen CVS-Server für NT (s. <http://www.cvsnt.org>).

Der letzte Satz deutet es schon an: CVS ist Server-Client-fähig. Das heißt, man kann im Netz einen CVS-Server installieren, auf den von den unterschiedlichsten Plattformen aus zugegriffen werden kann. Das funktioniert sowohl im lokalen Netzwerk wie auch über das Internet (wie sonst könnte die Linux-Entwicklung so verteilt funktionieren?). Und auch exotische Plattformen wie z.B. LynxOS⁶ bleiben im Gegensatz zu den meisten kommerziellen Produkten nicht außen vor.

8.4.2.1 Vorteile

- ⊕ für kleinere und größere Projekte einsetzbar
- ⊕ große Plattform-Unterstützung
- ⊕ stabil
- ⊕ auch im Netzwerk einsetzbar

8.4.2.2 Nachteile

- ⊖ schwierig zu bedienen
- ⊖ schwerer aufzusetzen als RCS

Die Bedienbarkeit gehört sicherlich nicht zu den Stärken von CVS, auch wenn es inzwischen graphische Frontends gibt. Vor allem als Administrator wird man nicht umhin kommen, sich mit der Kommandozeile anzufreunden.

8.4.3 CVS-Kommandos

8.4.3.1 Überblick

Ehe wir auf einzelne CVS-Befehle eingehen, finden Sie zum späteren Nachschlagen die wichtigsten Kommandos aufgelistet, die während der Arbeit mit CVS benötigt werden. Einen kompletten Überblick erhalten Sie über das Online-Manual (`man cvs` oder `info cvs`).⁷ Mit diesem Vorwissen bewaffnet werden wir danach Stan und Oliver über die Schulter schauen, und sie bei ihrer täglichen Arbeit mit CVS begleiten.

Allen CVS-Kommandos ist gemeinsam, dass sie durch `cvs` eingeleitet werden und folgendes Format aufweisen:

```
cvs [Optionen] Kommandos [Kdo-Optionen] [Argumente]
```

⁶ein Unix-ähnliches Echtzeit-Betriebssystem

⁷Unter <http://cvsbook.red-bean.com/> gibt es auch ein CVS-Buch online.

cvs login

Wenn man von einem CVS-Client auf einen CVS-Server zugreifen will, muss man sich vorher erst über `login` anmelden. Erst dann kann man weitere CVS-Befehle absetzen.

cvs import [-m "Kommentar"] *dir Vendor Release*

Mit diesem Befehl kann ein Projekt in CVS „importiert“ werden. Ohne die Option „-m“ wird man für jede Datei in diesem Projekt aufgefordert, eine Beschreibung anzugeben.

Beispiel (s. a. Kap. 8.4.4.2 auf Seite 172):

```
cvs import Armada Oliver start
```

Als *Vendor* bietet sich der Firmennamen oder der eigene Name an, für *Release* wird meistens „start“ verwendet.

cvs checkout [-r *Release*] [-D *Datum*] *Module...*

Damit wird eine Arbeitskopie angelegt, die die angegebenen Verzeichnisse (Module) enthält. Über die Option „-r“ oder „-D“ kann dabei auf ein bestimmtes Release bzw. Datum zurückgegriffen werden.

cvs update [-d] [*Verzeichnisse*]

Die Änderungen im aktuellen (bzw. den angegebenen) und darunterliegenden Verzeichnissen werden mit dem CVS-Repository⁸ abgeglichen und aktualisiert (d. h. ältere Versionen im Verzeichnis werden durch neuere Versionen aus dem Repository ersetzt). Dabei kann es zu Konflikten kommen (wenn Dateien geändert wurden, die im Repository einen neueren Stand aufweisen). Wurden Verzeichnisse neu angelegt, werden diese mit der Option „-d“ in die Arbeitskopie übernommen.

cvs commit [-m "Kommentar"] [*Dateien...*]

Damit werden die Änderungen der Arbeitskopien ins Repository eingestellt. Vorher macht CVS aber ein Update. Kommt es dabei zu Konflikten, müssen diese erst aufgelöst werden.

cvs add *Dateien...*

Damit werden neue Dateien oder Verzeichnisse für die Übernahme in das CVS-Repository vorgemerkt. Mit einem „`cvs commit`“ werden diese Dateien dann im Repository angelegt.

⁸Das Repository ist die „Datenbank“ von CVS, in der die verschiedenen Versionen und Verwaltungsinformationen abgespeichert werden.

cv^s remove *Dateien...*

Dateien oder Verzeichnisse können mit `remove` entfernt werden. Sie sind aber nach wie vor im Repository vorhanden, erscheinen aber nicht mehr bei einem Checkout.

cv^s tag *Marke Dateien...*

Will man einen Stand markieren, kann dies über das `tag`- oder auch `rtag`-Kommando geschehen. Während `rtag` direkt auf dem Repository arbeitet, bezieht sich `tag` auf die Arbeitskopie.

Die meisten CVS-Kommandos arbeiten rekursiv, d. h., sie beziehen auch Unterverzeichnisse mit ein.

8.4.3.2 Optionen

Neben den Kommando-spezifischen Optionen gibt es allgemeine Optionen, die zwischen „`cvs`“ und dem Kommando stehen können. Die wichtigsten sind:

- q | -Q** sorgt dafür, dass CVS ruhig (quiet) bzw. ganz ruhig (Quiet) ist.
- d *CVS-Rootverzeichnis*** gibt das Verzeichnis an, unter dem das Repository zu finden ist.
- e *Editor*** gibt den Editor an, der für die Eingabe von Log-Kommentaren aufgerufen werden soll. Ohne diese Option wird der `vi` bzw. der eingestellte Standard-Editor aufgerufen.
- n** führt den CVS-Befehl nicht wirklich aus, sondern simuliert ihn nur. Man sieht zwar die Ausgabe des CVS-Kommandos, aber es werden keine Änderungen an irgendwelchen Dateien vorgenommen. Dies ist ideal, um vorher die Wirkung eines Kommandos überprüfen zu können.
- t** (trace) protokolliert die Programmausführung von CVS. Vor allem mit der Option „-n“ ist diese Option hilfreich, um festzustellen, was das eine oder andere CVS-Kommando machen würde.
- x** verschlüsselt die Kommunikation zwischen CVS-Client und -Server.
- z 1-9** gibt den Kompressionsfaktor bei der Übertragung von Dateien an. Der Faktor ist der gleiche wie beim `gzip`-Kommando, da CVS darauf aufbaut (`gzip` muss im Suchpfad vorhanden sein!): 1 bedeutet geringe Kompression (geringe Rechenzeit), 9 steht für die höchste Kompressionsstufe (lange Rechenzeit).

8.4.3.3 Umgebungsvariablen

Die wichtigsten Umgebungsvariablen für CVS sind:

CVSROOT gibt das Verzeichnis an, unter dem das Repository abgespeichert ist (kann mit der Option „-d“ überschrieben werden).

CVSEEDITOR gibt den Editor an, der für die Eingabe der Log-Kommentare aufgerufen wird. Ist diese Umgebungsvariable nicht gesetzt, werden die Variablen VISUAL und EDITOR herangezogen, falls diese nicht gesetzt sind, kommt der vi zum Einsatz.

CVS_IGNORE_REMOTE_ROOT ignoriert die Referenzen auf fremde Rechner. Damit wird verhindert, dass versehentlich eine Netzverbindung geöffnet wird. Dies hilft, die Telefonkosten niedrig zu halten.

CVS_RSH gibt das Übertragungsprotokoll zum CVS-Server an, wenn nicht das CVS-interne pserver-Protokoll verwendet werden soll.

8.4.3.4 Hilfe

Neben dem Online-Manual (`man cvs`) gibt es für die schnelle Hilfe zwischen durch die Option „`--help-commands`“, mit der man eine Liste der CVS-Kommandos erhält und die Option „`--help-options`“, die eine Liste von allgemeinen Optionen ausgibt. Daneben kennen viele CVS-Kommandos zusätzlich eigene Optionen, die *hinter* dem CVS-Befehl aufgeführt werden. Die wichtigste Option, die bei allen CVS-Kommandos gleich ist, ist die Option „-H“. Mit ihr hat man einen Überblick über die möglichen Optionen mitsamt ihrer Bedeutung:

```
user@linux: > cvs -H diff
```

Damit gelangt man zum `diff`-Kommando, das sämtliche Optionen anzeigt, die bei „`cvs diff`“ möglich sind (das `diff`-Kommando zeigt die Unterschiede zweier verschiedener Versionen an):

```
Usage: cvs diff [-lNR] [rcsdiff-options]
      [[-r rev1 | -D date1] [-r rev2 | -D date2]] [files...]
      -l      Local directory only, not recursive
      -R      Process directories recursively.
      -D d1    Diff revision for date against working file.
      -D d2    Diff rev1/date1 against date2.
      -N      include diffs for added and removed files.
      -r rev1  Diff revision for rev1 against working file.
      -r rev2  Diff rev1/date1 against rev2.
      --ifdef=arg  Output diffs in ifdef format.
```

(consult the documentation for your diff program for rcsdiff-options. The most popular is -c for context diffs but there are many more). (Specify the --help global option for a list of other help options)

8.4.4 Projekt aufsetzen

Am Beispiel eines kleinen Projekts möchte ich die einzelnen Schritte erläutern, die zum Einrichten und Betrieb eines CVS-Projekts nötig sind. Bei diesem Projekt handelt es sich um ein „Schiffe-versenken“-Spiel, das den Projekt-Namen „Armada“ trägt und anfangs aus einem `src`-Verzeichnis mit den Klassen „Board.java“ (für das Spielfeld) und „Player.java“ in einem eigenen „armada“-Verzeichnis und aus einem Projekt-Verzeichnis („prj“) und einer JBuilder-Projekt-Datei („Armada.jpx“) besteht (s. Abb. 8.3).

Sobald die Projekt-Struktur einigermaßen feststeht, kann sie zusammen mit den Dateien über CVS verwaltet werden. Auch wenn es graphische Frontends zu CVS gibt, werde ich für die Einführung nur die CVS-Kommandos erläutern, weil

- ❑ bei der Verwendung der graphischen Oberflächen die Gefahr groß ist, dass man einfach die verschiedenen Menüpunkte ausprobiert ohne zu wissen, was man damit u. U. anrichtet,
- ❑ graphische Frontends nur einen Teil des Funktions-Spektrums von CVS abdecken,
- ❑ die Rückmeldung einer Benutzer-Aktion teilweise verbesserungsbedürftig ist,
- ❑ es bei auftretenden Problemen schwieriger ist, die Ursache herauszufinden, da nicht immer klar ist, welche Kommandos hinter der Oberfläche abgesetzt wurden.

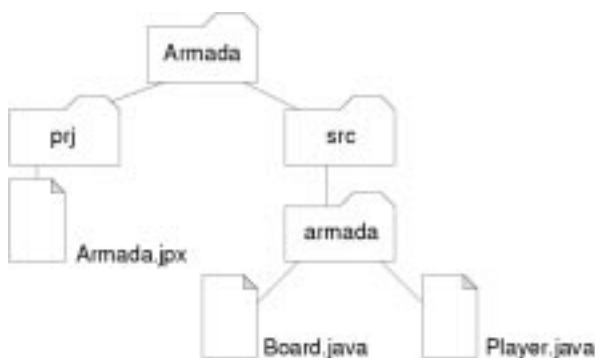


Abbildung 8.3: Verzeichnis-Struktur des Armada-Projekts

Auch wenn die Verlockung groß ist, gleich mit einer graphischen Oberfläche zu beginnen, ist es für das Verständnis hilfreicher, den Einstieg in CVS mit der Kommandozeile zu beginnen. Aber keine Angst: in Kap. 8.6 auf Seite 194 werden wir uns noch einige graphische Frontends anschauen, die Sie dann einsetzen dürfen, wenn Sie CVS verstanden haben.

8.4.4.1 Das Repository

Anfangs, wenn man noch der einzige Benutzer ist, kann man das Projekt unter seinem Heimatverzeichnis, z. B. unter „~/projects/Armada“, aufsetzen. Über die Umgebungs-Variable `CVSR00T` kann man dieses Verzeichnis für CVS voreinstellen, um es nicht jedesmal als Parameter in der Kommandozeile angeben zu müssen.

```
user@linux: > export CVSR00T=$HOME/projects/Armada ↵
```

Es empfiehlt sich, diese Variable dauerhaft in seine `.profile`-Datei aufzunehmen, da man sie in Zukunft öfters benötigen wird. Alternativ kann man zwar über die Option `-d` das Verzeichnis bei den meisten CVS-Kommandos auch angeben, aber auf die Dauer wird das lästig.

Hat man die Umgebung aufgebaut und das `$CVSR00T`-Verzeichnis angelegt, wird das *Repository* für unser Projekt mit

```
user@linux: > cvs init ↵
```

angelegt und initialisiert. Im Repository verwaltet CVS die einzelnen Versionen, den Aufbau des Projekt-Dateibaums sowie seine ganze Verwaltungsinformation unterhalb des `$CVSR00T`-Verzeichnisses. Das Repository ist die zentrale Stelle, aus der bestimmte Versionen angefordert oder Änderungen zurückgegeben werden können.

