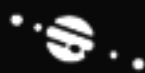


Gottfried Wolmeringer

Java lernen mit Eclipse 3

Galileo Computing 

Auf einen Blick

Vorwort	13
Teil 1 Grundwissen und Installation	19
1 Was ist Java?	21
2 Die Installation von Java	51
3 Was ist Eclipse?	77
4 Die Installation von Eclipse	93
Teil 2 Objektorientierte Programmierung mit Eclipse	115
5 Konzepte der OOP	117
6 Mit Eclipse arbeiten	159
7 Java – ein Baukasten für Objekte	207
8 OOP in der Praxis	267
9 Java-Oberflächen	333
10 Ausnahmen und Threads	399
11 Codekonventionen	425
Teil 3 Datenbanken	461
12 Datenbanken in Eclipse	463
13 Datenbanken und Java	491
Anhang	515
A Tipps	517
B Literatur und Webseiten	519
C Glossar	523
Index	531

Inhalt

Vorwort	13
Hinweise	15
Teil 1 Grundwissen und Installation	19
1 Was ist Java?	21
1.1 Warum programmieren?	23
1.2 Was ist eine Programmiersprache?	25
1.3 Kann auch ich Java programmieren lernen?	29
1.4 Die Java-Story	30
1.5 Wie funktioniert eine Programmiersprache?	38
1.5.1 Compiler	40
1.5.2 Interpreter	41
1.5.3 VMSprachen	42
1.5.4 Imperative und funktionale Sprachen	43
1.6 Wie erlernt man eine Programmiersprache?	47
1.7 Zusammenfassung	48
1.8 Aufgaben zum Kapitel	49
1.9 Webseiten zum Kapitel	50
2 Die Installation von Java	51
2.1 JDK oder JRE?	53
2.2 Hardwarevoraussetzungen	54
2.3 Die Installation	55
2.4 Die Installation der Dokumentation	63
2.4.1 Aufbau der JDK-Dokumentation	64
2.4.2 Test und Einsatz der JDK-Dokumentation	65
2.5 Der Einsatz von Java Web Start	66
2.6 Demos und Beispiele	69
2.7 Programmierung ohne Eclipse	70
2.8 Deinstallation	73

2.9	Zusammenfassung	73
2.10	Aufgaben zum Kapitel	74
2.11	Webseiten zum Kapitel	75

3 Was ist Eclipse? 77

3.1	Was ist eine IDE?	79
3.1.1	Konzepte einer IDE	80
3.1.2	Die Konzepte von Eclipse	83
3.1.3	Eclipse, ein Framework als IDE	84
3.2	Weitere Eclipse-Sprachen	85
3.3	So funktioniert eine IDE und so Eclipse	87
3.4	Zusammenfassung	91
3.5	Aufgaben zum Kapitel	91
3.6	Webseiten zum Kapitel	92

4 Die Installation von Eclipse 93

4.1	Versionen und Milestones	95
4.2	Hardwarevoraussetzungen	96
4.3	Die Installation	97
4.3.1	Installation von Filzip 3.01	98
4.3.2	Eclipse entpacken	99
4.3.3	Eclipse zum ersten Mal starten	101
4.4	Das Language Pack installieren	104
4.5	Die Installation von Plugins	106
4.5.1	Ein UML-Plugin installieren	106
4.5.2	Den GUI-Builder installieren	109
4.5.3	Ein Datenbank-Plugin installieren	109
4.6	Übersicht der Plugins	110
4.6.1	Plugins updaten/deinstallieren	110
4.7	Zusammenfassung	112
4.8	Aufgaben zum Kapitel	113
4.9	Webseiten zum Kapitel	113

Teil 2 Objektorientierte Programmierung mit Eclipse **115**

5 Konzepte der OOP **117**

5.1	Eine uralte Idee für modernste Software	119
5.1.1	Ein neues, altes Paradigma	121
5.1.2	Das Objekt und seine Klasse	122
5.1.3	Ein Baum voller Autos	124
5.2	UML im Einsatz	125
5.2.1	Ein Projekt und seine Pakete	126
5.2.2	Zeichnen statt programmieren	129
5.2.3	Attribute für unsere Autos	133
5.2.4	Auch Autos erben	136
5.2.5	Aus konkret wird abstrakt	139
5.2.6	Identität der Software in Objekten	140
5.2.7	Klassifizierung	143
5.2.8	Vererbung	144
5.2.9	Polymorphismus	148
5.3	Zusammenfassung	155
5.4	Aufgaben zum Kapitel	157
5.5	Webseiten zum Kapitel	158

6 Mit Eclipse arbeiten **159**

6.1	Grundkenntnisse Eclipse	161
6.2	Die Bestandteile	162
6.2.1	Die Bausteine	163
6.2.2	Welt aus Perspektiven	164
6.2.3	Auf der Suche nach dem verlorenen Editor	166
6.2.4	Nutzerfreundlichkeit pur	168
6.3	Die Geburt eines Projektes	171
6.3.1	Ein neues Projekt anlegen	171
6.3.2	Arbeitssets	174
6.3.3	Projekt löschen	175
6.4	Der Aufbau	176
6.4.1	Die Perspektiven	177
6.4.2	Die Menüleiste (Menuubar)	179
6.4.3	Datei	180
6.4.4	Bearbeiten	180
6.4.5	Quelle	181
6.4.6	Refactoring	183
6.4.7	Navigieren	186
6.4.8	Suchen	187
6.4.9	Projekt	187

6.4.10	Ausführen	189
6.4.11	Restliche Menüs	190
6.5	Die Views	190
6.5.1	Die Navigator-View	190
6.5.2	Die Editoren	193
6.6	Projekte übernehmen	200
6.6.1	Bestehende Projekte und Dateien in ein Eclipse-Projekt übernehmen	200
6.6.2	JDK-Projekte und einzelne Dateien übernehmen	201
6.6.3	Ganze Projekte von fremden IDEs übernehmen	201
6.6.4	Andere Eclipse-Projekte übernehmen	203
6.6.5	Änderungen an der Paketstruktur bestehender Projekte	203
6.7	Zusammenfassung	204
6.8	Aufgaben zum Kapitel	205
6.9	Webseiten zum Kapitel	205

7 Java – ein Baukasten für Objekte 207

7.1	Java und OOP	209
7.1.1	Vom Umgang mit Objekten	210
7.1.2	Die API-Dokumentation als Quelle allen Seins	212
7.1.3	Von abstrakten und konkreten Klassen bis zu den Schnittstellen	216
7.2	Zum Innenleben der Objekte	220
7.2.1	Primitive Datentypen	223
7.2.2	Komplexe Datentypen	227
7.2.3	Ein paar Worte zu Arrays, Listen und Stacks	229
7.2.4	Übungen zu den primitiven Datentypen	229
7.2.5	Die Sichtbarkeit in der Praxis	240
7.2.6	Die main-Methode	247
7.2.7	Die Steuerung des Programmflusses	253
7.3	Zusammenfassung	262
7.4	Aufgaben zum Kapitel	263
7.5	Webseiten zum Kapitel	265

8 OOP in der Praxis 267

8.1	Unser Projekt Carlipso	269
8.1.1	Projektdesign adhoc	270
8.1.2	Die Geschäftsobjekte	272
8.1.3	Innere Klassen	274
8.2	Von Vererbung und Polymorphie	294
8.2.1	Java-Klassen im Besonderen	295
8.2.2	Die Syntax bei Java-Klassen	298

8.3	Vom Umgang mit Objekten	302
8.3.1	Das Zusammenspiel von Klassen, Objekten und Methoden	304
8.4	ArrayList, auch ein komplexer Typ?	310
8.4.1	Namensraum in Java	319
8.5	Im Anfang war das Bild	320
8.6	Zusammenfassung	328
8.7	Aufgaben zum Kapitel	330
8.8	Webseiten zum Kapitel	332

9 Java-Oberflächen 333

9.1	Mit AWT lernte Java fensterln	336
9.1.1	Der Ast, an dem das Frame hängt	338
9.1.2	Swing	350
9.1.3	Jigloo, unser Werkzeug für Oberflächen	352
9.1.4	Zwischen Feld und Label	368
9.1.5	Das Ereignis-Modell	375
9.1.6	Modifizierungen an Ereignissen	379
9.2	Für Meldungen aller Art	380
9.2.1	Frame im Frame und interne Dialoge	386
9.2.2	Fashingskostüme für Oberflächen	388
9.3	SWT das Standard Widget Toolkit	390
9.4	Zusammenfassung	394
9.5	Aufgaben zum Kapitel	396
9.6	Webseiten zum Kapitel	397

10 Ausnahmen und Threads 399

10.1	Wo Ausnahmen zur Regel werden	401
10.2	Multithreading mit Java	409
10.3	Dämonen, die »dunkle Seite« in Java?	414
10.4	Instanz-Initialisatoren	418
10.5	Zusammenfassung	421
10.6	Aufgaben zum Kapitel	422
10.7	Webseiten zum Kapitel	423

11 Codekonventionen 425

11.1	Ordnung ist das halbe Leben	427
11.1.1	Weshalb sind Codekonventionen unverzichtbar?	427
11.1.2	Umsetzung der Codekonventionen	429

11.2	Dateiaufbau	429
	11.2.1 Dateinamen	429
	11.2.2 Der Klassenname	430
	11.2.3 Dateiendungen	431
11.3	Innerer Dateiaufbau	432
	11.3.1 Die Bestandteile einer Datei	432
	11.3.2 Der Dateikommentar	433
	11.3.3 Klassen- und Interfaces	434
11.4	Einrücken	435
11.5	Die Zeilenlänge	435
11.6	Kommentare	437
	11.6.1 Quellcodekommentare	437
11.7	Javadoc-Kommentare	439
	11.7.1 Methodenkommentar	440
11.8	Deklarationen	441
	11.8.1 Eine Deklaration pro Zeile	442
	11.8.2 Position von Deklarationen	443
11.9	Klassen- und Interface-Definitionen	444
11.10	Anweisungen	445
	11.10.1 Einfache Anweisungen	445
	11.10.2 Blöcke oder gebündelte Anweisungen	445
	11.10.3 Die if-Weise, if-else-Anweisungen	446
	11.10.4 Die for-Schleife	446
	11.10.5 Die while-Schleife	447
	11.10.6 Die do-while-Schleife	447
	11.10.7 Die switch-Anweisung	448
	11.10.8 Die try-/catch-Anweisung	448
	11.10.9 Die return-Anweisung	449
11.11	Leerräume	449
	11.11.1 Leerzeilen	449
	11.11.2 Leerzeichen	450
11.12	Namenskonventionen	451
	11.12.1 Anwendungen	452
	11.12.2 Packages	452
	11.12.3 Klassenbibliothek	452
11.13	Programmierpraktiken	454
	11.13.1 Zugriff auf Instanz- und Klassenvariablen	454
	11.13.2 Zugriff auf Klassenvariablen und -methoden	454
	11.13.3 Zuweisungen	455
	11.13.4 Instanziierung	455
	11.13.5 Rückgabewerte	456
11.14	Zusammenfassung	457
11.15	Aufgaben zum Kapitel	458
11.16	Webseiten zum Kapitel	459

12 Datenbanken in Eclipse 463

12.1	Einstieg in das Thema	465
12.2	Spielereien mit Access	466
12.3	Die Einrichtung von DbEdit	473
12.4	Allgemeine Betrachtungen zu Datenbanken	475
12.4.1	Zugriffssysteme und Datenbankserver	476
12.5	MySQL	478
12.5.1	MySql installieren	478
12.5.2	Die ersten Experimente mit einem Datenbankserver	482
12.5.3	Datenbanken erzeugen	484
12.6	Eclipse und MySQL	485
12.7	Zusammenfassung	488
12.8	Aufgaben zum Kapitel	488
12.9	Webseiten zum Kapitel	489

13 Datenbanken und Java 491

13.1	Java und die Datenbankanbindung	493
13.1.1	Noch einmal JDBC	495
13.2	Vom Treiber zur Verbindung	496
13.3	Daten aus der Datenbank lesen	498
13.4	Carlipso goes DB	499
13.5	Daten in die Datenbank schreiben	506
13.5.1	Die Reflection-API	507
13.6	Zusammenfassung	511
13.7	Aufgaben zum Kapitel	512
13.8	Webseiten zum Kapitel	513

Anhang 515**A Tipps 517**

A.1	Kommentare und andere Hilfen	517
A.2	Mehr Übersicht schaffen	518
A.2.1	Code minimieren	518
A.2.2	Views entnehmen	518
A.2.3	Editoren automatisch schließen	518

B	Literatur und Webseiten	519
C	Glossar	523
	Index	531

Vorwort

Java ist inzwischen eine sehr bekannte und vielgenutzte Programmiersprache. Man sollte sie halbwegs beherrschen, wenn man sich Softwareentwickler oder Programmierer schimpfen möchte. Eine neue Version, die Version 1.5/5, und neue bessere Programmierwerkzeuge waren der Anlass, dieses Buch zu schreiben. Die Idee ist ganz einfach. Warum sollte man Java mit unzulänglichen Mitteln lernen? Ist es nicht besser, direkt von Anfang an, das beste Entwicklerwerkzeug für Java zu benutzen?

Das berechtigt zur Frage: »Woher will der Autor wissen, was das beste Werkzeug zur Programmierung mit Java ist?« Nun, das kann ein Mensch alleine wirklich nicht entscheiden. Aber der Leser dieser Zeilen hat das Buch nicht gekauft, weil ihm der Einband so gut gefallen hat. Für ihn hat sich der Begriff **Eclipse** bereits mit irgendwelchen Inhalten gefüllt. Irgendwie wurde sein Interesse geweckt. Sagen wir so: »Der Markt hat entschieden, dass Eclipse ein gutes Werkzeug ist.« Und das wurde nicht am Katheder ausgehandelt, sondern in der Praxis, wo der harte Daseinskampf der Programmierer ausgefochten wird.

Zuerst sah es so aus, als würde Java zu einer Sprache für kleine Webseiten-Tools verkümmern. Dann entdeckte man es für richtige Anwendungen, und schließlich programmierte man große Projekte damit. Für die Erstellung von Web-Animationen benötigte man nur einen einfachen Texteditor. Für die ersten Anwendungen genügten Editoren mit der Funktionalität eines **Vi** oder **Emacs**. Aber wer wollte damit ein Team von 100 Universitätsabgängern ausstatten, um ein **PPS** zu erschaffen? Es war nur eine Frage der Zeit, bis für große Aufgaben die großen Werkzeuge geschaffen wurden, und Eclipse ist ein solches Werkzeug.

Wieder könnte man vorlaut sein und fragen: »Warum um alles in der Welt möchte uns der Autor mit einem System, mit dem man riesige Anwendungen schreibt, das Programmieren beibringen?« Nun ist Eclipse zum Glück noch kein abgehobenes System geworden, obwohl es hier und da Anzeichen dazu gibt. Man hat trotz der Professionalität den Anfänger nicht aus den Augen verloren. Gerade die neuste Version (3.01) mit den neuen, so genannten »Cheat Sheets« zeigt dies deutlich. Denn **Cheat Sheets** sind nichts anderes als Lernprogramme.

Trotz des professionellen Werkzeuges werden wir also in diesem Buch ganz vorne anfangen. »Mein Java-Einstieg mit Eclipse« soll ein Einsteigerbuch sein. Das Buch will jedoch nicht das Buch »Mein Einstieg in die EDV« ersetzen und beginnt darum auch nicht mit dem Aufbau des binären Zahlensystems, diese Seiten werden ohnehin immer überblättert. Es geht in erster Linie um Praxis,

und vorrangig werden die Mittel von Eclipse genutzt, um dem Leser die praktische Programmierung näher zu bringen. Daher ist die einzige Voraussetzung, die man mitbringen sollte, die Lust zum Experimentieren, am und mit dem Computer. Ist diese Lust vorhanden, sollte es gelingen, spielerisch in die Welt der Java-Programmierung vorzudringen, auch wenn sie im ersten Moment wie ein undurchdringliches Dickicht wirken sollte. »Spielerisch« heißt jedoch nicht, dass wir Animationen programmieren. Überhaupt werden wir den spielerischen Teil ganz weglassen. Es ist zwar sehr gut, sich von der spielerischen Seite einem Lernstoff zu nähern, aber es lenkt auch vom Kern ab.

Natürlich können auch fortgeschrittene Leser das Buch gerne zur Hand nehmen, um diesen oder jenen Sachverhalt noch einmal zu rekapitulieren. Sie mögen jedoch nicht zu viel erwarten und sollten sich auf die Kapitel stützen, die sie interessieren – es bleibt ein Buch für Anfänger. Der wiederum sollte das Buch wie einen Roman durcharbeiten, von der ersten bis zur letzten Seite. Jedes Kapitel baut auf das vorher Gesagte auf. Danach sollte der geneigte Leser zwar nicht unbedingt der Java-Fachmann geworden sein, aber er hat seine ersten Schritte in einer der wichtigsten Programmiersprachen von heute hinter sich und sollte in der Lage sein, mit Java und Eclipse zu arbeiten.

Zum Buch gibt es eine CD-ROM, auf der sich alle Beispiele des Buches befinden.

Es würde mich freuen, wenn Buch wie CD-ROM dazu beitragen könnten, Spaß und Freude zu fördern, die der schöpferische Prozess zu schenken vermag, auch der, eigene Software zu produzieren.

Gottfried Wolmeringer

Kusel, im März 2005

6 Mit Eclipse arbeiten

6.1	Grundkenntnisse Eclipse	161
6.2	Die Bestandteile	162
6.3	Die Geburt eines Projektes.....	171
6.4	Der Aufbau	176
6.5	Die Views	190
6.6	Projekte übernehmen	200
6.7	Zusammenfassung.....	204
6.8	Aufgaben zum Kapitel	205
6.9	Webseiten zum Kapitel	205

- 1 Was ist Java?**
- 2 Die Installation von Java**
- 3 Was ist Eclipse?**
- 4 Die Installation von Eclipse**
- 5 Konzepte der OOP**
- 6 Mit Eclipse arbeiten**
- 7 Java: Ein Baukasten für Objekte**
- 8 OOP in der Praxis**
- 9 Java-Oberflächen**
- 10 Ausnahmen und Threads**
- 11 Codekonventionen**
- 12 Datenbanken in Eclipse**
- 13 Datenbanken und Java**

6 Mit Eclipse arbeiten

Nachdem wir nun verstanden haben, was eine Klasse ist und wie man mit Objekten programmiert, sollte sich eine positive Beziehung zum Arbeiten mit Eclipse entwickeln. Dazu wird in diesem Kapitel der Umgang mit Eclipse geübt. Was nicht ausschließt, dass man in dem nächsten Kapitel schon die Theorie der OOP und im übernächsten die Java-Programmierung zur Kenntnis nimmt und hin und wieder zurückblättert, um den Umgang mit dem Werkzeug Eclipse zu vertiefen. Genauso gut kann es sein, dass in diesem Kapitel das eine oder andere noch nicht verstanden wird, weil die Sprache auf Dinge zu sprechen kommt, die erst in späteren Kapiteln besprochen werden. Scheuen Sie sich auch nicht, aus kommenden Kapiteln noch einmal nach Kapitel 5 zurückzublättern, um die Eclipse-Bedienung hin und wieder zu vertiefen.