Da CVS auf RCS basiert, werden auch hier bei den einzelnen Versionen von Text-Dateien (in CVS als *Revision* bezeichnet) nur die Differenzen (Deltas) zur Vorgängerversion abgespeichert, wodurch vor allem Textdateien sehr komprimiert archiviert werden. CVS verwendet auch dieselben Schlüsselwörter wie RCS, d. h., es können sämtliche Schlüsselwörter aus Kapitel 8.3.3 auf Seite 159 eingesetzt werden.

8.4.4.2 Import

Der einfachste Weg, um ein Projekt für CVS aufzusetzen, führt über das `import`-Kommando. Sobald die Struktur und die ersten Dateien feststehen und einen Stand darstellen, auf dem man aufbauen kann, begibt man sich in das Projekt-Verzeichnis und löscht alle Dateien, die nicht in CVS landen sollen (z. B.

Backup-Dateien und übersetzte Java-Klassen), und startet den Import. Das `import`-Kommando hat dabei folgendes Format:

```
cvcs import [ -m "Kommentar" ] dir Vendor Release
```

dir ist das Verzeichnis, unter dem das Projekt im Repository abgelegt wird und das später für den Checkout (s. Kap. 8.4.5 auf der nächsten Seite) benötigt wird. Als *Vendor* nimmt man üblicherweise den Firmennamen (oder den eigenen Namen) und „start“ oder „first_release“ als *Release*, aber Sie können auch anderen Werte, die Sie für sinnvoll halten, hier hinterlegen.

Wenn Sie die Option „-m“ weglassen, geht für jede Datei ein Editor auf, in dem Sie dann den Kommentar eingeben dürfen. Einfacher geht es, wenn Sie den Kommentar über die Option „-m“ angeben.

```
user@linux: > cd Armada ↩
user@linux: > rm -r *.bak classes ↩
user@linux: > cvcs import -m "imported sources" Armada Oli start ↩
cvcs import: Importing /home/oliver/projects/Armada/Armada/prj
N Armada/prj/Armada.jpj
cvcs import: Importing /home/oliver/projects/Armada/Armada/src
cvcs import: Importing /home/oliver/projects/Armada/Armada/src/armada
N Armada/src/armada/Board.java
N Armada/src/armada/Player.java

No conflicts created by this import
```

Statt des Löschens unerwünschter Dateien und Verzeichnisse können diese auch über die Option „-I“ (für „Ignore“), die mehrfach angegeben werden kann, vom Import ausgeschlossen werden.

„cvcs import“ verlangt als zusätzlichen Parameter noch einen „Vendortag“ und einen „Releasetag“, den wir der Einfachheit halber mit dem eigenen Namen („Oli“) und „start“ vorbelegen. Empfehlenswert ist es auch, einen Kommentar über die Option „-m“ anzugeben – sonst wird man beim Import für jede Datei nach einem Kommentar gefragt, was bei Dutzenden von Dateien dann doch etwas lästig ist.

Mit der Option „-n“ kann ein Simulationslauf gestartet werden. Hierbei sieht man, was CVS machen würde, ohne dass es ausgeführt wird. Vor allem mit der Ignore-Option („-I“) kann dies ganz hilfreich sein, besonders beim ersten Mal.

Mit diesem Import haben wir jetzt in CVS ein *Repository* angelegt, in dem unser Projekt mitsamt seiner noch jungen Projekt-Geschichte abgelegt wird.

8.4.5 Der erste Checkout

Um festzustellen, ob der Import erfolgreich war, werden die importierten Dateien in einem beliebigen Verzeichnis (z. B. ~/work) über „checkout“ aus dem Repository geholt:

```
user@linux: > cvs checkout -P Armada
cvs checkout: Updating Armada
cvs checkout: Updating Armada/prj
U Armada/prj/Armada.jpx
cvs checkout: Updating Armada/src
cvs checkout: Updating Armada/src/armada
U Armada/src/armada/Board.java
U Armada/src/armada/Player.java
```

Damit wird eine *Arbeitskopie* des Projekts im aktuellen Verzeichnis unter „Armada“ angelegt (s. Abb. 8.4), in dem sich die verwalteten Quellen mit der Revision 1.1.1.1 wiederfinden (die Revision kann über „cvs log“ abgefragt werden).

Wenn Sie sich die ausgecheckten Verzeichnisse näher anschauen, werden Sie feststellen, dass sich in jedem Verzeichnis noch ein CVS-Verzeichnis befindet. Alles, was Sie lokal über cvs-Kommandos verändern, wird hier vermerkt. Diese Verwaltungsinformation wird für einen späteren Abgleich mit dem Repository benötigt.

Die Option „-P“ ist nicht unbedingt notwendig, zumindest noch nicht in der Anfangsphase eines Projekts. Sie verhindert, dass leere Verzeichnisse aus dem Repository übernommen werden. Leere Verzeichnisse entstehen, wenn ein Ver-

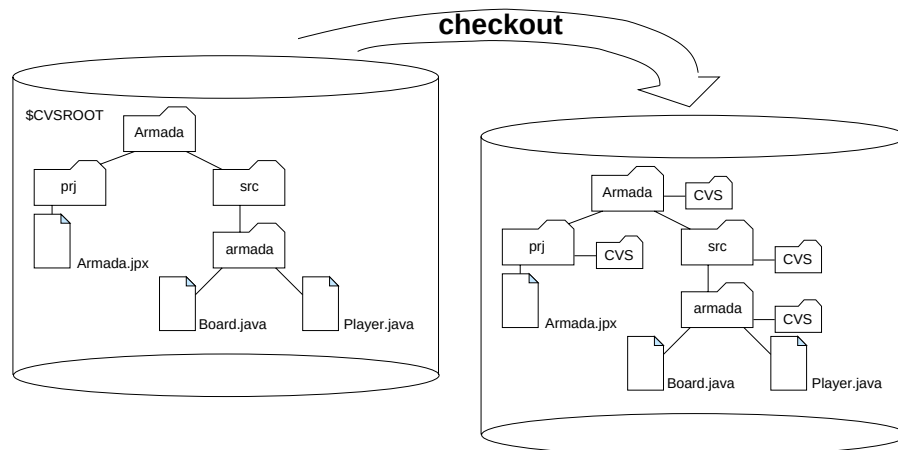


Abbildung 8.4: checkout

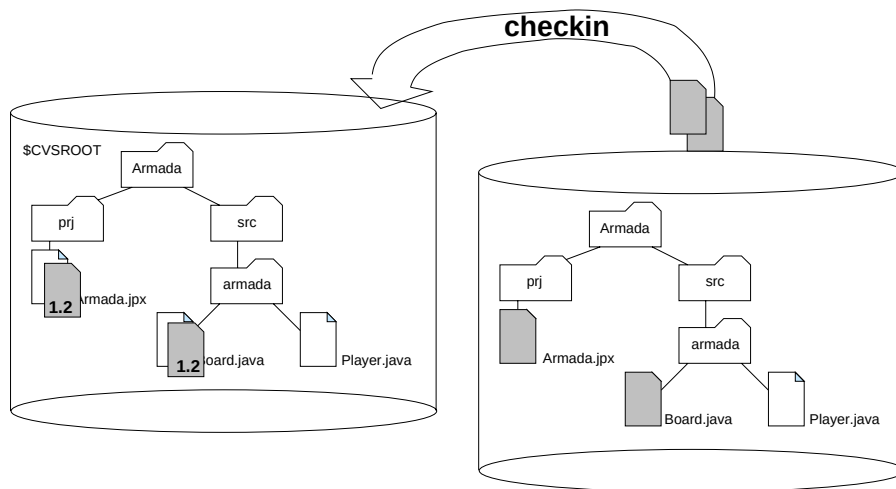


Abbildung 8.5: checkin

zeichnis aus CVS gelöscht wird – das Verzeichnis wird nämlich nicht wirklich gelöscht, sondern nur sein Inhalt.⁹

8.4.6 Der erste Checkin

Nachdem wir eine Kopie durch das Auschecken erhalten haben, machen wir uns an die Arbeit und codieren den Konstruktor der Board-Klasse. Diese Änderungen haben nur lokale Auswirkungen. Um diese Änderung in das Repository zurückzuschreiben, existiert das `commit`-Kommando (s. Abb. 8.5):

```
user@linux: > cvs commit -m "Constructor added"
cvs commit -m "Constructor added" .
cvs commit: Examining .
cvs commit: Examining prj
cvs commit: Examining src
cvs commit: Examining src/armada
Checking in prj/Armada.jpx;
/home/oliver/projects/Armada/Armada/prj/Armada.jpx,v <-- Armada.jpx
new revision: 1.2; previous revision: 1.1
done
Checking in src/armada/Board.java;
/home/oliver/projects/Armada/Armada/src/armada/Board.java,v <-- Board.java
new revision: 1.2; previous revision: 1.1
done
```

⁹Im Repository ist der Inhalt dieses Verzeichnisses weiterhin vorhanden für den Fall, dass eine ältere Version ausgecheckt werden soll.

Auch hier ist der Parameter „-m“ sehr hilfreich, da man ohne ihn für jede geänderte Datei nach einem Kommentar gefragt wird. Bevor allerdings die Änderungen zurückgeschrieben werden, läuft noch ein *Update* des Arbeitsverzeichnisses ab, um mögliche Konflikte zu entdecken und beheben zu können. Da wir aber momentan der einzige Entwickler sind, ist damit noch nicht zu rechnen.

Das `commit`-Kommando arbeitet, wie die meisten `CVS`-Kommandos, rekursiv, d. h., es werden das aktuelle Verzeichnis (bzw. das Verzeichnis, das man als letzten Parameter angegeben hat) und sämtliche Unterverzeichnisse nach geänderten Dateien durchsucht.

8.4.7 Elemente hinzufügen

Als Nächstes fügen wir eine neue Klasse „BoardView“ im Verzeichnis `src/armada` hinzu. Um diese Datei `CVS` bekannt zu machen, rufen wir ein „`CVS add`“ in diesem Verzeichnis auf:

```
user@linux: > cd src/armada ↵
user@linux: > cvs add BoardView.java ↵
cvs add: scheduling file 'BoardView.java' for addition
cvs add: use 'cvs commit' to add this file permanently
```

Damit landet diese neue Datei aber noch nicht im `CVS`-Repository, sondern wird nur für das nächste `commit` vorgemerkt.

```
user@linux: > cvs commit -m "BoardView.java added" ↵
cvs commit: Examining .
RCS file: /home/oliver/projects/Armada/Armada/src/armada/Board...
done
Checking in BoardView.java;
/home/oliver/projects/Armada/Armada/src/armada/BoardView.java, ...
initial revision: 1.1
done
```

Erst jetzt wird die neu erstellte Datei ins `CVS`-Repository gestellt. Ähnliches gilt für Verzeichnisse. Angenommen, ich möchte noch ein Verzeichnis „`lib`“ mit der Datei „`junit.jar`“¹⁰ (oder irgendeine andere Bibliothek) aufnehmen, muss ich zuerst das Verzeichnis unter `CVS`-Verwaltung bringen, ehe ich die gewünschte Bibliothek hinzufügen kann:

```
user@linux: > cvs add lib ↵
Directory /home/oliver/projects/Armada/Armada/lib added to ...
```

Im Gegensatz zum Hinzufügen einer Datei wird ein Verzeichnis sofort im Repository angelegt, auch wenn man lokal auf der Arbeitskopie arbeitet. Die Dateien,

¹⁰Diese Jar-Datei beinhaltet das JUnit-Testframework, das noch in Kap. 19.3 auf Seite 495 näher behandelt wird.

die sich in diesem Verzeichnis befinden, landen aber nicht automatisch im Repository, sondern müssen gesondert hinzugefügt und anschließend „committed“ werden (das add-Kommando ist eines der wenigen Kommandos, das nicht rekursiv arbeitet).

8.4.8 Binäre Dateien

Beim Hinzufügen von binären Dateien ist Vorsicht geboten. CVS durchsucht beim Auschecken die angeforderten Dateien nach RCS-Schlüsselwörtern (CVS baut auf RCS auf) und ersetzt so z.B. „\$Id\$“ durch die aktuelle Version. Auch wenn sehr unwahrscheinlich ist, dass genau diese Byte-Kombination in der Binär-Datei vorhanden ist, so kann es doch nicht ausgeschlossen werden.¹¹ Daher sollten Binär-Dateien über die Option „-kb“ hinzugefügt werden, da dadurch die Ersetzung der Schlüsselwörter (keyword substitution) unterbunden wird.

```
user@linux: > cvs add -kb lib/junit.jar ↵
cvs add: scheduling file 'lib/junit.jar' for addition
cvs add: use 'cvs commit' to add this file permanently
```

8.4.9 Dateien löschen

Da wir nun einmal nicht mehr mit JBuilder, sondern mit XEmacs weiterarbeiten wollen, entschließen wir uns, das prj-Verzeichnis und die darin enthaltene Armada.jpj-Datei zu löschen. Aus Sicherheitsgründen akzeptiert CVS den remove-Befehl erst, wenn die betroffenen Dateien auch tatsächlich gelöscht wurden:¹²

```
user@linux: > rm prj/Armada.jpj ↵
user@linux: > cvs remove prj ↵
cvs remove: Removing prj
cvs remove: scheduling 'prj/Armada.jpj' for removal
cvs remove: use 'cvs commit' to remove this file permanently
```

Das remove-Kommando arbeitet rekursiv, d.h., es reicht in unserem Fall aus, nur das prj-Verzeichnis zum Löschen zu markieren. Um es dauerhaft aus dem Repository zu entfernen, muss es noch „committed“ werden:

```
user@linux: > cvs commit -m "JBuilder no longer used" ↵
cvs commit: Examining .
cvs commit: Examining lib
```

¹¹Probieren Sie mal „ident /usr/bin/ident“ aus. Sie werden feststellen, dass einige \$Id\$-Kennungen in dieser Binärdatei vorhanden sind. Wenn Sie jetzt diese Datei ohne die Option „-kb“ in CVS ein- und auschecken würden, würde diese Kennung verändert und damit die Binärdatei höchstwahrscheinlich unbrauchbar werden.

¹²Passen Sie auf, dass Sie nur die Datei und nicht das Verzeichnis löschen. Die Verzeichnisse (genauer: das darin enthaltene CVS-Verzeichnis) werden von CVS noch benötigt.

```
cvcs commit: Examining prj
cvcs commit: Examining src
cvcs commit: Examining src/armada
Removing prj/Armada.jpx;
/home/oliver/projects/Armada/Armada/prj/Armada.jpx,v <-- Armada.jpx
new revision: delete; previous revision: 1.2
done
```

Wenn Sie das Projekt-Verzeichnis auflisten, werden Sie feststellen, dass das `prj`-Verzeichnis immer noch in Ihrer Arbeitskopie vorhanden ist. Falls es Sie stört, können Sie es löschen (`rm -r prj`).

Und wenn Sie das Projekt in einem anderen Verzeichnis auschecken, sehen Sie, dass das gelöschte `prj`-Verzeichnis als leeres Verzeichnis weitergeführt wird:

```
user@linux: > cvcs checkout Armada 
cvcs checkout: Updating Armada
cvcs checkout: Updating Armada/lib
U Armada/lib/junit.jar
cvcs checkout: Updating Armada/prj
cvcs checkout: Updating Armada/src
cvcs checkout: Updating Armada/src/armada
U Armada/src/armada/Board.java
U Armada/src/armada/BoardView.java
U Armada/src/armada/Player.java
```

Innerhalb des Repositories ist das Verzeichnis nach wie vor vorhanden, nur mit dem Unterschied, dass die Elemente dieses Verzeichnisses als gelöscht markiert sind. Um zu vermeiden, dass leere Verzeichnisse ausgecheckt werden, gibt man zusätzlich die Option „-P“ an.

```
user@linux: > cvcs checkout -P Armada 
...
```

Werden Dateien mit dem `remove`-Befehl gelöscht, werden diese Dateien nur aus der aktuellen Sicht gelöscht. Im Repository sind diese Dateien weiterhin vorhanden – das Repository verliert nichts.

8.4.10 Dateien umbenennen

Möchte man Dateien oder Verzeichnisse umbenennen oder in ein anderes Verzeichnis verschieben, ist dies mit CVS *nicht* möglich. Man kann sich nur dadurch behelfen, dass man die Datei unter dem gewünschten Namen bzw. im gewünschten Verzeichnis neu anlegt und die alte Datei dann löscht. Allerdings geht damit auch die Geschichte verloren und die Revisions-Zählung beginnt wieder bei 1.1.

Daher sollte im Kommentar ein Verweis auf die alte Datei stehen, damit man sich die History zumindest händisch wieder rekonstruieren kann.

8.4.11 CVS im Team

Während Oliver bis jetzt der einzige Entwickler im Armada-Projekt war, bekommt er nun einen neuen Mitarbeiter, Stan L., zugewiesen, der ihn bei der Entwicklung unterstützen soll. Damit sich Stan die Sourcen auschecken kann, muss er auf das Repository zugreifen können, und zwar nicht nur lesend, sondern auch schreibend. Am besten, man verwaltet die Benutzer in derselben Gruppe und schaut, dass die Gruppe im Repository Schreibberechtigung hat (notfalls über „`chmod -R g+w $CVSROOT`“).

Sollten Stan und Oliver in unterschiedlichen Gruppen zuhause sein, kann man auch eine gemeinsame Gruppe in `/etc/group` eintragen:

```
armada:x:102:oliver,stan
```

Hier wurde eine gemeinsame Gruppe „armada“ für die Benutzer „oliver“ und „stan“ angelegt. Um unter der neu eingerichteten Gruppen arbeiten zu können, muss man mit dem „`newgrp`“-Kommando die Gruppe wechseln:

```
user@linux: > newgrp armada ↵
user@linux: > id -nG ↵
armada tools
```

Mit „`id -nG`“ kann man feststellen, unter welcher Gruppe man gerade arbeitet. Sie wird als erste aufgelistet (s. o.).

Als Alternative zum obigen Verfahren kann auch das Gruppen-SetGID-Bit für das `$CVSROOT`- und allen drunterliegenden Verzeichnissen gesetzt werden (als „root“).

```
linux: # find $CVSROOT -type d -exec chmod g+ws {} ↵
```

Ist das SetGID-Bit eines Verzeichnisses gesetzt, erhalten neu angelegte Dateien automatisch die gleiche Gruppe wie das Verzeichnis. In unserem Beispiel bedeutet dies, das neue Dateien beim Zurückschreiben in das Repository die Gruppe „armada“ erhalten, ohne dass man explizit über `newgrp` in diese Gruppe wechseln muss.

8.4.11.1 Update

Nachdem sich Stan den aktuellen Stand ausgecheckt hat, beginnt er mit der Implementierung von `BoardView.java` (im Verzeichnis `armada/src`). Ist er mit der Methode „`show()`“ fertig, gibt er die Änderungen an das Repository zurück („ `cvs commit...`“) und benachrichtigt uns von der Änderung.

Tabelle 8.2: „update“-Kennungen

Zeichen	steht für	Bedeutung
U	Update	Die Datei wird auf den neuesten Stand gebracht.
A	Added	Die Datei wurde hinzugefügt.
R	Removed	Die Datei wurde gelöscht.
M	Modified	Die Datei wurde modifiziert.
C	Conflict	Beim Zusammenfügen der Datei aus dem Repository mit der lokal veränderten Kopie trat ein Konflikt auf, der nicht automatisch behoben werden konnte.
?		Diese Datei gibt es nur im Arbeitsverzeichnis, nicht aber im Repository.

Wie kommt Oliver an die neue Version von `BoardView.java`? Alles, was er dafür machen muss, ist ein „cvs update“:

```
user@linux: > cvs update src/armada ↵
cvs update: Updating src/armada
U src/armada/BoardView.java
```

Beim Update vergleicht CVS für jede Datei im aktuellen und in den darunterliegenden Verzeichnissen, ob im Repository bereits eine neuere Datei vorliegt und zeigt das Ergebnis dieses Abgleichs in Form eines Buchstabens (s. Tab. 8.2) an.

Möchte man nur sehen, welche Dateien sich geändert haben, welche hinzugefügt oder gelöscht wurden, ist die Option „-n“ hilfreich. Hier wird der „update“ nicht wirklich ausgeführt, aber man sieht, was passieren würde, wenn man ihn tatsächlich (ohne die Option „-n“) ablaufen lassen würde.

8.4.11.2 Konflikte

Konflikte können entstehen, wenn zwei (oder mehrere) Entwickler dieselbe Datei bearbeiten und zurückschreiben wollen. Da Stan und Oliver bis jetzt an verschiedenen Dateien arbeiten, gibt es in unserem Vorzeigeprojekt noch keine Konflikte. In der Zwischenzeit hat aber Stan an `BoardView.java` weitergearbeitet und als Version 1.3 „committed“, während Oliver noch über `Board.java` Version 1.2 brütet und feststellt, dass ihm in Stans `BoardView.java` noch eine `main()`-Methode zum Testen fehlt. Kurzerhand fügt er sie in seinen lokalen Stand ein.

Nachdem Oliver mit seinen Änderungen fertig ist, möchte er seinen Stand in das Repository zurückschreiben:

8.4.11.3 cvs release

Nach kurzer Zeit scheidet Stan aus dem Projekt wieder aus. Seine Arbeitskopie kann man einfach von Hand löschen (`rm -r Armada`) oder über das `release`-Kommando:

```
stan@linux: > cvs release -d Armada ↵  
? classes  
U src/armada/BoardView.java  
You have [0] altered files in this repository.  
Are you sure you want to release directory 'Armada': y ↵
```

„`cvs release`“ löscht die Arbeits-Kopie ebenfalls, prüft aber vorher ab, ob es noch Änderungen gibt, die nicht ins Repository zurückgegeben wurden. Ohne die Option „-d“ wird das Verzeichnis nicht gelöscht, sondern der „Release“ nur lokal vermerkt.

Vorteil von „`cvs release`“ gegenüber dem manuellen Löschen ist neben der Überprüfung auf offene Checkouts auch die Protokollierung im Repository. Damit ist später noch nachvollziehbar, wann der Ausstieg eines Entwicklers stattgefunden hat.

8.4.12 Etikettierung

Irgendwann hat man in der Entwicklung einen Stand erreicht, den man gern einfrieren möchte, um evtl. später auf ihn zugreifen zu können. Meist tritt dieser Wunsch nach folgenden Ereignissen ein:

- ❑ ein wichtiger Meilenstein wurde erreicht



Abbildung 8.6: Konservierung eines Versionstandes

- ❑ der aktuelle Stand wird zur neuen (Alpha-, Beta-, Final-)Version erhoben
- ❑ ein wichtiges Ereignis (erster fehlerfreier Lauf, Weitergabe an Testabteilung,...) ist eingetreten
- ❑ die Version wurde zum Testen frei- bzw. weitergegeben
- ❑ man möchte kennzeichnen, welche Versionen der verschiedene Dateien zusammenpassen bzw. zusammengehören
- ❑ sonstige Gründe

Das „Einfrieren“ erfolgt in CVS mit Hilfe sogenannter *Tags* (zu deutsch: Marke, Etikett). Diese werden an die aktuelle Version jeder Datei geheftet und erlauben später das gezielte Ansprechen der so markierten Datei-Versionen, ohne die laufende Weiterentwicklung zu stören (s. Abb. 8.6 auf der vorherigen Seite).

```
user@linux: > cvs tag stans_legacy ↩
cvs tag: Tagging .
cvs tag: Tagging lib
T lib/junit.jar
cvs tag: Tagging src
cvs tag: Tagging src/armada
T src/armada/Board.java
T src/armada/BoardView.java
T src/armada/Player.java
```

Damit erhalten sämtliche Dateien im aktuellen und in darunterliegenden Verzeichnissen das Etikett „stans_legacy“ angehängt (s. Abb. 8.7). Nach einem „commit“ werden diese Markierungen auch im CVS-Repository an den entsprechenden Versionen angeheftet. Mit dem `rtag`-Kommando kann man die Markierungen auch direkt im Repository anbringen.

Will man später auf eine bestimmte Version zurückgreifen, kann dies anhand des Datums (Option „-D“) oder anhand der Markierung über die Option „-r stans_legacy“ beim checkout erfolgen:

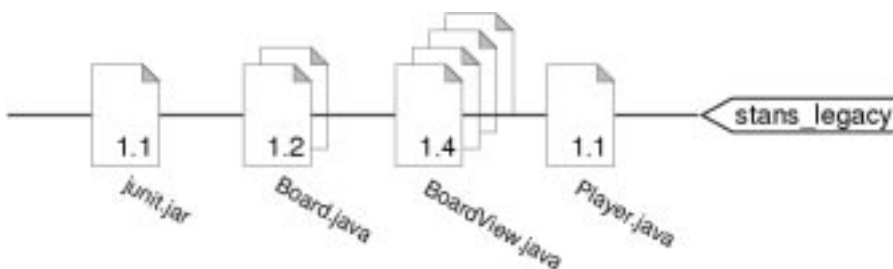


Abbildung 8.7: Markierung einer Revision

```
user@linux: > cvs checkout -r stans_legacy Armada ↩  
...
```

Sollte man im Verlauf der Entwicklung feststellen, dass man den Überblick über die Markierungen verloren hat, kann man überflüssige Tags nachträglich wieder entfernen (über die Option „-d“ beim tag-Kommando).