Eclipse wird den ungeübten Anwender zunächst dadurch schockieren, dass es ihm einen ganzen Fächer von verschiedenen Fenstern vor die Nase hält. Es ist aber durchaus sinnvoll, so viel verschiedene »Gucklöcher« auf das zu haben, was man in Form von Code erschaffen möchte. Und, mit den installierten Plugins haben wir noch mehr Möglichkeiten geschaffen, noch mehr Blickwinkel aufgetan, aus denen wir unsere Arbeit machen und betrachten können.

6.1 Grundkenntnisse Eclipse

Bei Editoren gab es schon immer einen heiligen Streit, um die Form der Bedienung. Früher hatte man die Möglichkeit, Befehle einzutippen oder Tastenkürzel zu verwenden. Mit den grafischen Oberflächen kam eine neue Form der Bedienung hinzu, die Tastenkürzel blieben jedoch. Eclipse lässt sich schneller mit diesen so genannten **Shortcuts** bedienen. Deshalb stelle ich sie hier an den Anfang, zumindest die wichtigsten. Man sollte sie die erste Zeit der Einarbeitung immer neben der Tastatur liegen haben. Wir werden später noch einmal auf die Tastenkürzel, speziell im Editor eingehen.

Kürzel	Funktion
Strg + S	Speichern und Kompilieren
Strg + Alt + F	Quelltext formatieren

Tabelle 6.1 Tastenkürzel

Kürzel	Funktion
Strg + ⬆ + <input type="text"/>	Parameter der Methode anzeigen (innerhalb der Klammern)
Strg + <input type="text"/>	Automatische Kompletierung des Namens (Klasse, Methode u.Ä.)
Strg + I	Code Hilfe/Quick fix
Strg + Mausklick	Springe zum Code (zur Definition) des aktuellen Objektes
Strg + 7	Aktuelle Zeile (markierte Zeilen) auskommentieren
Strg + B	Auskommentierung der aktuellen Zeile (markierten Zeile) löschen
Strg + A	Alles markieren
Strg + L	Gehe zu Zeile
Strg + Q	Gehe zur letzten Änderung
Strg + ⬆ + F4	Schließe alle Dateien im Editor
Strg + ⬆ + S	Speichere alle geänderten Dateien aus dem Editor
Strg + .	Gehe zum nächsten Fehler
Strg + ,	Gehe zum vorigen Fehler
Strg + F	Suchen und Ersetzen
Strg + K	Zur nächsten Fundstelle
Strg + E	Zeile löschen
Strg + Z	Änderung rückgängig machen (Undo)
Strg + O	Navigation in der aktuellen Klasse (sehr hilfreich)
⬆ + F2	Öffnen der API-Dokumentation zum aktuellen Kontext
Strg + P	Drucken

Tabelle 6.1 Tastenkürzel (Forts.)

6.2 Die Bestandteile

Einerseits ist Eclipse weit mehr als nur ein Entwicklerwerkzeug, es ist eine Philosophie.

Andererseits besteht es im Grunde genommen nur aus einem leeren Framework, in dem zahlreiche Plugins ihre Arbeit verrichten können. Dadurch ist es so flexibel, dass man Eclipse im Grunde zu allem nutzen könnte, was man mit einem Computer kann.

6.2.1 Die Bausteine

Wenn Sie bereits längjährige Computererfahrung haben, sagt Ihnen vielleicht Lotus 1–2–3 etwas oder Framework von Ashton Tate. Ja, es war Mitte der 80er und ist somit schon eine Weile her. Das waren Gesamtlösungen aus Textverarbeitung, Tabellenkalkulation, Datenbank und Programmiersprache unter einer Oberfläche und in einem Programm. Der Unterschied zu Eclipse ist die Starrheit eines solchen integrierten Softwarepaketes. Es ist eine monolithische Lösung und doch ein Entwicklungsschritt weiter als die reine Textverarbeitung oder Tabellenkalkulation.

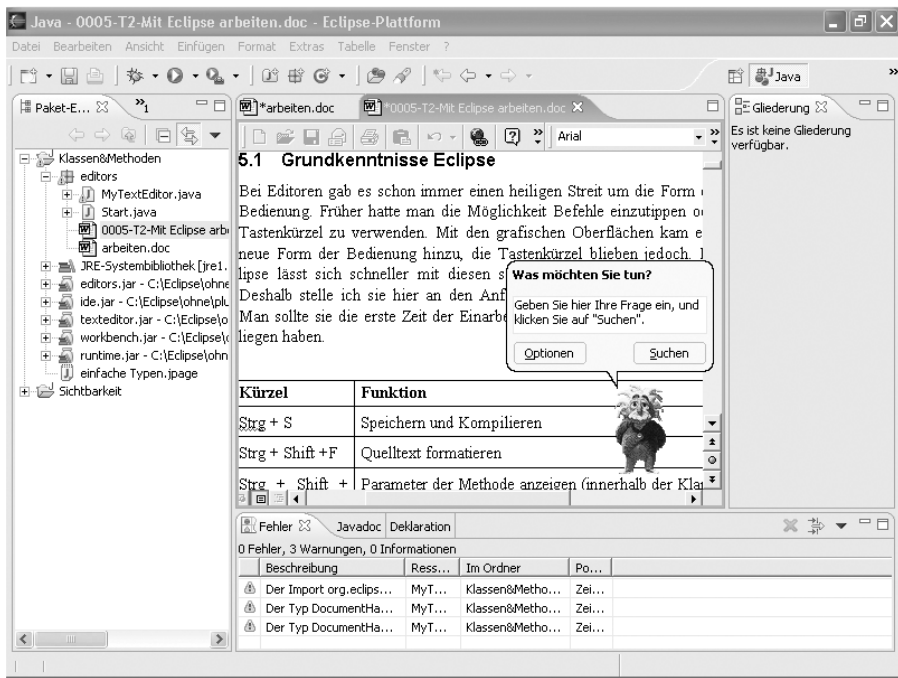


Abbildung 6.1 Word läuft in Eclipse

In Windows lebte diese Idee auf Betriebssystemebene weiter. Verschiedene Anwendungen arbeiten unter einer Oberfläche zusammen. Eclipse wird aus dieser Sicht zu einer Art speziellem Windows. Aber es ist so flexibel, dass man ohne weiteres daraus ein Officesystem machen könnte, wenn man die entsprechenden Plugins dazu programmiert. Ebenso gut könnte man ein CAD-System daraus entwickeln. Der Witz ist eben eine engere Verknüpfung zwischen den Komponenten als Windows es allein möglich macht. Auch Word-Texte lassen sich damit schreiben (Abbildung 6.1, hierzu muss Word allerdings installiert sein), Datenbanken erstellen und verwalten, ja selbst Tetris spielen.

Eclipse setzt kompromisslos das ein, was die objektorientierte Programmierung mit Java bietet und gelangt dadurch zu einer erstaunlichen Leistungsfähigkeit. Die Quintessenz des Systems ist die Präsentation der Daten in verschiedenen **Perspektiven** (Perspectives), **Sichten** (Views) und **Editoren** (Editors).

Workbench, also Werkbank, nennt man in Eclipse die gesamte Oberfläche des Programmes. Mit Sicht (View) bezeichnet man die Teilfenster, aus denen sich das Gesamtfenster, die Workbench aufbaut.

Eine Perspektive ist die thematische Zusammenfassung von bestimmten Teilfenstern zu einer Workbench, die so bestimmten Aufgaben gerecht wird. Es gibt zahllose dieser Perspektiven, die den Anwendungsfeldern von Eclipse zugeordnet sind. Natürlich gibt es eine Java-Perspektive, aber auch Perspektiven für das Debugging, für Ant oder das Testen von Programmen mit JUnit. Jedes installierte Plugin kann weitere Perspektiven mitbringen.

Editoren wiederum sind ganz bestimmte Sichten, eben Views, mit denen man Dateien (Objekte) bearbeiten kann.

6.2.2 Welt aus Perspektiven

Ohne Perspektive ist im Eclipse Arbeitsbereich (Workspace) gar nichts zu sehen. Davon kann man sich leicht überzeugen. Die Perspektiven können rechts oben ausgewählt werden. **RMT** auf die **Perspektiven-Schaltfläche** · **Alle schließen** (Abbildung 6.2): Was bleibt, ist eine leere graue Fläche mit einem Hinweis.

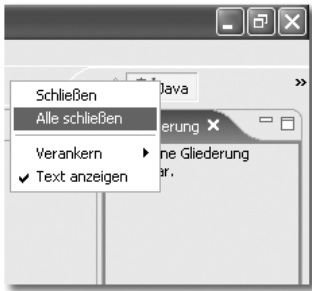


Abbildung 6.2 Alle Perspektiven schließen

Zum Aktivieren einer Perspektive klickt man auf das **Perspektive öffnen**-Symbol rechts oben. Die wichtigen Perspektiven lassen sich direkt auswählen, der Rest über den Menüpunkt **Andere...** Die gleiche Auswahl findet man unter **Fenster · Perspektive öffnen**.

Perspektiven können auch nicht erzeugt, sondern nur kopiert und abgewandelt werden. Die Änderungen, die sich nicht auf die Anordnung der einzelnen Sichten

ten beziehen, also Menüs und Symbolleiste, nimmt man über **Fenster · Perspektive anpassen** vor.

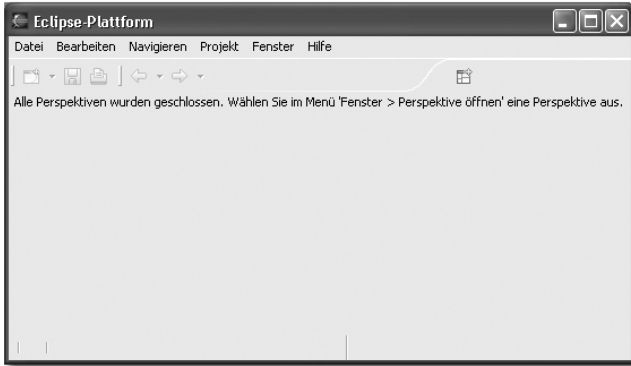


Abbildung 6.3 Framework ohne Perspektive(n)?

Anders ist es mit den Sichten (Views). Sie lassen sich unabhängig von den Perspektiven öffnen und beliebig positionieren. »Beliebig« heißt nun nicht völlig frei auf dem Arbeitsplatz. Es gibt bestimmte Positionen, an denen die Sichten sich anordnen lassen. Die Abbildung 6.4 zeigt die bestehenden Möglichkeiten.

Positionen von Sichten

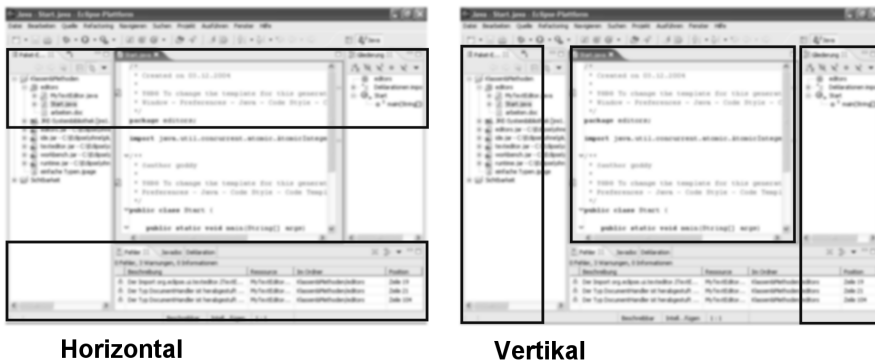


Abbildung 6.4 Positionierung der Views

Die Sichten lassen sich frei um das Editorfeld herum anordnen. Also in der Mitte und auf allen Seiten, außerdem in jedem Teilbereich, in dem bereits eine Sicht liegt.

Innerhalb des Editorfeldes können sie genauso frei angeordnet werden. Editoren können nur innerhalb des mittleren Editorfeldes angeordnet werden. Hier lassen sie sich aber auch an allen Seiten orientieren.

6.2.3 Auf der Suche nach dem verlorenen Editor

Der Java-Editor soll für die nächste Zeit der wichtigste Teil von Eclipse sein. Wir werden uns jedoch zunächst einen ganz anderen Editor anschauen. Einen Editor, der Teil des Debuggers ist.

Es ist ein ganz besonderer Editor, nämlich die so genannte **Scrapbookseite**. Sie läuft auch unter dem Namen **Snippeteditor**.

Warum werden wir ihn verwenden? Er ist besonders gut geeignet, die Eigenschaften der Sprache Java kennen zu lernen. Sozusagen einer der Gründe, warum man gleich mit Eclipse Java erlernen sollte und nicht mit irgend einem Editor (emacs, vi). Deshalb werden wir ihn im Rahmen dieses Buches intensiv einsetzen. Für den Nutzer stellt er eine Art Java-Interpreter dar. Was er in Wirklichkeit nicht ist. Es handelt sich vielmehr um eine Art »Bett« zum Ausführen von Java-Teilprogrammen, wobei er die fehlenden Komponenten der Programmschnipsel zu einem lauffähigen Programm ergänzt, dass er nach der Kompilierung von der virtuellen Maschine ausführen lässt. Die Entwickler von Eclipse kamen wohl durch Smalltalk auf die Idee, einen solchen Editor zu entwickeln.

Um das ScrapBook nutzen zu können, muss man zunächst eine ScrapBook-Datei anlegen. Nachfolgend wird die Datei in einem Editor geöffnet und man kann sie nutzen, um Programmbausteine zu testen.

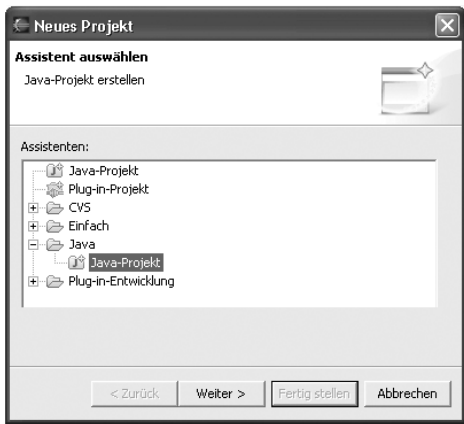


Abbildung 6.5 Projektart auswählen

Eine **Scrapbookseite** lässt sich nur innerhalb eines Java-Projektes anlegen. Also legen wir unser erstes Projekt, das Projekt »Java lernen«, an.

Datei · Neu · Projekt... Im Dialog **Neues Projekt** im Feld **Assistenten:** zunächst **Java** und dann **Java Projekt** auswählen (Abbildung 6.5). Nach Ankli-

cken der Schaltfläche **Weiter** ins Feld Projektname: »Java lernen« eintippen und mit der Schaltfläche **Fertig stellen** abschließen.

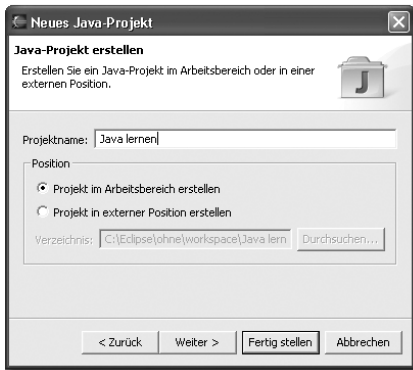


Abbildung 6.6 Projektnamen eingeben

Im nächsten Schritt wählt man mit der Maus in der linken Sicht (Hierarchie- oder Paket-Explorer) das eben erstellte Projekt aus und klickt die RMT. **Neu · Andere... · Java · Java-Ausführung/Debug · Scrapbook-Seite**. Mit der Schaltfläche **Weiter** gelangen wir zur Namenseingabe (Abbildung 6.6). In das Feld **Dateiname:** geben wir **Test1** ein. Nach dem Anklicken der »Fertig stellen«-Schaltfläche öffnet sich der Editor mit einer leeren Seite. Hier ist es nun, das ominöse Scrapbook!



Abbildung 6.7 Namen für die Scrapbookseite eingeben

Wenn wir mit der RMT hineinklicken, bekommen wir ein Kontextmenü zu sehen, das uns mit den deaktivierten Menüpunkten **Untersuchen**, **Anzeigen**, **Ausführen**, **Auswertung stoppen** wie gesagt stark an Smalltalk erinnert (Abbildung 6.8). Schreiben wir einen einfachen Text in Anführungsstrichen hinein, markieren ihn und wählen über **RMT · Anzeigen**. Eclipse erklärt uns nach kurzer »Denkpause«, dass es sich um einen String handelt. Wie man die Scrapbookseite richtig nutzt, werden wir uns bald genauer ansehen.

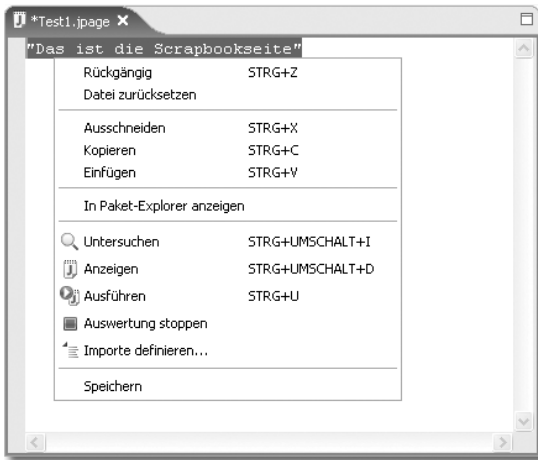


Abbildung 6.8 Das Kontextmenü der Scrapbookseite

6.2.4 Nutzerfreundlichkeit pur

Entscheidend für die Nutzerfreundlichkeit von Eclipse ist die Möglichkeit, Oberflächenkomponenten beliebig zusammenstellen zu können. Es gibt eine Reihe von Standardperspektiven, die man entweder über das Menü **Fenster · Perspektive** öffnen oder über das Perspektiven-Icon (Perspektive öffnen) rechts oben auswählen kann.

Perspektiven lassen sich mit **Speichern als...** kopieren und nach eigenen Vorstellungen anpassen, so dass man sie immer wieder nutzen kann.

Ähnlich wie bei den großen Officepaketen ist es durchaus sinnvoll, das Eclipse-Fenster mehrmals geöffnet zu haben. Es ist nicht so, dass dafür das gesamte Programm vollständig neu in den Speicher geladen werden muss. Der Aufruf erfolgt über: **Fenster · Neues Fenster**.

In diesem Menü wird auch eine Taskliste der aktiven Eclipse-Fenster aufgebaut. So dass man mit dem Fenster-Menü bequem von einem Eclipse zum anderen wechseln kann. Zu beachten ist dabei, dass Eclipse zwar ein neues Fenster öffnet, aber mit speziellen Sachverhalten.