8.4.13 Verzweigungen (branches)

Die Entwicklung schreitet weiter voran, da wird ein Fehler in genau der Version gemeldet, die nach dem Abgang von Stan zum Ausprobieren freigegeben wurde. Da wir diese Version damals mit dem Tag „stans_legacy“ markiert hatten, können wir sie wieder auschecken, und es gelingt uns auch, den Fehler zu reproduzieren. Weil wir aber auf einer älteren Version arbeiten, ist es ratsam, an dieser Stelle zu verzweigen und einen „Branch“ (Option „-b“) aufzumachen (die Option „-d“ legt das ausgecheckte Projekt im angegebenen Verzeichnis „armada_patch“ an):

```
oliver@linux: > cvs checkout -r stans_legacy Armada -d armada_patch ↩  
...  
oliver@linux: > cd Armada ↩  
oliver@linux: > cvs tag -b stans_bugfix_branch ↩  
...
```

Bis jetzt ist dieser Abzweig nur in der lokalen Arbeitskopie vermerkt. Will man den Branch bereits im Repository verankern, ohne sich eine Arbeitskopie auszuchecken, kann dazu das rtag-Kommando verwendet werden.

```
oliver@linux: > cvs rtag -b -r stans_legacy stans_bugfix_branch Armada ↩
```

Der Branch wird mit Hilfe des angegebenen Tags („stans_bugfix_branch“) gekennzeichnet. Um ihn von einem normalen Tag unterscheiden zu können, empfiehlt es sich, dies durch eine entsprechende Namensgebung (z. B. ein angehängtes „_branch“) hervorzuheben.

Mit diesem Branch können wir unabhängig von der aktuellen Hauptlinie („Main Branch“) agieren und den Fehler beheben.

8.4.13.1 Sticky Tags

Die Fehlersuche auf dem Bugfix-Zweig ist abgeschlossen und die geänderten Board- und BoardView-Klassen sollen wieder ins CVS zurückgegeben werden:

```
oliver@linux: > cvs -n commit src/armada ↩  
cvs commit: Examining src/armada  
cvs commit: sticky tag 'stans_legacy' for file 'src/armada/...
```

```
branch
cvs commit: sticky tag 'stans_legacy' for file 'src/armada/...'
cvs [commit aborted]: correct above errors first!
```

Hier wird versucht, die Dateien als neue Version des Haupt-Asts einzuchecken. Da aber zum Ändern eine ältere Version ausgecheckt wurde, sind die Dateien von CVS als „sticky“ (verunreinigt, klebrig) gekennzeichnet. Um CVS mitzuteilen, dass diese Dateien zum „stans_bugfix_branch“-Zweig gehören, muss die Option „-r“ gesetzt werden:

```
user@linux: > cvs commit -m "show() changed" -r \ ↩
> stans_bugfix_branch src/armada ↩
cvs commit: Examining src/armada
Checking in Board.java;
/home/oliver/projects/Armada/Armada/src/armada/Board.java,v...
new revision: 1.2.2.1; previous revision: 1.2
done
Checking in BoardView.java;
/home/oliver/projects/Armada/Armada/src/armada/BoardView.java,...
new revision: 1.4.2.1; previous revision: 1.4
done
```

Der neue Zweig von BoardView.java, der von der Version 1.4 abzweigt, wird unter 1.4.2 weitergeführt, wobei die letzte Stelle hochgezählt wird (1.4.2.1, 1.4.2.2, ...). Aus unerfindlichen Gründen verwendet CVS für Seitenzweige immer gerade Zahlen, beginnend mit 2.¹³

Die nächsten Änderungen an Board.java und BoardView.java finden auf diesem Bugfix-Branch statt. Daher muss beim nächsten „commit“ der Branch nicht mehr explizit angegeben werden:

```
user@linux: > cvs commit src/armada ↩
cvs commit: Examining src/armada
Checking in src/armada/BoardView.java;
/home/oliver/projects/Armada/Armada/src/armada/BoardView.java,...
new revision: 1.4.2.2; previous revision: 1.4.2.1
done
```

Hier wurde jetzt nur BoardView.java geändert. Parallel dazu bleibt die Entwicklung auf dem Haupt-Zweig (main branch) auch nicht stehen, so dass BoardView.java dort inzwischen in der Revision 1.6 vorliegt (s. Abb. 8.8 auf der nächsten Seite).

¹³Wenn Sie von diesem Zweig einen weiteren Zweig anlegen, würde dieser unter der Nummer 1.4.2.1.2 geführt.

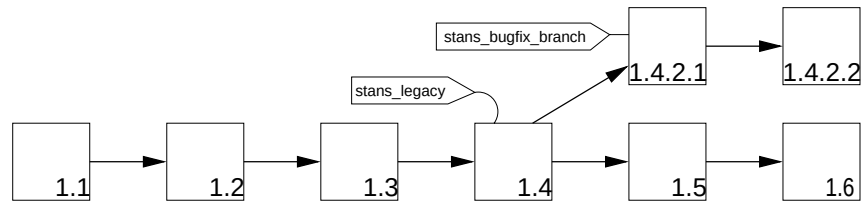


Abbildung 8.8: Branch (BoardView.java)

8.4.13.2 Die Wiedervereinigung (merge)

Auf dem Nebenzweig ist die Entwicklung abgeschlossen und man möchte die Änderungen und Fehlerbereinigungen, die in diese Linie eingeflossen sind, auch im Hauptzweig einbringen. Dazu ist ein „Merge“ notwendig. Der Merge selbst besteht aus 3 Schritten:

- ❑ Update mit dem (Bugfix-)Branch: `update -j` („join“)
- ❑ Konflikt lösen
- ❑ Commit

Die Vereinigung der beiden Zweige findet auf der aktuellen Version des Zweiges statt, in den diese Änderung einfließen soll; in unserem Fall ist dies der Hauptzweig, auf dem Oliver arbeitet. Schauen wir Oliver über die Schulter, wie er die beiden Entwicklungslinien zusammenführt:

```

oliver@linux: > cvs update -j stans_bugfix_branch
cvs update: Updating .
...
cvs update: Updating src/armada
RCS file: /home/oliver/projects/Armada/Armada/src/armada/Board.java,v
retrieving revision 1.2
retrieving revision 1.2.2.1
Merging differences between 1.2 and 1.2.2.1 into Board.java
rcsmerge: warning: conflicts during merge
RCS file: /home/oliver/projects/Armada/Armada/src/armada/BoardView.java,v
retrieving revision 1.4
retrieving revision 1.4.2.2
Merging differences between 1.4 and 1.4.2.2 into BoardView.java
rcsmerge: warning: conflicts during merge
  
```

Mit der Option „-j“ (join) wird angezeigt, dass der angegebene Zweig mit dem aktuellen Stand zusammengeführt wird. Und wie bei einem „normalen“ Update kann es auch hier zu Konflikten kommen bzw. die Konflikte werden bei einem „join“ die Regel sein. Im nächsten Schritt gilt es jetzt, diese Konflikte zu lösen, ehe die Zusammenführung mit einem „commit“ abgeschlossen werden kann.

```

oliver@linux: > cvs commit -m "merge with stans_bugfix_branch"
cvs commit: Examining .
cvs commit: Examining lib
cvs commit: Examining src
cvs commit: Examining src/armada
Checking in src/armada/Board.java;
/home/oliver/projects/Armada/Armada/src/armada/Board.java,v...
new revision: 1.5; previous revision: 1.4
done
Checking in src/armada/BoardView.java;
/home/oliver/projects/Armada/Armada/src/armada/BoardView.java,v...
new revision: 1.7; previous revision: 1.6
done

```

Leider merkt sich CVS nicht, dass die neue Version durch einen „Merge“ entstand, sondern speichert nur die Änderungen. Wenn jetzt Stan auf dem Bugfix-Zweig weiterarbeitet und Oliver später einen Merge davon machen will, würden beim Update die Änderungen, die eigentlich schon eingearbeitet wurden, noch einmal übertragen. Man kann aber beim Update-Kommando die Join-Option („-j“) zweimal angeben, um nur die Änderungen zwischen zwei Versionsständen zu bekommen. Dazu sollte die Version, die Ausgangspunkt für den „Merge“ war, mit einem Label versehen werden, z. B. „stans_bugfix_merge_1“ (s. Abb. 8.9):

```

user@linux: > cvs tag stans_bugfix_merge_1
user@linux: > cvs commit -m "first merge with main branch"

```

Passen Sie auf, dass Sie den Label im Bugfix-Zweig setzen. Alternativ können Sie den Label auch über das rtag-Kommando setzen:

```

user@linux: > cvs rtag -r stans_bugfix_branch \
> stans_bugfix_merge_1 Armada

```

Später können dann mittels

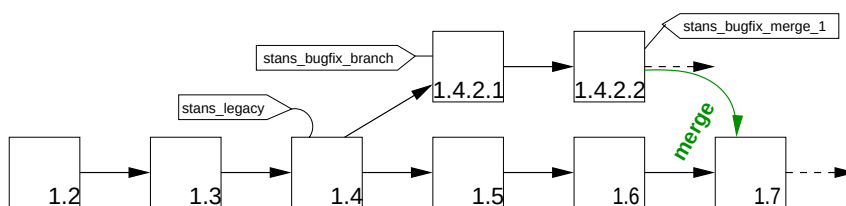


Abbildung 8.9: Merge (BoardView.java)

```
user@linux: > cvs update -j stans_bugfix_merge_1 -j \ ↵  
> stans_bugfix_branch ↵
```

neue Änderungen im Bugfix-Zweig in die aktuelle Version integriert werden.

8.4.14 Einige Tipps

8.4.14.1 Stabile Versionen

Es empfiehlt sich, nur stabile Stände in CVS abzuspeichern und CVS nicht zur Datensicherung zu missbrauchen. Für die Suche nach dem letzten stabilen Stand vergeht sonst u. U. viel unnötige Zeit.

8.4.14.2 Regelmäßige Updates

Bei der Entwicklung im Team empfiehlt es sich, regelmäßig einen Update anzustoßen, um Änderungen möglichst früh mitzubekommen. Andernfalls setzt man sich der Gefahr aus, dass man nach „cvs commit“ von zu vielen Konflikten überrascht wird. Nach einem

```
cvs [ -n ] update [ -d ]
```

werden die geänderten Dateien angezeigt und eingespielt (wenn es keine Konflikte gegeben hat). Wurden neue Verzeichnisse angelegt, so sorgt die Option „-d“ dafür, dass beim update die neu hinzugekommenen Verzeichnisse mit samt den darin enthaltenen Dateien ebenfalls angelegt werden. Möchte man nur mitbekommen, welche Dateien verändert wurden, ohne dass der Update tatsächlich durchgeführt wird, ist die Option „-n“ hilfreich,

8.4.14.3 Lokale Änderungen verwerfen

Will man zum letzten im CVS abgespeicherten Stand zurückkehren, kann man die betroffene Datei einfach löschen und anschließend ein

```
user@linux: > cvs update ↵
```

im betroffenen Verzeichnis aufrufen. Eventuell gibt man noch den Namen der Datei an – dann geht der Update einen Tick schneller. So wird die gelöschte Datei wieder aus dem CVS-Repository geholt.

Voraussetzung für diese Vorgehensweise ist, dass sich im Repository keine unbrauchbare Version der gewünschten Datei befindet. Ansonsten muss man erst die letzte brauchbare Version suchen.

8.5 CVS im Netz

CVS eignet sich nicht nur für die lokale Versionierung, sondern kann seine Stärken vor allem bei verteilter Entwicklung ausspielen. Einer der größten Vorteile ist dabei die Unabhängigkeit vom CVS-Server. Lediglich zum Abgleich seiner Änderungen oder zum Abholen geänderter Dateien braucht man eine Netzverbindung, wobei dies durchaus eine Modem- oder ISDN-Verbindung sein kann. Nicht umsonst ist CVS erste Wahl im OpenSource-Bereich und auch bei vielen anderen Projekten, die über verteilte Standorte oder übers Internet koordiniert werden.

Aus Sicht des Entwicklers spielt es (fast) keine Rolle, ob das Repository lokal auf seinem Rechner, auf einem Server im Netzwerk, im Intranet oder gar Internet liegt. Er muss lediglich CVSR00T anpassen bzw. das neue Repository über die Option „-d“ ansprechen. Und wenn er ein Projekt bereits ausgecheckt hat, braucht er nicht einmal das, da diese Information in der Datei CVS/Root in jedem ausgecheckten Verzeichnis abgelegt und von CVS verwendet wird.