Die verfügbaren Projekte werden zwar im Navigator (Hierarchie- oder Paket-Explorer) angezeigt, doch weder das **ArbeitsSet** ist eingestellt noch eine bestimmte Datei aktiviert.

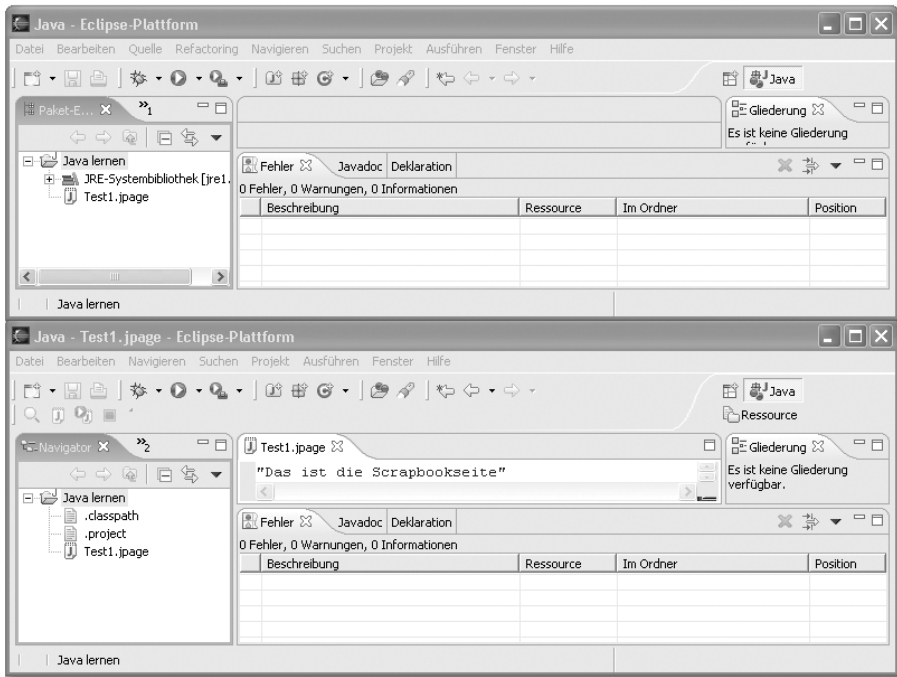


Abbildung 6.9 Zwei Eclipse-Fenster für ein Projekt

Dieses neue Eclipse-Fenster ist ein Unterfenster (Child) des alten Fensters. Wird das erste Eclipse geschlossen, schließt sich das zweite mit. Es wird auch wieder als zweites Fenster geöffnet, wenn Eclipse gestartet wird. Diese Fähigkeit ist natürlich nicht auf ein Child-Fenster begrenzt. Auf diese Art wird es möglich, mit mehreren Perspektiven gleichzeitig zu arbeiten. Sehr interessant ist dies auch für Projekte, die aus mehreren Teilprojekten bestehen, beispielsweise J2EE-Applikationen, Servlets oder JSP-Anwendungen.

In den verschiedenen Eclipse-Instanzen lassen sich natürlich verschiedene Java-Dateien laden. Wenn einmal die gleichen Dateien geladen sein sollten, werden die Änderungen an den Daten untereinander abgeglichen.

Sehr sinnvoll ist beispielsweise auch, eine Eclipse-Workbench für die Codebearbeitung und eine Workbench für die Kompilierung zu nutzen. Auf diese Weise kann man die Java-Perspektive für die Kodierung etwas abspecken, ohne Konsole arbeiten und den Editor vergrößern. Die Gliederungssicht ist auf der linken Seite über den Explorern platzsparender zu positionieren (Abbildung 6.10).

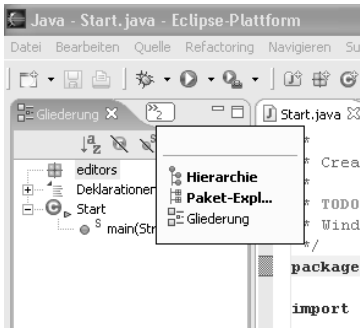


Abbildung 6.10 Die Gliederungssicht nach links verschieben

Im Compiler-Eclipse lässt sich dagegen die Konsole vergrößern und eventuell eine der CVS-Sichten (Teamarbeit) dazulegen (Abbildung 6.11).

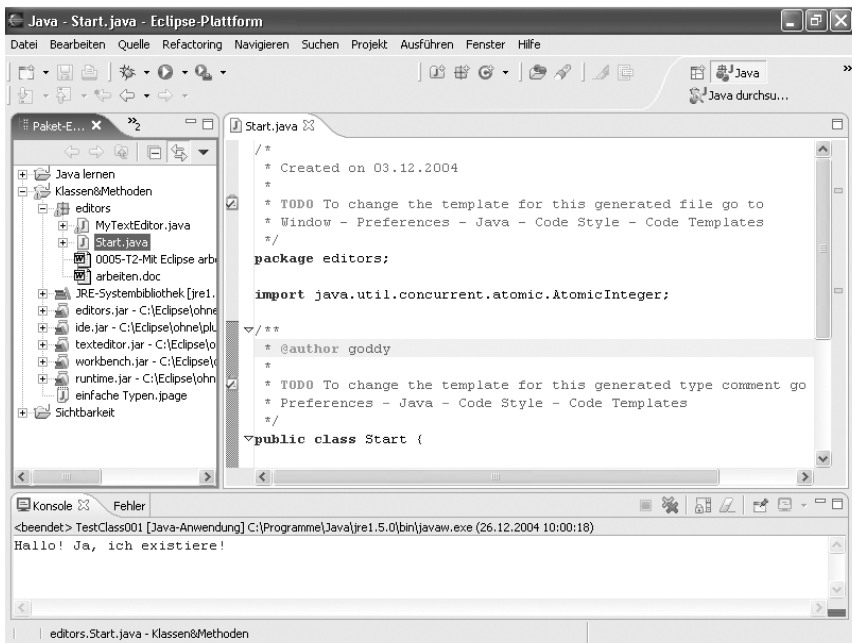


Abbildung 6.11 Eclipse für Compiler-Läufe mit Konsolen- und Fehlersicht

Es ist nicht notwendig, sich möglichst große Editorfenster zu schaffen, denn jedes Fenster kann (wie bei Windows üblich) durch einen Doppelklick auf die Kopfleiste maximiert und durch einen weiten Doppelklick wieder verkleinert werden.

Interessant ist der Einsatz mehrerer Eclipse-Instanzen vor allem, wenn man mit zwei Monitoren arbeitet (Abbildung 6.9). Durch ein optimales Event Handling

der Workbench-Instanzen wird z.B. auch beim Debuggen die aktuelle Position in allen Child-Instanzen angezeigt.

6.3 Die Geburt eines Projektes

Nichts hat sich in den letzten Jahren und Jahrzehnten stärker gewandelt als die EDV-Landschaft. Mit ihr hat sich die Software und die Methodik der Softwareentwicklung verändert. Man sollte sich keinen Illusionen hingeben, Software ist nie dadurch entstanden, dass man eine Idee bekam und sich dann für Wochen vor dem Monitor eingegraben hatte, um in endlosen Hammer-symphonien die geniale Anwendung zu schaffen. Software zu erstellen, war immer auch eine planerische Herausforderung.

Wenn man ein Projekt plant, sollte man zunächst wissen, auf welcher virtuellen Maschine das Endprodukt laufen wird. Eclipse ist ein Java-Produkt und benötigt daher zum Ablauf eine JVM. Die Version 3 benötigt JDK 1.4 oder höher, um funktionieren zu können. Das besagt aber gar nichts über die Java-Software, die mit Eclipse erstellt werden kann, die Wahl dieses Entwickler-JDKs ist nämlich unabhängig von der Runtime-Umgebung für Eclipse selbst. Zusätzlich sollte man berücksichtigen, dass auch Java sich weiterentwickelt und daher die Möglichkeit besteht, dass eine neue virtuelle Maschine verfügbar ist, wenn das Projekt in die Auslieferungsphase kommt. Intensives Testen ist nun angesagt. In anderen Fällen muss man prüfen, ob die erstellte Software auch auf einer älteren VM läuft. Das kann wichtig sein, wenn es sich um ein Webprojekt handelt.

6.3.1 Ein neues Projekt anlegen

Bei der Neuerstellung eines Projektes geht man am besten ganz systematisch vor, um nicht im Nachhinein große Änderungen oder Erweiterungen vornehmen zu müssen. So ist es sinnvoll, selbst ein Arbeitsverzeichnis anzulegen, um unabhängig von den Default-Pfaden zu sein, die standardmäßig innerhalb des Eclipse-Pfades liegen. Gemeint ist natürlich das Verzeichnis **eclipse\workspace**. Man beginnt mit dem Menüpunkt: **Datei · Neu · Projekt** und wählt in der Dialogbox **Java – Java Projekt**. Als Nächstes ist ein Projektname anzugeben. Sinnvoll ist auch hier ein selbstredender Name oder gar der Name, unter dem später das erstellte Programm vertiebt wird, falls er bereits bekannt ist.

Das Projekt kann entweder im Defaultverzeichnis (in der Regel sollte das der Desktop sein) oder in einem vorher angelegten Projektverzeichnis angelegt werden. Dazu ist **Projekt in externer Position erstellen** zu aktivieren und das gewünschte Verzeichnis anzugeben. Arbeitet man mit der Default-Einstellung, wird das Projektverzeichnis mit dem Namen des Projektes unter **C:\soft-**

`ware\ eclipse\ workspace` angelegt. Der aktuelle Eintrag ist aber auch im entsprechenden Feld sichtbar. Wir wechseln auf **Projekt in externer Position erstellen** und geben als Projektverzeichnis `c:\arbeit\java\workspace` ein.