8.5.1 Lokale Netze

Die einfachste Möglichkeit, CVS im Netz zu betreiben, ist der Zugriff über „rsh“. Dazu muss auf der Serverseite im Heimatverzeichnis des Repository-Besitzers die Datei „.rhosts“ angelegt werden. In diese Datei werden diejenigen Rechner und Benutzer eingetragen, die auf das Repository Zugriff erhalten sollen. Getestet werden kann das erfolgreiche Aufsetzen der .rhosts-Datei, indem man als fraglicher Benutzer ein rsh startet:

```
user@linux: > rsh -l cvsroot hedwig date ↵
```

Damit wird auf dem Rechner „hedwig“ unter dem Benutzer „cvsroot“ das date-Kommando abgesetzt (man kann natürlich auch ein anderes Kommando zum Testen auswählen). Erhält man dabei statt des aktuellen Datums lediglich ein „Permission denied.“, hat man irgendetwas falsch gemacht. Wichtig bei der .rhosts-Datei ist, dass sie nur für den Eigentümer (im obigen Beispiel ist dies „cvsroot“) Lese- und Schreibberechtigung hat (chmod 600 .rhosts).

Danach passt man CVSR00T an das neue Repository an:

```
user@linux: > export CVSR00T=:ext:cvsroot@hedwig:/home/cvsroot ↵
```

Jetzt kann man auf dieses Repository, das auf dem Rechner „hedwig“ liegt und vom Benutzer „cvsroot“ verwaltet wird, genauso wie auf ein lokales Repository zugreifen. Die Methode ext gibt an, dass das externe rsh-Kommando verwendet wird. Über die Umgebungs-Variable „CVS_RSH“ kann man auch ein anderes Kommando für den Zugriff einstellen.

Index

Symbole

-classic 31
 -nolading 504
 .rhosts 189
 \$RPM_OPT_FLAGS . 735
 %changelog 737
 %config 736
 %defattr 736
 %dir 736
 %doc 736
 %files 736
 %post 736
 %postun 736
 %pre 736
 %preun 736
 Überschrift 627
 Übersichtsseite .. 625, 628
 Übertragung 191
 _REENTRANT 584
 C++ 581
 3D 48

A

Abhängigkeiten . 42, 216, 303
 zirkulär 319
 Abkürzungen 645
 abstract 566
 AbstractList 567
 Abstrakte Klassen ... 566
 Abstraktion 515, 670
 Academic Licence ... 245

AccessControlException . 124
 Account-Client 447
 Account.java 443
 AccountBean 443
 AccountBean.java ... 445, 450
 AccountHome.java .. 444
 AccountPK.java 443
 actions 303
 Activation Bar 217
 activity diagram 226
 Administration 14
 afterBegin() 458
 afterCompletion() ... 458
 Aggregation 213
 Akronyme 645
 Aktionen 222, 303
 Aktivitäten
 parallel 226
 Akzeptanz-Test 494
 Algorithmus
 DSA 135
 RSA 135
 Alias 134, 135
 all-permissions 750
 AllocObject() 592
 Alpha-Prozessor 30
 AlphaWorks 43
 Altlasten 579
 Amaya 56, 77
 Amigos 205
 Analyse 204

Abhängigkeits-Analyse 99

Anforderungen

Hardware 25
 Software 26
 Ant 270, 323, 324
 Attribute 327
 Classpath 333
 Fileset 333
 Filter 349
 javadoc 339
 Kommandozeile ... 349
 Optionen 350
 Path 333
 Pattern 334
 Patternset 333
 Plugin 326
 Projekt 327
 Properties 329
 Build-in 331
 spezielle 332
 System 332
 Referenz 333
 Richtlinien 353
 Target 327
 Task 336
 Antidote 326
 Anweisung 386
 Anwendungs-Entwickler 434
 Anwendungsfälle ... 209
 Anzeige 102
 Apache 704

- JServ 401
- Software Licence .. 712
- Webserver 745
- Apache-Modell 703
- API 831
- AppBrowser 280
- Appender 479
- Applet 24, 100
 - signiert 122
- applet-desc-Element . 751
- applet-Tag 398
- Applets 119, 270, 397,
749, 751, 793
- appletviewer 100, 398
- application 418
- Application
 - Programming
 - Interface 831
- Application Server .. 460,
831
- IAS 276
- application-desc 744, 751
- Application-Manager 748
- Arbeitskopie 174
- Architektur 205
 - 2-Tier 426
 - 3-Tier 427
 - n-Tier 427
- Archiv 108, 346
- Archivierer 107
- Archivierung 154
- ArgoUML 207
- Array-Operationen .. 600
- ArrayIndexOutOf-
BoundsException ...
600, 602
- ArrayStoreException 600
- Artistic Licence 711
- ASCII 116, 588
- ASP 395
- assert
 - assertEquals 498
 - assertNotNull 498
 - assertNull 498
 - assertSame 498
- Assert-Klasse 485
- Associations 211
- Attribut 378
- Attributwert 378
- Audio 48
- Aufräumarbeiten 735
- Auschecken 157, 158, 716
- Auskommentieren ... 654
- Auslieferung 755
- Ausprägung 378
- Authentifizierung 49
- author 627
- Author 159
- Authorisierung 49
- Auto-Deploy 463
- autocommit 390
- AUTOEXEC.BAT 722
- Automatisierung 506
- Autor 620
- Autorisierung 742
- Autorität 748
- AWT 8, 24, 294
- awtui.TestRunner ... 495
- B**
- Background Future
 - Evaluation 528
- Backslash 312
- Banner-Kommentare 650
- Basar-Methode 699
- BasicConfigurator ... 480
- Basis-Klassen 85
- Batch-Kommandos .. 390
- Batch-Methoden 528
- Bean Managed
 - Persistence . 432, 449
- Bean Shell 266
- Bean-Instanz 436
- Bean-Klasse 436, 440
- Beans 725
- beforeCompletion() .. 458
- begin 456
- Benchmark 31
- Benennungen 660
- Benutzer-Kennung .. 383
- Berechtigungs-Klasse 127
- Bereichsüberprüfung 121
- BerliOS 713
- Bertrand Meyer 482
- Beta-Tester 701
- Beziehungen ... 211, 378,
432
- Bibliothek 47
 - Enterprise Technologie
47
 - Optional Packages .. 47
 - Standard Erweiterung .
47
- Bibliotheken 607
- Bildbetrachtung 48
- Blackdown 26
- Block-Kommentare .. 653
- Blowfish 48
- BMP 432, 449, 831
- BOE 206
- Booch 205
- Bootclasspath . 85, 94, 99,
626
- Borland 275, 373
- bottom 627
- Bourne-Shell 315
- Branch 184
- Breakpoint . 102, 258, 283
- BSD
 - Lizenz 711
- BufferedInputStream 535,
550
- BufferedOutputStream ..
550
- Bug 494
- Bug-Fixe 758
- Bugzilla 763
- Build 302
 - Prozess 301
- BUILD 737
- Build-Abschnitt 735
- Build-Datei 327
- Build-Protokoll 325
- Build-Tool 323
- Build-Verzeichnis 735
- build.xml 326, 352
- BuildRoot 734
- Business Object
 - Engineering 206

- Business-Logik 419
 Business-Methode ... 437
 Byte-Code 97, 831
 Bytecode 139, 511
- C**
- C 581
 C-Implementierung . 584
 C-String 584
 C# 518, 779
 CA 133
 CACAO 30
 Cache 748
 Caching 525
 Caching-Attribut 433
 Caching-Mechanismus .. 440
 CaffeineMark 31
 embedded 33
 CallTypMethod 595
 CallTypMethodA() ... 595
 CallTypMethodV() ... 595
 CallableStatement ... 387, 390
 CallNonvirtualTypMethod-Funktionen 596
 CallNonvirtualTypMethodA-Funktionen 596
 CallNonvirtualTypMethodV-Funktionen 596
 CallStaticTypMethod-Funktionen 597
 Capability Maturity Model 154, 831
 catch 103
 Categories 478
 CERT 705
 Certificate Authority 133
 Certificate Signing Request 133
 Cervisia 198
 CGI 48, 395, 399, 831
 Chain of Trust 135
 Changelog 737
 charset 628
 checkin 157, 158, 163, 175
 checkout ... 157, 158, 164, 174, 191
 ci 157, 158, 163
 Class-Loader 85, 121, 504
 Class.newInstance() . 436
 ClassCastException .. 504
 ClassCircularityError 590
 ClassFormatError ... 590
 classic 101
 ClassNotFoundException 371
 Classpath 333
 Extensions Classpath .. 86
 User Classpath 86
 CLASSPATH .. 24, 41, 86, 92, 95, 107, 145, 477, 495, 583, 626
 ClassWriter 632
 clean 735
 Client 439, 672
 Client-Server-Architektur 384
 Client/Server 409
 Clipboard 751
 Clonable 565, 671
 close() 385
 Closed Source ... 704, 707
 CMM 154, 831
 CMP 432, 443, 832
 Felder 446
 CMS 154
 co 157, 158, 164
 COBOL 579
 Code
 Inspections 676
 Reading 676
 Code-Completion ... 270
 Code-Copy 649
 Code-Generierung ... 214
 codebase 744
 CodeBase 127
 CodeInsight 276
 Codierung 253
 Collection 515, 530
 COM 580
 commit 388, 456
 COMMIT 388
 Common Gateway Interface 395, 831
 Common Public Licence . 712
 Compiler 97, 269
 Component Interface 459
 Component Object Model 580
 Components 725
 CONFIG.SYS 722
 Configuration
 Management System 154
 configure 257
 connect 383
 Connection 387
 setAutoCommit ... 390
 Container 445
 Container Managed Persistence . 432, 443, 832
 content pane 280
 Contract
 Klasse 482
 Violation 482
 Controller 420
 Cookies 396, 752
 Copyleft 710
 Copyright 733
 CORBA 49, 139, 276, 363, 371, 425, 832
 Core Class Library 47
 Core Classes 85
 Core Library 37
 counter.policy 795
 CounterApplet .. 122, 793
 CPU-Verbrauch 546
 Cracker 698
 Crimson 74
 Cross-Compiler 99
 Crypt 548
 CSR 133
 Ctrl-D 116
 Cursor 384

- cvs
 - checkout 191
 - commit 188
 - diff 170
 - einloggen 191
 - import 172
 - init 172
 - pserver 190
 - release 182
- CVS ... 166, 270, 277, 297, 325
 - Branch 184
 - GUI 194
 - Kommandos 167
 - Konflikte 180
 - Merge 186
 - Netz 189
 - Optionen 169
 - Pserver-Protokoll .. 190
 - Repository 172
 - Revision 172
 - Schlüsselwörter ... 172
 - Sticky Tag 184
 - Umgebungsvariablen . 170
 - Update 179
- CVS_AUTH_PORT .. 190
- CVS_RSH 189
- CVSROOT 172, 189
- Cygwin 10
- D**
- Daemon 832
- Daily Build 302
- DAO 835
- Data Access Objects . 835
- Data Display Debugger .. 607
- Data-Object 835
- Data-Objekte 673
- Date 159
- Datei
 - /etc/inetd.conf 190
 - /etc/services 190
 - Konfigurations-Datei .. 481
 - Log-Datei 479
 - Policy 126
 - versteckte 311
- Datei-Endung 310
- Datenbank 51, 276
 - InterBase 277
 - Optimierung 390
 - relational 377, 380
 - schließen 385
- Datenbanken 373
- Datenbanktabelle 447
- Datenbanktreiber 380
- Datenbankzugriffe .. 389
- Datensätze 378
- Datensatz 378
- Datentyp
 - einfach 586
 - höhere 587
 - jobject 587
 - jsize 587
 - size 587
- Datentypen 385
- DbC 482
- dbx 258
- DCL 765
- ddd 607
 - ddd-dynamic 256
 - ddd-semistatic 256
 - ddd-static 257
- DDD 256
- Deadlock 91, 832
- Debian Bug TS 762
- Debug
 - Information 98, 608
- DEBUG 478
- Debuggen 607
- Debugger 43, 94, 100, 255, 269, 283, 474
 - integriert 276
 - Poor Mans Debugger .. 475
 - Remote 276
- Decompiler 139
- DefineClass() 589
- Deinstallation ... 722, 728
- Deinstallations-Skripte .. 736
- Deklaration 660
- DELETE 379
- DeleteGlobalRef() ... 591
- dependencies ... 216, 303
- Deployer 434
- Deployment 405, 433, 438, 446, 453, 673
 - Deskriptor 438
- deprecated 98, 628
- deprecated-list.html . 628
- DES 48
- description 734
- Design 204, 205
 - Pattern 840
- Design by Contract .. 482
- Design1Pattern 669
- Destinguished Name 135
- destroy() 123
- Deutsch 660
- Diagramm
 - Aktivitäts-Diagramm .. 226, 242
 - Klassen-Diagramm 241
 - Kollaborations-Diagramm 221
 - Komponenten-Diagramm 230
 - Paket-Diagramm .. 215
 - Sequenz-Diagramm ... 217
 - Statechart Diagram 222
 - Verteilungs-Diagramm 230
 - Zustands-Diagramm .. 222
- Diagrammtypen 206
- diff 165, 170
- Diff-Tool 194
- Diffie-Hellman 48
- Direktive
 - Include 414
 - Page 414
 - Taglib 414
- dirty code 667
- dirty reads 389

-
- Disassembler ... 113, 140, 141
 DISPLAY 96
 DLL 96
 DN 135
 DNS 48
 doc comments 650
 docletpath 630
 Doclets 485, 625, 627, 629, 652
 doctitle 627
 DOCTYPE 60
 Document Type
 Definition 69
 doencoding 629
 doGet() 400, 406
 Dokumentation . 617, 635
 Dokumenten-Typ 60
 DOM 74
 Domäne 378
 DOMConfigurator ... 480
 doPost() 408
 Double Choco Latte . 765
 download-Attribut .. 750
 DSA 135
 DTD 69
 Dummy-Targets 314
 dump 102
 Dynamic Loader 96
 Dynamisierung 395
- E**
- eager 750
 Eager Initialization .. 528
 echo 313
 echtzeitfähig 8
 Echtzeitfähigkeit 512
 Eclipse 288
 Editor 253, 269
 Effizienz 665
 Eiffel 482
 Ein-/ Ausgabe 533
 einchecken 157, 158
 Einhaltung 664
 Einpacken 737
 Einrückung 657
- EJB 50, 276, 325, 420, 460, 798, 832
 2.0 458
 Compiler 439
 Container 434
 Entwickler 434
 Query Language .. 459
 Richtlinien 672
 ejb-jar.xml 799, 803
 ejbActivate() 436, 440
 ejbCreate() 436
 ejbLoad() 445, 450
 EJBoss 461
 ejbPassivate() ... 436, 440
 ejbRemove() 436
 ejbStore() 445, 450
 Elisp 261
 Emacs .. 42, 259, 325, 351, 683, 701, 760, 763
 Embedded Bereich .. 579, 832
 encoding 60, 399, 626
 Engineering
 Reverse Engineering .. 139, 144, 242, 246
 Roundtrip Engineering 242
 Englisch 660
 Enhydra 401
 Enigma 581
 ensure 482
 Enterprise
 Java Beans 50, 420, 425, 798, 832, 841
 Technologie 49
 Entität 378
 Entitätsmenge 378
 Entity Bean . 432, 435, 443
 Entwicklungsarbeit ... 15
 Entwicklungskosten . 512
 Entwurfsmuster 842
 Environment Variables 86
 Environment-Variablen .. 307
 EOF 116
 Erich Gamma 495
- ERROR 478
 Etikett 183
 Etikettierung 182
 Exception 7, 103, 476, 623
 AccessControlException 124, 131
 SecurityException . 126
 ExceptionClear() 591
 ExceptionDescribe() . 591
 ExceptionInInitializerError 593, 595–597
 ExceptionOccurred() 591
 Exceptions . 474, 590, 665
 execute 384
 executeQuery 384
 expect 506, 842
 export 133
 ext 189
 extcheck 117
 ExtendedClassWriter 633
 Extension Checker ... 117
 Externalizable 623
 eXtreme Programming .. 204, 206, 668, 678, 756
- F**
- FATAL 478
 FatalError() 591
 fcntl() 579
 Features 211, 725
 Fehlerbehandlung ... 387
 Fehlercode 313
 Fehlerseite 415
 Fehlersuche 512
 Fehlervermeidung ... 482
 Fein-Design 214
 Fernwartung 507
 Files-Abschnitt 736
 final 530, 680
 findByPrimaryKey() . 444
 FindClass() 590
 Finder-Methoden ... 445, 451, 672
 Firewall 409
 Flexibilität 670
 FlipFlop.java 440
 FlipFlopBean.java ... 440

Font-Path 90
font.properties 87
footer 627
Formatierung 656
Formulare 406
Forté 273
Frameworks 670
Free Software Foundation
698
FreeHep 635
Fremdschlüssel 378
FSF 698
ftp 508
FTP 325
Fußzeile 627
Fujaba 239

G

Garbage Collection 8, 31,
512, 513, 578, 832
Garbage Collector 94,
104, 527, 580
Garbage-Collector
Aktivität 545
gc 104
GC 31
gcc 44
GCJ 44
gdb 44, 256, 258, 607
Generalisierung 213
Generalization 213
Generic Collection
Library 239
Generic Types 7
Get-Request 400, 406
GetTypField() 593
GetArrayLength() ... 600
getColumnCount() .. 388
getConnection 383
getDatabaseProduct-
Name() 388
getErrorCode() 387
GetFieldID() 593
GetJavaVM() 605
getMetaData() 388
GetMethodID() 594
getNextException() .. 387

GetObjectArrayElement()
600
GetObjectClass() 593
getpass() 581
getResource() 743
getSQLKeywords() .. 388
getSQLState() 387
GetStaticFieldID() ... 596
GetStaticMethodID() 597
GetStaticTypField-
Funktionen 596
GetStringChars() 598
GetStringLength() ... 598
GetStringUTFChars()
584, 588, 589, 599
GetStringUTFLength() ...
599
GetSuperclass 590
Getter-Methode 648
Getter-Methoden 530
GetTypArrayElements-
Funktionen 601
GetTypArrayRegion-
Funktionen 602
GetVersion 589
getWarnings 387
Gleichheit 498
Glimpse 763
Glossar 831
GML 57
GNATS 760
Gnome 200
GNU 698, 710
Classpath 37
Compiler 44
Debugger 607
General Public Licence
710
GNUJSP 410
time 548
GNU Emacs 260
GoF 669
grant 126, 127
Granularität 528, 672
Graphics 398
Green

OS 6
Projekt 6
Grob-Design 214
group 628
Group 733
Gruppierung 628
GUI 833
Design-Tools 270
Erstellung 282
GUI-Builder 276
gvim 254

H

Hacker 698
Haltepunkt . 102, 258, 283
Hash-Tabelle 475, 525
Hauptabschnitt 111
header 627
Header 159, 733
Header-Datei-
Generierung 582
Heap 94, 537, 833
Analyse 544
Anzeige 545
Size 518
Hello-World-Applet . 397
Hello-World-Servlet . 400
Hello.java 157, 436
HelloBean.java 436
HelloHome.java 436
HelloWorld-Servlet .. 403
Help-Link 628
Helpdesk 758
helpfile 628
Hidden Files 311
Hilfe 93
Hilfe-System 49
Hilfs-Variablen 647
Home-Interface 436, 459,
672, 798, 801
Hot Spots 515
HotJava 6
HotSpot 30, 578
Compiler 101, 282
HPjmeter 542
hprof 536
href 744

-
- HTML ... 50, 55, 394, 618, 624, 635
 Character-Set 628
 Editoren 423
 Ersatzdarstellung . 624
 Generierung 409
 Gerippe 414
 Stylesheet 628
 HTMLStandardWriter ... 632
 HTTP 394
 Protokoll 396
 Requests 409
 Sessions 396
 HTTP-Protokoll 508
 httpd 832
 HttpJspPage 415
 HttpServlet 400
 HttpServletRequest .. 401
 HttpServletResponse 401
 HTTPSession 408
 Hyperlink 394
 Hypertext 63
- I**
 IAS 276
 IBM 373
 Public Licence 712
 iContract 484
 Id 159
 IDE 269, 833
 Einbindung 683
 ident 162, 165
 Identität 498
 IDL 50
 ignore 103
 IIOP 49, 363, 371
 Impl-Klasse 669
 Implementierung 204
 import 133, 656
 Import 172, 174
 Include-Datei 581
 Index 628
 Indizierung 390
 inetd 190, 832
 INFO 478
 InforBus 48
 Informix 373
 init() 122
 Inkompatibilität 116, 324
 Inkonsistenz 455, 682
 Inlining 31, 530
 Inprise 275
 Application Server 276
 InputStream 535
 INSERT 379
 Install-Abschnitt 735
 Installation
 automatisiert 729
 Installationen 719
 Installations-Skripte . 736
 Installations-Toolkit . 720
 Installations-Verzeichnis . 730
 Installer.java 721
 InstallShield 723
 instanceof . 565, 571, 665, 671
 InstantiationException ... 592
 Integrated Development Environment 833
 Interaktionsdiagramm ... 217
 Interface ... 515, 561, 665, 669
 Marker-Interface .. 565
 Serializable 565
 Internet 384
 Link 622
 Internet Inter Orb
 Protocoll 371
 Intranet 384
 Invariante 483, 484
 invoke 574
 ioctl() 579
 IOException 582
 IPL 712
 IsAssignableFrom() .. 590
 IsInstanceOf 593
 iso-8859-1 628
 iso8859-1 60
 IsSameObject() 593
- Ist-Zustand 204
- J**
 J2EE .. 9, 25, 460, 833, 842
 J2ME 9
 J2RE 24
 J2SDK 24
 J2SE 9, 24
 JAAS 49
 Jacobsen 205
 jad 139, 145
 JAF 48
 JAI 48
 Jakarta 324, 401, 476
 James Gosling 260
 Japhar 37
 jar 107
 Jar-Archiv ... 93, 107, 135
 Jar-Datei ... 107, 270, 346
 jarsigner 125, 135, 748
 Java 6
 COM Interface 580
 Community Process ... 779
 Compiler 97, 621
 Database Connectivity 380
 Debugger 398
 Development Kit ... 23
 Dokumentation ... 617
 Foundation Classes . 24
 IDE 269, 275
 Interpreter 92, 266
 Java 2 Plattform 7
 Java Development Kit 8
 Java-Bean 416
 JavaBeans 7, 48
 JavaComm 49
 JavaHelp 49
 JavaMail 48
 JavaServlet 48
 JavaStation 7
 JavaUnix 9
 Kompilation 582
 Mail 460
 Message Service ... 460

- Naming and Directory
 - Interface 460
- Native Interface 24, 577
- Network Launching
 - Protocol 740
- Objekt-Informationen .
 - 589
- Optimierer 552
- Optionen 92, 626
- Plattform 9
- Plugin 418
- Runtime Environment .
 - 23, 97
- Runtime Interface . 580
- Server Pages .. 50, 410,
 - 460
- Server Web
 - Development Kit 410
- Servlet 50
- Servlet Development
 - Kit 401, 410
- Shell 264
- Tools 107
- Transaction API ... 456
- Transaction Service 456
- WebStart 740
- Wrapper 579
- Java 2 121
 - Enterprise Edition ... 9,
 - 25, 833
 - Micro Edition 9
 - Plattform 24
 - Runtime Environment .
 - 24
 - Software Development
 - Kit 24
 - Standard Edition . 9, 24
- Java Virtual Machine . 23
- Java-String 584
- java.lang.System 122
- java.library.path 607
- java.policy 90, 126
- java.security 90
- java.security.Permission .
 - 127
- java.sql 380
- Java2D 24, 48
- Java3D 48
- javac 23, 97
- JavaCC 239
- javadoc 287, 339, 617, 625
- Javadoc
 - Kommentare .. 618, 650
 - Link 622
- javah 581, 607
- javakey 132
- JavaMail 51
- javap 113
- Javascript 745
- javaw 92
- javaws 741, 748
- javax 47
- JAXB 76
- JAXP 75
- Jboss 461
- JBuilder ... 275, 326, 423,
 - 547, 679, 681
- Enterprise 276
- Foundation 276
- Personal 276
- Professional 276
- JCE 48
- jclass 583, 589
- JCP 779
- jCVS 195
- JD 43
- jdb .. 23, 94, 100, 256, 258,
 - 398
- Kommandos 101
- JDBC ... 51, 325, 373, 380,
 - 460
- 2.0 390
- Befehle 385
- Datentypen 385
- Treiber 375, 380
- Tuning 389
- URL 383
- JDBCAppender 482
- JDE 263
- JDK 8, 23, 24
 - 1.0 8
 - 1.1 8
 - 1.1.8 27
 - 1.2 8, 26
 - 1.3 8, 27, 28, 279
 - 1.4 9, 482
- Switching 277
- JDPA 104
- JFace 295
- JFC 24
- jikes 40
- Jikes
 - Debugger 43
- JKESPATH 41
- JIT-Compiler .. 30, 90, 94,
 - 96, 578
- JitterBug 764
- Jlint 682
- JMF 48
- jmk 320
- JMS 51, 460
- JMX 49
- JNDI 48, 433, 460
- JNI 24, 97, 577
 - Datentypen 586
 - Version 589
- JNI-Funktionen 589
- JNI_ABORT 602
- JNI_COMMIT 602
- JNI_FALSE 586
- JNI_TRUE 586
- JNIEnv 583, 589
- JNLP 740, 833
 - API 751
 - BasicService 751
 - ClipboardService .. 751
 - Datei 744, 749
 - DownloadService . 752
 - Elemente 749
 - FileOpenService ... 752
 - PersistenceService . 752
 - PrintService 752
 - Services 751
- Jokerzeichen 311
- Jopt 552
- JRE 23, 97
- jRequest 714
- JRI 580