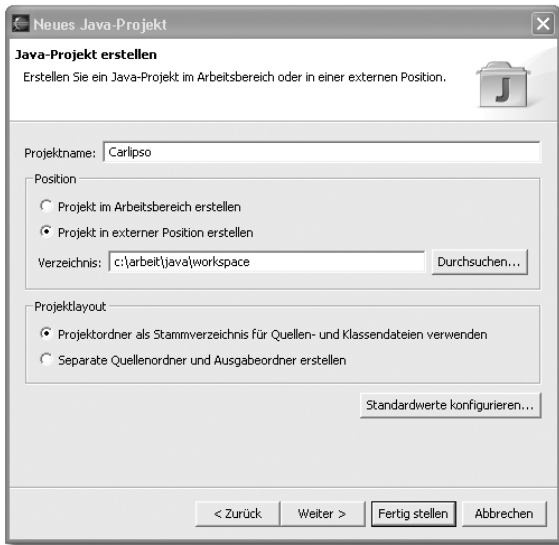


Abbildung 6.12 Projektnamen festlegen

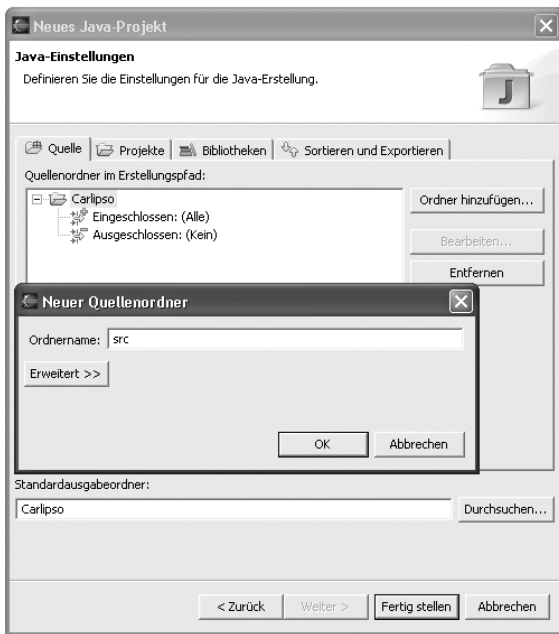


Abbildung 6.13 SRC-Verzeichnis anlegen

Im weiteren Verlauf lassen sich spezielle Quell- und Zielpfade sowie die Libraries angeben. Hier muss man die Schaltfläche **Weiter** nutzen; klickt man die **Fertig stellen**-Schaltfläche, wird ein Standardprojekt erstellt.

Durch die Weiter-Schaltfläche gelangt man auf eine Dialogbox, die mehrere Registerkarten enthält. Bei einer Standardinstallation sind es die Registerkarten:

- ▶ Quelle
- ▶ Projekte
- ▶ Bibliotheken
- ▶ Sortieren und Exportieren

Auf der **Quelle**-Registerkarte kann man das gewünschte Quell- und Zielverzeichnis für den Code des Projektes angeben. Dazu dient die Schaltfläche »Ordner hinzufügen...«. Im folgenden Dialog wird der entsprechende Projektordner ausgewählt und über »Ordner hinzufügen...« ein Quellverzeichnis erstellt (im Weiteren werde ich **src** als Namen des Quellverzeichnisses verwenden) (Abbildung 6.13). Bei CVS-Nutzung bietet sich die Möglichkeit an, das Verzeichnis mit einem View-Verzeichnis oder einen VOB-Verzeichnis zu verlinken. Benutzt wird die »Erweitert >>«-Schaltfläche. Nach der Bestätigung bietet Eclipse Ihnen an, das Projekt als Quellverzeichnis zu entfernen und ein Zielverzeichnis mit dem Namen **bin** zu erzeugen. Bei der Verwendung von CVS müssen Sie natürlich wieder ein eigenes Verzeichnis erstellen und mit dem CVS-Verzeichnis verknüpfen.

Natürlich lassen sich auch nachträglich noch Ordner in ein Projekt einfügen. Hier klickt man das entsprechende Projekt im Navigationsfeld mit der **RMT** an und wählt: **Neu · Ordner**, wieder gibt man den Ordner-Namen an und kann über **Erweitert** ein externes Verzeichnis auswählen, um darauf zu linken.

Auf der **Projekte**-Karte kann festgelegt werden, welche bereits bestehenden Projekte ebenfalls in den Build-Pfad aufgenommen werden sollen.

Die Registerkarte **Bibliotheken** dient dazu, die JAR- und Class-Dateien zu bestimmen, die für das Projekt notwendig sind. Mit **Add JARs** lassen sich JAR-Dateien aus den bestehenden Projekten importieren. Mit der Möglichkeit, External JARs einzubinden, lassen sich, neben den System-Bibliotheken, auch alle denkbar anderen Bibliotheken für die Java-Entwicklung nutzen. Die Systembibliotheken selbst finden sich unter »Bibliothek hinzufügen«, wo auch Plugin-JARs eingebunden werden können. »Variable hinzufügen« bietet zuvor noch die Möglichkeit, Pfade auf Verzeichnisse oder auf einzelne Dateien zu erstellen und zu nutzen sowie die Projekte wunschgemäß zusammensetzen.

»Klassenordner hinzufügen...« bietet noch einmal die Möglichkeit, Klassenverzeichnisse aus bestehenden Projekten einzubinden.

Bei **Sortieren und Exportieren** geht es darum, die Sortierung im Paket-Explorer oder in der Hierarchie-Sicht festzulegen.

Der Paket-Explorer liefert je nach Konfiguration einen Gesamtüberblick über den Arbeitsbereich des Systems. Alles, was sich darin befindet und zum aktuellen Projekt gehört, wird bei einem Build-Vorgang für die Erzeugung des Zielsystems genutzt. Nun ist es aber nicht immer angebracht, alle Dateien des Projektes oder gar mehrere Projekte als Gesamtüberblick im Paket-Explorer vorliegen zu haben und bei der Arbeit ständig die aktuellsten Dateien dort herauszusuchen. Hier bietet der Paket-Explorer so genannte **Arbeitssets** an.

6.3.2 Arbeitssets

Ein Arbeitsset (Working-Set) ist eine Art Filter auf die Dateien, die für Ihre aktuelle Arbeit von Bedeutung sind. Angezeigt werden nur die Dateien des ausgewählten Arbeitssets. Die Arbeitssets sind völlig unabhängig vom Build-Prozess und dienen nur der Erhöhung der Übersichtlichkeit für den Programmierer.

Im Menü des Paket-Explorers (erreichbar über den Pfeil nach unten in der Kopfzeile) werden die verfügbaren Arbeitssets angezeigt. So lässt sich rasch von einem zum anderen Arbeitsset umschalten.

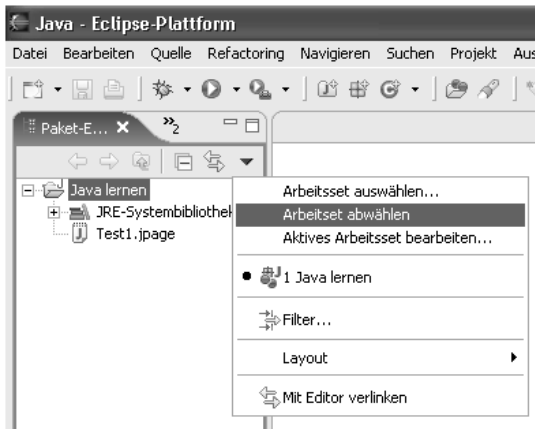


Abbildung 6.14 Arbeitssets abwählen

Es bietet sich an, innerhalb eines großen Projektes mehrere Arbeitssets einzusetzen, die einen sinnvollen Ausschnitt der erstellten Klassen anzeigen. Im Menü kann man auch die Arbeitssets komplett deaktivieren.

Pfeilmenü · Arbeitsset abwählen

Das funktioniert natürlich nur, wenn ein bestimmtes Arbeitsset aktiv ist (Abbildung 6.14). Ein neues Arbeitsset wird über **Pfeilmenü · Arbeitsset auswählen · Neu... · Java · Weiter** angelegt. Einen Namen für das Arbeitsset im Feld **Arbeitssetname**: eingeben (Abbildung 6.15), z.B. Java lernen. Im **Arbeitssetinhalt-Feld** wird das aktiviert, was man im Arbeitsset sehen möchte.

Über den Menüpunkt **Pfeilmenü · Aktives Arbeitsset bearbeiten...** ist es auch ohne weiteres denkbar, bestehende Arbeitssets umzubauen.



Abbildung 6.15 Neues Arbeitsset erstellen

Arbeitssets sind in allen Perspektiven verfügbar, wenn sie erst einmal erstellt wurden. Ein ausgewähltes Arbeitsset ist natürlich perspektivenspezifisch. Das heißt, wenn ein Arbeitsset in der Ressource-Perspektive genutzt wird, wird es nicht automatisch in der Java-Perspektive ebenfalls aktiv.

6.3.3 Projekt löschen

Sie können die in Eclipse erstellen Projekte vollständig löschen, ohne sich des Windows-Datei-Explorers bedienen zu müssen. Eclipse löscht also auf Wunsch nicht nur die Projektdaten im Eclipse-System, sondern auch alle Projektdateien von der Festplatte.

RMT auf den Projektnamen im Paket-Explorer und Menüpunkt **Löschen**. In der Dialogbox können Sie, bevor Sie OK anklicken, festlegen, ob die Dateien physikalisch gelöscht werden sollen oder nicht.

6.4 Der Aufbau

Die Eclipse-Workbench und die verschiedenen Perspektiven sind zusammengefügt aus Bars, Leisten, Sichten (Views) und Editoren. Eine Standardperspektive besteht aus:

- ▶ Menüleiste (Menubar)
- ▶ Symbolleisten
- ▶ Perspektivleiste
- ▶ Navigatorsicht
- ▶ Editor
- ▶ Gliederungssicht
- ▶ Registerleiste
- ▶ Konsole
- ▶ Statusleiste

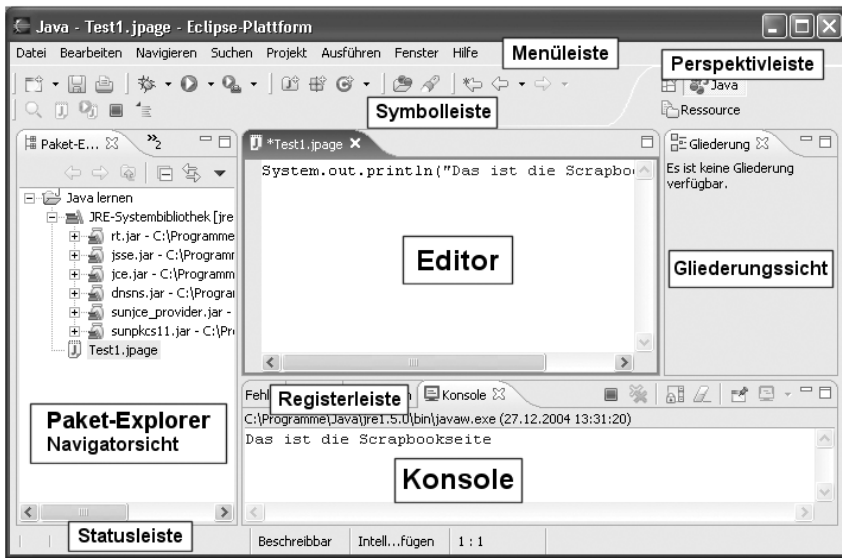


Abbildung 6.16 Der Aufbau der Eclipse-Workbench

Bis auf die Statusleiste, die aktuelle Informationen anzeigt, dienen alle anderen Leisten (Bars) der Bedienung des Systems. Sichten dienen, wie ihr Name schon sagt, zum Anzeigen von Inhalten und Zuständen. Sie können mit dem Menü **Fenster · Sicht anzeigen** ausgewählt werden. Editoren dienen dagegen der Bearbeitung von Dateien. Sie sind mit den Dateitypen verbunden und werden geöffnet, wenn man eine Datei des bestimmten Types erstellt oder öffnen will.

6.4.1 Die Perspektiven

Perspektiven sind entsprechend den Anwendungsfällen von Eclipse zusammengestellte Oberflächen.

Zur Arbeit mit den Perspektiven dienen die Menüpunkte unter **Fenster** · ...

- ▶ **Perspektive öffnen**
- ▶ **Perspektive anpassen...**
- ▶ **Perspektive speichern als...**
- ▶ **Perspektive zurücksetzen**
- ▶ **Perspektive schließen**
- ▶ **Alle Perspektiven schließen**

Perspektive öffnen

Mit **Perspektive öffnen** lässt sich, je nach Vorhaben, die entsprechende Perspektive öffnen. Sollte man ein Java-Projekt öffnen und eine unpassende Perspektive, wie z.B. die Resource-Perspektive aktiv sein, wird Eclipse vorschlagen, die passende Java-Perspektive zu öffnen. Falls man über die Toolbar das Debugging startet, wird Eclipse die Debug-Perspektive aktivieren.

Selbst wenn noch keine Plugins installiert sind, verfügt Eclipse bereits über eine stattliche Anzahl (acht) an Perspektiven.

Daraus lassen sich jederzeit neue Perspektiven bilden. Je nachdem, welche Perspektive geöffnet ist, wird eine andere Auswahl an Perspektiven unter **Fenster** · **Perspektive öffnen** angezeigt. Eine Gesamtliste wird erst unter dem nächsten Menüpunkt **Andere...** geöffnet.

Perspektive anpassen...

Über diese Funktion lassen sich einzelne Menüpunkte der aktuellen Perspektive individuellen Erfordernissen anpassen.

Es sind die Menüpunkte:

- ▶ **Datei** · **Neu**
- ▶ **Fenster** · **Perspektive öffnen** – z. B. Java
- ▶ **Fenster** · **Sicht anzeigen**

In der »Perspektive anpassen«-Dialogbox lassen sich diese Menüpunkte in der Untermenü-Optionsbox vorwählen (Abbildung 6.17). In den Direktaufruf-Kategorien und dem Direktaufruf-Fenster lassen sich die Punkte bestimmen, die in den entsprechenden Menüs enthalten sein sollen.

Eine zweite Registerkarte (Befehle) erlaubt umfangreichere Einstellungen zur Zusammensetzung der Menü- und Symbolleiste einer Perspektive.

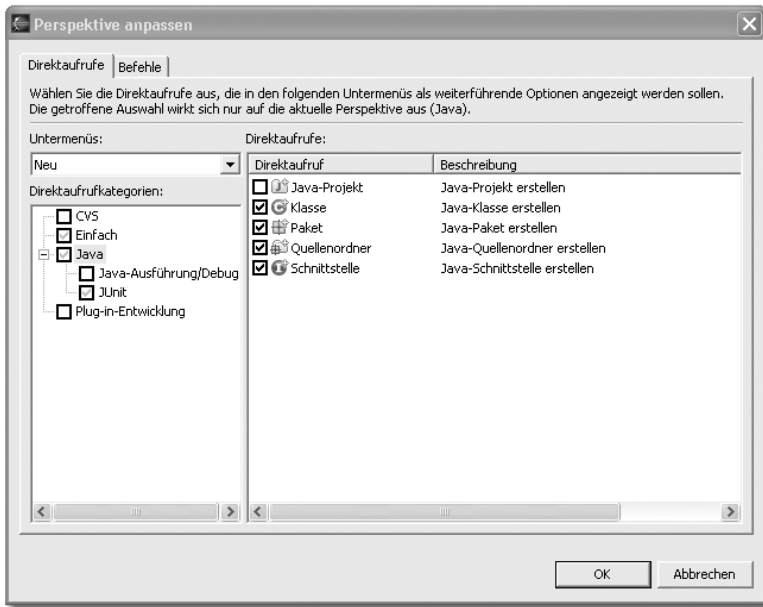


Abbildung 6.17 Perspektive anpassen

Die Sichten einer Perspektive lassen sich individuell anordnen, neben- und übereinander legen. Schließen lässt sich eine Sicht mittels ihrer Systemschaltfläche (X in der Kopfleiste). Eine geschlossene View kann wieder mit **Fenster · Sicht anzeigen** zurückgeholt werden.

Perspektiven sind keine unabänderbaren Oberflächen, sie können jederzeit umgruppiert werden. Deshalb ist es wichtig, dass man sie wieder in den Urzustand zurückversetzen kann. Das gelingt am einfachsten mit **Fenster · Perspektive zurücksetzen**.

Die zwei noch fehlenden Menüpunkte der Perspektiven-Steuerung betreffen das Schließen von Perspektiven:

- ▶ **Perspektive schließen**
- ▶ **Alle Perspektiven schließen**

Wird eine Perspektive geschlossen, zeigt Eclipse die letzte vorher verwendete Perspektive an. Die geschlossene Perspektive verschwindet auch aus der Perspektivbar am rechten oberen Rand. Sie kann also nur wieder über das Menü oder das erste Icon in der Perspektivleiste geöffnet werden.

Sollten alle Perspektiven geschlossen werden, so wird außerdem das Projekt gespeichert, falls die aktuellste Version noch nicht gespeichert worden ist.

6.4.2 Die Menüleiste (Menubar)

Wenn alle Perspektiven geschlossen sind, sehen Sie Eclipse sozusagen in der abgespeckten Form. Die Menüleiste enthält nur die Menüpunkte, die nicht zu einer Perspektive zu zählen sind. Es sind nur wenige Menüpunkte aktiv. (Unter **Fenster** gibt es eine Reihe Punkte, die aktiv und trotzdem ohne Funktion sind.)

Aktive Menüpunkte sind im Grunde nur:

Datei · Arbeitsbereich wechseln...	Anderes Arbeitsbereich-Verzeichnis (Workspace) einstellen
Datei · Beenden	Schließen des Programmes
Fenster · Neues Fenster	Eclipse noch einmal öffnen
Fenster · Perspektive öffnen	dient zum Öffnen einer Perspektive und ermöglicht so erst die konkrete Nutzung von Eclipse (siehe Kapitel: Perspektive öffnen).
Fenster · Benutzervorgaben	Hiermit lassen sich alle wichtigen Einstellungen am System durchführen. Die Einstellmöglichkeiten sind übersichtlich in einem Strukturbaum angeordnet. Viele Plugins hängen hier ihre Properties-Einstellungen mit ein, so dass man für alle Einstellungen erst einmal unter diesem Menüpunkt nachsehen sollte, bevor man sie woanders sucht. Die Zusammenstellung hängt zwar von den installierten Plugins ab, jedoch nicht von der aktiven Perspektive.
Hilfe	Die Menüpunkte des Help-Menüs sind auch ohne Perspektive verfügbar. Insbesondere gibt es zu jedem wichtigen Plugin eine Welcome-Seite, auf der man nähere Hinweise zur Bedienung findet.