-
- JSDK 401, 410
 JServ 401
 JSP . 50, 270, 276, 410, 460
 Architekturen 419
 Direktiven 414
 Engine 410, 414
 forward 418
 getProperty 417
 Implementierungen ...
 410
 include 417
 Laufzeitumgebung 418
 plugin 418
 Prinzip 410
 setProperty 417
 Skript-Code .. 414, 418
 Tags 414, 416
 useBean 417
 JSP-Elemente 414
 JspPage 415
 JSR47 482
 JSSE 49
 jstring 583
 JSWDK 410
 JTA 51, 456, 460
 JTest 485
 JTS 51, 456, 460
 JUnit 247, 325, 495
 junit.jar 495
 Just-In-Time 90
 JVM 23, 26, 121
 Interface 605
 JVMPI 535, 543
- K**
- Kaffee 34
 KamaSutra-Technik . 665
 Kategorien 478, 684
 Kathedrale 699
 KDE 198, 712
 Kennzeichnung 161
 Kent Beck 495
 Kernel
 Performance 547
 keystore 126, 747
 keytool 125, 132, 747
 kill 190
- KISS-Prinzip 58
 Klammer-Hervorhebung
 262
 Klasse 669
 Klassen
 Abstrakt 566
 anonyme 319
 innere 319
 Klassen-Graph 545
 Klassen-Operationen 589
 Klassendiagramm ... 211
 Klassennamen 645
 Klassenpfad 92, 98
 KLyX 636
 Kommando-Objekte . 528
 Kommandos 311
 Kommentierung 617, 649
 Kommerzialisierung . 698
 Kommunikation 211, 673
 Komponenten .. 205, 277
 Hardware 230
 Software 230
 Komposition 213
 Kompression 191
 Konfigurations-
 Management 297
 Konflikt 117, 188
 Konsole 479
 Konstanten 671
 Kopfzeile 627
 Kryptographie 48
 kUML 245
 Kurz-Kommentare .. 654
- L**
- Länderkennung 626
 länderspezifisch 663
 Ladezeiten 397
 LaTeX 635
 Laufzeit-Überprüfungen .
 512
 Launch-Time 741
 Layout 410, 419
 lazy 750
 Lazy Initialisation ... 528
 LD_LIBRARY_PATH . 96
 LD_PRELOAD 96
- LDAP 48
 Lebenslinie 217
 Leerzeichen 658
 Legacy-Systeme . 579, 833
 Lernziele 373
 Lesser GPL 711
 LGPL 711
 lines 98
 Lines of Code VII
 Lines Of Code 660
 link 627
 Link 394, 621
 linkoffline 628
 Lisp 261
 list 103
 Listings 793
 Lizenz-Abfragen 144
 Lizenz-Modell 709
 LOC VII, 660
 Local Client 459
 Local Home Interface 459
 Local Interface 459
 locale 626
 Lock 158
 Lock-Strategien 156
 Locker 159
 Log 159
 log4j 476
 Logging 473
 Logik 420
 Login-Anmeldung ... 406
 Lokale Variablen 647
 Lotus Notes 48
 lpd 832
 Lucid 260
 LyX 636
- M**
- Mail 48, 479
 Mailing-Liste 717
 main section 111
 make 257, 302
 rekursiv 314
 Makefile 41, 303
 verteilt 314
 zentral 314

- Makro-Optimierung . 515
- Makros 306, 734
 - intern 308
- man 165
- Management
 - Netzwerk 49
- Mandatory 458
- MANIFEST 93
- Manifest-Datei . 110, 117, 147
- MANIFEST.MF 110
- Manipulation ... 145, 147
- MAPI 48
- Marke 183
- Marker-Interface 364, 534
- Markierung 620
- Marktmodelle 707
- Maschinenencode 511
- mc 112
- MD5 147
- Mehrfachvererbung ... 7, 565
- Meldungen 684
- memory 104
- Merge 186
- Message Driven Beans ... 460
- message pane 280
- Message Service 51
- Meta-Daten 382, 388
- META-INF . 110, 135, 439
- Meta-Sprache 57
- Methoden-Aufruf ... 594
- Methodennamen 645
- Metriken 247, 660
- Metroworks 30
- MICO 832
- Middle Tier 427
- Midnight Commander ... 112
- Migration 749
- Mikro-Optimierung . 515
- Milestones 764
- MIME-Typ 416, 741
 - applicaton/x-java-jnlp-type 745
- MiniSQL 382
- MIT Licence 712
- MKS 10
- mysql.sourceforge.net 375
- Mocha 139
- Monitor-Operationen 604
- MonitorEnter() 604
- MonitorExit() 604
- Mosaics 394
- Mounten 833
- Mozilla 712, 763
- Mozilla Public Licence ... 712
- MPL 712
- MS-Office 637
- MS-Word 635
- Msqldriver 382
- Mult-Threading 7
- Multi-Site 154
- Multimedia 48
- MVC 420
- MySQL 373, 374
- Mythical Man-Month 700
- N**
- Nachbedingung 484
- Nachvollziehbarkeit . 154
- Name 160, 733
- Name Space 121
- Namensdienst ... 48, 368
- Namenskonvention . 656
- NASA 374
- Nathan Meyers 541
- native 581
- Native Method Interface . 580
- Native-Deklaration .. 582
- native2ascii 116, 662
- nativer Code 833
- Navigations-Leiste .. 628
- NDS 48
- NEdit 255
- NetBeans 269, 326
- Netscape ... 394, 580, 712
 - Composer 423
- Netscape Public Licence . 712
- NetSerializableException 371
- Netzwerk 49, 384
- NewTypArray-Funktionen 601
- NewGlobalRef() 591
- NewObject() 592
- NewObjectA() 592
- NewObjectArray() ... 600
- NewObjectV() 592
- NewString() 598
- NewStringUTF 584
- NewStringUTF() 599
- next() 384
- NIS 48
- NMI 580
- NNTP 48
- noarch 738
- NoClassDefFoundError . 87
- nodeprecated 628
- nodeprecatedlist 628
- nohelp 628
- noindex 628
- Non-Commercial-Licence 713
- nonavbar 628
- nosince 628
- NoSuchFieldError .. 593, 596
- NoSuchMethodError 595, 597, 604
- Notation 205
- notree 628
- Novell 48
- nowarn 98
- NPL 712
- null 498
- Null-Suffix-Regel 311
- O**
- Oak 6
- Oberklasse
 - abstrakt 670
- Obfuscator 144

-
- ObjectInputStream . 533
 - Object 113, 587
 - ObjectOutputStream . 533
 - ObjectStreamField ... 623
 - Objekt 102
 - Kreierung 527
 - Pools 527
 - Referenzen 512
 - OO 205
 - Datenbank 380
 - Open Source 697
 - OpenGL 833
 - OpenJIT 30
 - OpenSource 374
 - Definition 709
 - OpenTools
 - API 277
 - OpenTrack 761
 - Optimierung 30, 144, 474
 - OptimizeIt 543
 - Optional Packages 47
 - Optionale Pakete 24
 - Optionen
 - Standard-Optionen . 92
 - OR-Mapping 432
 - Oracle 373
 - OracleDriver 382
 - OROMatcher 239
 - OSF 834
 - OSS 697
 - Lizenzen 709
 - Markmodelle 707
 - OT 761
 - Out of Memory 94
 - OutOfMemoryError .. 94, 590, 592, 593, 595–598, 600
 - overview-summary.html 625
 - P**
 - package 215, 625
 - Package 711
 - Package-Struktur 672
 - Packages 656
 - Paging 527
 - paint() 124, 398
 - Paket
 - Erstellung 738
 - PAM 49
 - Parallel-Entwicklung 155
 - Parameter 621
 - Passivierung 440
 - Passivierungs-Attribut .. 433
 - Passphrase 716
 - Passwort ... 134, 135, 716
 - Passwort-Server 190
 - PATH 87, 97
 - Pattern 840
 - PDA 834
 - PDF 635
 - Peer Review 705
 - Peer Reviews 677
 - Perfektion 702
 - Performance 511, 578
 - Performance-Pack ... 578
 - Permission 127
 - AllPermission 127
 - AudioPermission .. 127
 - AWTPermission ... 127
 - FilePermission 127
 - NetPermission 128
 - PropertyPermission ... 128
 - ReflectPermission . 128
 - RuntimePermission ... 128
 - SecurityPermission 128
 - SerializablePermission 128
 - SocketPermission .. 128
 - SQLPermission 128
 - Permission Class 127
 - Permission denied ... 189
 - Persistenz . 429, 432, 561, 834
 - PersonalJava 34
 - Phantom-Probleme .. 455
 - pharmacy 200
 - Phase
 - Entwurf 204
 - Konstruktion 204
 - Konzeption 204
 - PHP 395, 766
 - Plattform
 - Unabhängigkeit .. 324, 511, 577
 - Plug-In 293
 - Policy
 - Date 122
 - URL 122
 - policytool 126, 131
 - POP3 48
 - Port 508
 - Post-Request 408
 - Postcondition 483
 - PostgreSQL 373
 - PostScript 87, 635
 - PowerPC 30
 - Präsentation 420
 - Precondition 483
 - Prep-Abschnitt 734
 - PreparedStatement .. 384, 386, 390
 - Primärschlüssel 377, 443, 802
 - Primary Key 443
 - print 102
 - Prioritäten 91
 - Prioritäts-Regel 307
 - private 530, 625
 - Problem Domain 204
 - Problem-Bereich 204
 - Profiler 535
 - ProfileViewer 539
 - Profiling 94
 - Interface 535, 543
 - Offline 547
 - Remote 547
 - program
 - counter 103
 - Programm
 - Zähler 103
 - Programmierrichtlinien .. 641, 676, 839
 - Programmierstil 677
 - project pane 280
 - Projekt

- Management 313
- Properties .. 130, 329, 352
 - System 332
 - systemabhängig ... 353
- Property 92
 - Datei 330, 353
 - Element 751
 - java.security.manager . 122
 - java.security.policy 126
 - Objekt 528
 - user.home 124
- Property-Objekt . 673, 835
- PropertyConfigurator 480
- protected 625
- Protection Domain .. 122
- Protokoll
 - SOAP 62
- Protokolle
 - testen 508
- Protokollierung 474
- Prototypen 422, 757
- Prozess 91
 - iterativ 204
 - Verlagerung 521
- Pserver-Protokoll 190
- Pseudo-Targets 306
- pthread 585
- public 562, 625
- Pufferung 535, 550
- Q**
- Q Public Licence 712
- QA Systems 678
- QPL 712
- QStudio™ 678
- Qt 712
- Quelldatei 98
- R**
- Rückgabewert 621
- Race Condition .. 91, 834
- Range Check 121
- Rational
 - Unified Process ... 206
- Rational Unified Process . 204
- Raw Native Interface 580
- Raymond 699
- RCS 157
 - Branch 162
 - Kommandos 163
 - Kommentar 163
 - Schlüsselwörter ... 159
 - Status 163
 - symbolischer Name ... 163
- rcsdiff 165
- RCSfile 160
- rcsmerge 163, 165
- read 549
- Real Programmer 697
- Realtime 9
- Reduktion 145
- Redundanzen 648
- REENTRANT 584
- Refactoring 286
- Refaktorisierung 649
- Referencable 671
- Referenz 31, 333, 498, 574, 834
 - Id 330, 333
 - Implementierung .. 712
- Reflexion 571, 834
- Regeln 303
- RegisterNatives() 603
- Registrierung 603
- Reihe 377, 378
- Relation 378
- Relationen 377
- Release 733
- Release-Verwaltung . 154
- ReleaseStringChars() 599
- ReleaseStringUTFChars() 599
- Releasetag 173
- ReleaseTypArrayElements-
Funktionen 602
- Religion 260
- Remote Client 459
- Remote Home Interface . 459
- Remote Interface 459
- Remote Method
 - Invocation .. 363, 460
- Remote-Interface 436, 440, 672, 798, 801
- Remote-Objekt 366
- remove() 442
- Repository . 172, 297, 834
- Req 763
- ReqNG 763
- request 418
- Request
 - Bearbeitung 420
- Request For Comment ... 834
- Request Tracker 761
- Request-Methode 400
- Requests 399
- require 482
- Required 458
- RequiresNew 458
- Resource-Elemente .. 750
- Ressourcen 104, 743
- response 418
- ResultSet .. 384, 385, 387, 388, 390
- RetroGuard 145
- Reverse Engineering . 236
- Reviews 676
- Revision 160
- RFC 834
 - 1421 135
- Richtlinien 315, 641
- rlog 161, 164
- RMI ... 139, 363, 426, 439, 460
 - Beispiel 795
 - Compiler 367
 - IIOP 49
 - Namensdienst 367
 - Protokoll 363
- rmic 364, 365, 367
- rmiregistry 365
- RMS 698
- RNI 580
- rollback 388, 456
- Root-Package 656

-
- RootDoc 629
 - RPC 426
 - RPM 348, 732
 - RPM_BUILD_ROOT 735
 - rpmrc 735
 - RPMS 737
 - RSA 135
 - rsh 189
 - RT 761
 - rules 303
 - Rules of Optimization ... 514
 - Rumbaugh 205
 - Runnable 665, 671
 - Runtime-Checks .. 7, 512
 - RUP 204, 206
 - S**
 - Samba 764
 - Sandbox 119, 743
 - Sandkasten 119
 - SAX 74
 - Schlüssel
 - öffentlicher 134
 - Algorithmus 135
 - Datenbank 134
 - privat 135
 - privater 134
 - Schnittstelle
 - parallel 49
 - seriell 49
 - Schreibweisen 654
 - Schutz 147
 - Schwimmbahn 226
 - scp 716
 - Scriptlet 414, 418
 - SCSL 712
 - SDM 239, 240
 - sealed 111
 - Security 90, 429
 - Elemente 750
 - Manager . 122, 131, 743
 - Policy 122
 - Warning 742
 - Security-Manager ... 121
 - See Also 622
 - SELECT 379
 - Serialisierung ... 440, 533
 - Serializable 116, 364, 440, 533, 565, 671
 - serialver 116
 - serialVersionUID 116
 - serialwarn 628
 - Seriennummer 116
 - Server 672
 - server.xml 405
 - ServerException 372
 - service() 399
 - Service-Packs 757
 - ServiceException 731
 - ServiceManager 752
 - Services 725
 - Servlet 139, 276
 - Architekturen 419
 - Engine 399
 - Implementierungen ... 401
 - Parameter-Übergabe .. 406
 - Verzeichnis 404
 - servlet.jar 403
 - Servlets 48, 399, 460
 - session 418
 - Session 408
 - Kontext 415
 - Variable 415
 - Session Beans
 - Stateful 431, 440
 - Stateless 430
 - SessionSynchronisation .. 458
 - SetTypField() 594
 - setAutoCommit() ... 388, 390
 - setEntityContext() ... 436
 - SetObjectArrayElement() 600
 - setSecurityManager . 122
 - SetStaticTypField-
 - Funktionen 596
 - Setter-Methode 648
 - Setter-Methoden 530
 - SetTypArrayRegion-
 - Funktionen 603
 - setup 734
 - setup.jar 728
 - SGML 57
 - SHA-1 147
 - Shakespeare 650
 - Shared Library ... 96, 97, 536, 584, 585
 - Shared Object 585
 - Shared Object Pool .. 527
 - Shell 548
 - SHELL 315
 - shuJIT 30
 - shutdown.sh 402
 - Sicherheit 49, 90, 119, 429, 705, 747
 - Sicherheitskonzept .. 741
 - Sicherheitsmodell ... 119
 - Signatur 125, 588
 - SignedBy 127
 - Signierung 124, 748
 - Single-Source-Prinzip 246
 - Skalierbarkeit 429
 - Skeleton 367, 439
 - Skript-Programmierung . 312
 - Smalltalk 8
 - Smoke Test 302
 - SMTP 48
 - SMTP-Protokoll 508
 - SNMP 49
 - SOAP 62
 - Software
 - Börse 713
 - Sonderbedeutung ... 624
 - Sonderzeichen 116
 - SortedSet 515
 - source 98
 - Source 160, 734
 - SourceForge 713
 - sourcepath 626
 - SOURCES 737
 - SourceXchange 713
 - Spalten 377
 - Spaltenname 378

- Spec-Datei 732
 - SPECS 737
 - Speedbar 262
 - Speicher
 - freier 104
 - Verbrauch 545
 - Speicherverbrauch .. 104
 - Speicherverwaltung . 121
 - Sperre 158
 - Spionage 144
 - Spiral-Modell 204
 - splitindex 627
 - Sponsor 713
 - SQL 378, 835
 - Befehle 378
 - Datentypen 385
 - Entry Level 382
 - Fehlercode 387
 - Full Level 382
 - Intermediate Level 382
 - Optimierung 390
 - SELECT-Anweisung ... 384
 - SQLException 387
 - SQLWarning 387
 - Stored Procedures . 390
 - SRPMS 737
 - SSH 198, 715
 - SSL 49
 - Stack-Größe 95
 - Stack-Trace 591
 - Stallmann 698
 - Standard
 - Extension 117
 - Standard Extension 24, 47
 - Standard Query
 - Language 378
 - Standard-Doclet 627, 629
 - Standard-Editor 253
 - Standard-Erweiterung ... 99, 117
 - Standard-Erweiterungen 114
 - Standard-Shell 315
 - Standard-Widget-Toolkit 294
 - StarOffice ... 17, 423, 637
 - startup.sh 402
 - State 160
 - Stateful Session Beans ... 431
 - Stateless Session Beans .. 430
 - Statement .. 384–387, 390
 - static 530
 - statische Felder 596
 - stderr 591
 - Stellvertreter-Objekte 364
 - Stored Procedures ... 387
 - Story Driven Modeling .. 239, 240
 - Story-Boards 240
 - strace 549
 - String 587, 588
 - String Substitution .. 307
 - String-Operationen .. 598
 - StringBuffer 533
 - Strings 533
 - structure pane 280
 - Structured Query
 - Language 835
 - Stub 367, 439
 - Stub-/Skeleton-Generator 364
 - Stylesheet 345
 - stylesheet.css 628
 - stylesheetfile 628
 - Stylesheets
 - extended 63
 - Sub-Shell 311
 - Submit 406
 - Subsysteme 205
 - Suffix-Regeln 310
 - Summary 733
 - Sun
 - Community Process ... 482
 - Community Source
 - License 712
 - Sun ONE Studio 273
 - sunwjit 30
 - Support 15, 703, 708
 - swimlane 226
 - Swing 24, 294
 - swingui.TestRunner . 495
 - SWT 294
 - Sybase 373
 - Synchronisation . 458, 682
 - Synchronisations-Monitor 537
 - Synchronisationsbalken . 226
 - synchronized 530
 - Syntax-Highlighting 253, 255, 262, 270
 - Syslog 475, 478
 - System
 - getProperty() .. 92, 751
 - loadLibrary() . 585, 750
 - out.println 475
 - Trace 549
 - System-Administrator ... 434
 - System-JSP 412
 - Systemroutinen 549
- T**
- Tabelle 377, 378
 - Tabellenname 378
 - Tabulator 303
 - Tags 60, 484, 619
 - tar 107
 - Tar-Datei 346
 - Target 303, 327, 355
 - Targets
 - Standard 318
 - Tastatur-Kürzel 261
 - Tcl/Tk 194
 - TCO 14
 - TCP/IP 697
 - telnet 508
 - Templates 7
 - Test
 - Framework 495
 - Phase 493
 - Reihenfolge 505
 - Versionen 757
 - Test-Fälle 211

- Test-Unterstützung . 325,
 342
 Test-Zertifikat 747
 Testbarkeit 506
 Testen 204, 493
 Tester 702
 TestRunner 495
 Tests 302
 TeX 635
 textui.TestRunner 495
 Textverarbeitung 635
 this 644, 659
 Thread
 Graph 545
 Probleme 549
 sicher 415
 Threads 91, 103, 537
 Green Thread .. 99, 585
 Green Threads 91, 399,
 608
 Gruppe 103
 Native Threads 91, 399
 POSIX 93, 585
 Throw 590
 Throwable 590
 ThrowNew() 590
 time 547
 tkCVS 194
 TkGNATS 760
 TLS 49
 ToDo 287
 Together 245
 Tomcat 276, 401, 410
 tomcat.sh 402
 tools.jar 629
 TopLink 325, 432
 Tracking 154
 Transaktionen ... 51, 388,
 433, 454
 Transaktions-Kontext 456
 Transaktions-Service . 456
 Transaktionsattribute 457
 Transaktionsstufen .. 388
 Transport-Objekte ... 673,
 835
 TreeSet 515
 Treiber
 Typ 380
 Typ-2 391
 Treiberauswahl 391
 Trouble-Ticket-Systeme ..
 758
 TrueType 87
 Tuning 9, 516
 Tupel 378
 two-phase-commit .. 456
 tya 30
 Typ-Signaturen 588
U
 UI-Toolkit 294
 Umgebungs-Variablen ...
 307, 577
 Umgebungsvariable . 86,
 95
 ANT_HOME 326
 CVS_AUTH_PORT 190
 CVS_RSH 189
 DEBUG_PROG 95, 549,
 608
 DISPLAY 96
 DO_NOT_CHECK_MEM
 95
 DO_NOT_FREE 95
 JAVA_COMPILER .. 96
 JAVA_FONTS ... 90, 96
 JAVA_HOME ... 86, 96,
 402, 585, 607
 JDK_HOME 86
 JDK_NO_KERNEL_FIX
 96
 LD_LIBRARY_PATH ..
 96, 536, 607
 LD_PRELOAD . 96, 608
 PATH 97
 THREADS_FLAGS . 96
 TOMCAT_HOME 402,
 403
 UML .. 205, 286, 835, 843
 UML-Tools 238
 Umlaute 116, 662
 underscores 655
 Ungarische Notation 666
 UniCastRemoteObject ...
 366
 Unicode 835
 Unified Modeling
 Language 835
 Unified Modelling
 Language 205
 Unified Process 204
 Uniform Resource
 Identifier 835
 Uniform Resource
 Locator 835
 Uninstall-Klasse 722
 uninstall.jar 728
 UnmarshalException 371
 UnregisterNatives() . 604
 UnsatisfiedLinkError 582,
 585, 609
 Unterstriche 655
 UP 204
 update 390
 Update 176, 271, 379
 UPDATE 379
 Updateable ResultSets ...
 390
 URI 835
 URL 127, 407, 835
 Rewriting 397
 Use Cases 209
 Use-Seite 627
 UTF-Zeichenketten .. 588
V
 VAJ 298
 Value Objects ... 673, 835
 Variable 102
 Variablen
 Recycling 665
 Variablen-Information 98
 vars 98
 Vector 515
 Vendortag 173
 verbose 98, 626
 Verbose-Level 475
 Verifikation 136
 VeriSign 748
 Verklemmung 91, 832

- Verschlüsselung 48
Verschlüsselungsprogramm
 581
Verschleierer 144
Verschleierung 145
versiegeln 111
version 627
Version 622, 623, 733
Versionierung ... 153, 741
Versions-Information 589
Versions-Management ...
 297
Versionsverwaltung . 277
Verteilung 429
Vertrags-Modell 482
VertragsSchutzVerletzung
 482
Verzeichnis 627
vi 253
Video 48
View 420
Viren 12
Virtual Machine 24
Visual Age 298
VisualAge 423, 547
VM 24
 Info 545
VM-Optionen 99
VO 835
Vorbedingung 484
- W**
W3C 56
WAM 206, 756, 846
WAR 325
WAR-Archiv 405
War-Datei 346
WARN 478
Warnung 98
Wartung 703
Wasserfallmodell 204
Web-Seite 716
Web-Seiten
 dynamisch 410
Web-Server 745
Web-Services 62
web.xml 403
- WEB_INF 403
Webbrowsern 741
WebLogic 325, 401
Weblogic Application
 Server 277
weblogic-cmp-rdbms-
 jar.xml 804
weblogic-ejb-jar.xml . 800,
 804
WebRunner 6
WebSphere 401, 423
WebStart 740
WebTTS 762
Wert 378
Werte-Semantik 533
Wertebereich 378
WHERE-Klausel 379
Whiteboard Edition . 245
Widget 294
Wiederverwendung . 668
Wildcards 311, 379
windowtitle 627
Wirtschaftlichkeit 14
Wizard 276, 720, 725
WORA 17, 91, 835
Workbench 294
Workspace 294
World Wide Web .. 6, 394
WOTA 91
Wrapper 409, 667
write 549
Write Once Run
 Anywhere ... 17, 835
writeExternal 623
WWW 6, 394
WYSIWYG 636, 835
- X**
X-Window-System .. 836
X.509 134, 836
X11-Window 96
Xalan 68, 345
XEmacs 259, 683
Xenix 5
Xerces 74, 75, 239
XHTML 62, 63
XLT 345
- XMI 237, 244
XML 55, 327, 345, 409
 Attribute 60
 Aufteilung 352
 Binding 76
 Dokument
 invalid 73
 valid 73
 well-formed 73
 DOM 74
 EDI 61
 Editor 76
 JAXP 75
 Kommentar 60, 327
 Parser 73
 Referenz 352
 SAX 74
 Schematas 70
 Tags 60
 Unterstützung 326
 WML 61
 XMI 61
 Zeichencodierung .. 60
XML4J 74
XP 204, 206, 678, 756
XSL 63
 Prozessor 68
 Transformation 68
XSLT 68
- Z**
Zeichen-Konvertierung ..
 116
Zeichencodierung ... 626
Zeichensatz .. 87, 96, 416,
 629
 Codierung 399
 skalierbar 87
Zeiger .. 31, 574, 581, 834
Zeile 378
Zeilen 378
Zeilenende 312
Zeilennummer 98
Zeilenumbbruch 657
Zertifikat ... 132, 136, 742
Zertifizierungs-Stelle 133

Zip-Datei 107, 346	Zustand	Zwei-Phasen-Commit ...
Zugriffkontrolle 49	Method-Ready 436	456
Zusicherung 485	Nirwana 436	