Es sprengt unweigerlich den Umfang dieses Buches, wenn wir nun alle Menüpunkte aller Plugins durchsprechen würden. Es sollen hier nur die wichtigsten betrachtet werden.

Aktivieren Sie nun die Java-Perspektive, um die Erweiterungen der Menüs durch diese Perspektive kennen zu lernen.

Fenster · Perspektive öffnen · Andere... · Java Die Menüs werden zusätzlich erweitert, wenn eine Java-Datei geladen ist und der Mauszeiger im Editor steht.

6.4.3 Datei

Datei · Externe Datei öffnen... Ohne Eclipse verlassen zu müssen, lassen sich mit diesem Menüpunkt externe Dateien in Textdateien, Grafiken und Ähnliches öffnen. Je nach installiertem Plugin kann Eclipse diese Dateien auch richtig anzeigen. Falls kein passendes Plugin verfügbar ist, versucht Eclipse die Datei mit dem Texteditor anzuzeigen.

6.4.4 Bearbeiten

Die ersten Bearbeiten-Menüpunkte verstehen sich von selbst. Interessant wird es mit dem Punkt:

Bearbeiten · Auswahl erweitern auf Die aktive Auswahl im Editor lässt sich nach verschiedenen Kriterien erweitern (zum nächsten, vorigen Element).

Bearbeiten · QuickInfo-Beschreibung anzeigen (F2) Zeigt eine funktionelle Beschreibung der Klasse, Methode oder des Tags an, auf der der Mauszeiger gerade steht oder was markiert ist. Setzen Sie diese Funktion ein, wenn sie einen Parameter oder dessen Typ suchen.

Nutzen Sie die Funktion ebenfalls, wenn Sie die Beschreibung des aktuellen Fehlers benötigen. Falls mit dem aktuellen Term ein Fehler verbunden ist, wird die ausführliche Fehlermeldung angezeigt.

Bearbeiten · Unterstützung für Inhalt (Strg+Leertaste) Eclipse versucht Ihnen konkrete Vorschläge zur Schreibweise zu unterbreiten. Die Vorschläge sind kontextbezogen, hängen also von der Position des Mauszeigers im aktuellen Quellcode ab. Von Bedeutung sind nur die Zeichen vor dem Mauszeiger.

Setzen Sie diese Funktion ein, um Ihre Arbeitsweise zu beschleunigen. Schreiben Sie den aktuellen Term immer nur so weit aus, wie nötig, damit Eclipse ihn identifizieren kann. Verwenden Sie jetzt die STRG+Leertaste zur Vervollständigung. So lässt sich die Tipparbeit wesentlich beschleunigen. Verwenden Sie die Funktion, wenn Sie die Schreibweise einer Methode, einer Klasse oder eines Parameters suchen.

Bearbeiten · Schnellkorrektur (Strg+1) Die Funktion dient zur Behebung des aktuellen Fehlers. Er werden meist mehrere Möglichkeiten angeboten, die naheliegendsten sind zuerst aufgeführt.

Die alternativ angebotenen Lösungen lassen sich auswählen, wobei dabei die Änderungen oder der einzufügende Quelltext angezeigt wird.

Bearbeiten · Parameterhinweise (Strg+Shift+Leertaste) Speziell die verschiedenen Signaturen einer Methode oder des Konstruktors mit den Parametern werden mit dieser Methode angezeigt. Die gewünschte Schreibweise kann ausgewählt und eingefügt werden.

6.4.5 Quelle

Diese Menüspalte betrifft den Quellcode und seine Bearbeitung. Die meisten Punkte sind erst aktiv, wenn der Mauszeiger sich innerhalb des Editors befindet oder etwas markiert ist.

Die ersten Punkte betreffen die Kommentierung. Der Wert eines Quellcodes steigt und fällt mit seiner Kommentierung, somit sollte man diese Funktionen nicht vernachlässigen. Wir werden sie trotzdem nicht besprechen, weil sie sich von selbst verstehen.

Die Schiebe-Funktionen (**Quelle · Nach rechts/links**) verschieben den markierten Quellcode entsprechend der Tab-Weite.

Den Punkt **Quelle · Formatieren** hatten wir schon kurz erwähnt. Mit ihm lässt sich der Quellcode in die Form bringen, die bei **Fenster · Benutzervorgaben** eingestellt ist. Die Formatierung kann dabei recht gezielt eingesetzt werden. Ist nämlich ein Teil des Codes markiert, wird Eclipse nur den markierten Code formatieren und den Rest der Datei unverändert lassen.

Quelle · Einrückung korrigieren ist ebenfalls eine interessante Funktion. Sie korrigiert unabhängig von der richtigen Formatierung die Einrückung entsprechend der Klammerebenen.

Mittels **Quelle · Sort Member** lassen sich die Klassen und Methoden in einer Java-Datei sortieren. Diese Funktion dient unabhängig von einer Formatierung dazu, den Code übersichtlich zu halten.

Wofür man beim Kodieren ebenfalls Mühe aufwenden muss, sind die **Importe** der verwendeten Pakete. Nicht nur, dass sie korrekt referenziert sein müssen, man muss sich auch ständig Gedanken machen, ob man die Klasse noch einzeln anzieht oder lieber mit einem Jockerzeichen das ganze Paket. Eclipse nimmt einem diese Arbeit ab. Die Einstellung mittels **Fenster · Benutzervorgaben · Java · Codedarstellung · Importe Verwalten** legt die Sortierreihenfolge fest und bestimmt, ab wie viel angezogenen Klassen pro Paket mit einem * gearbeitet werden soll. Die Sortierung der Imports kann in eine Datei exportiert und so auf andere Rechner übertragen werden. Später lässt sich per **Quelle · Importe verwalten** die Liste der Imports im Quelltext automatisch in Form bringen.

Mit **Quelle · Import hinzufügen** lässt sich eine einzelne Klasse importieren, wenn man sie zuvor markiert hat. Das ist insbesondere bei gleichnamigen Klassen aus verschiedenen Paketen interessant.

Bei der Funktion **Quelle · Methoden überschreiben/implementieren...** geht es wirklich nur darum, geerbte Methoden zu überschreiben und nicht darum, eine völlig neue Methode einzufügen. Es wird eine Übersicht aller ererbten Methoden angezeigt, in der die gewünschten Methoden markiert werden können. Zusätzlich kann die Position gewählt werden, an die die neue Methode im Quellcode hingesetzt werden soll.

Steht der Mauszeiger auf dem Namen einer Variablen, lassen sich über **Quelle · Getter und Setter generieren...** automatisch die zugehörigen Getter- und Setter-Methoden erstellen.

Zu den verwendeten Klassen und ihren Methoden lassen sich mittels der Funktion **Quelle · Stellvertretermethoden generieren...** delegierende Methoden erzeugen, die den Aufruf der eigentlichen Methode bereits integriert haben.

Zunächst zeigt die Funktion **Quelle · Konstruktor mit Feldern generieren...** eine Übersicht der bereits definierten Variablen. Sie lassen sich markieren und gehen so in die Signatur des Konstruktors ein.

Über den Menüpunkt **Quelle · Konstruktor aus Superklasse hinzufügen...** erhält man eine Dialogbox mit allen Konstruktoren der aktuellen Superklasse. Der Konstruktor, der verwendet werden soll, wird markiert und mit einem Mausklick erstellt Eclipse die neue Konstruktor-Methode.

Die aktuelle Methode mit einem Javadoc-tauglichen Kommentar zu versehen, gelingt ganz einfach mit dem Menüpunkt: **Quelle · Javadoc-Kommentar hinzufügen.**

Geht es darum, das Event Handling zu erstellen, leistet ein weiterer Menüpunkt gute Dienste: **Quelle · In TRY-/CATCH-Block einbetten.** Es genügt die betroffene/n Zeile/n zu markieren und diese Funktion aufzurufen. Eclipse erstellt daraufhin einen sauberen try-/catch-Block.

Quelle · Zeichenfolgen auslagern... Der Name ist etwas unglücklich gewählt. Es geht um die Texte in Anführungszeichen, also in der Form »Ein Text«. Diese so genannten String-Literale werden gesucht und in einer Tabelle angezeigt. Die Dialogbox bietet die Variante, die Strings an- oder abzuwählen. Sie werden mit einem Key versehen und in eine externe Datei ausgelagert. Auf diese Weise kann das Programm sehr einfach internationalisiert, also in verschiedenen Sprachversionen realisiert werden.

Im Gegensatz dazu sucht die folgende Funktion **Quelle · Auszulagernde Zeichenfolgen suchen...** nur die möglichen Kandidaten für eine Auslagerung der Strings heraus.

Quelle · Zeilenbegrenzer konvertieren in Das die verschiedenen Betriebssysteme unterschiedliche Absatzzeichen benutzen, ist Ihnen vielleicht schon bekannt. Statt Konvertierprogramme wie **dos2unix** zu verwenden, können Sie den Zeilenumbruch mit diesem Menüpunkt so einstellen, wie es dem Betriebssystem entspricht. Es wird immer die gesamte Datei konvertiert, egal ob eine Auswahl besteht oder nicht.

6.4.6 Refactoring

Unter Refactoring versteht man die Überarbeitung eines Programmes, um es zu verbessern. Daher sind die Funktionen dieser Menüspalte für die Entwicklungsarbeit fast noch wichtiger, als die Möglichkeiten unter der Spalte Quelle.

Rückgängig und **Wiederherstellen** beziehen sich auf die letzte, aufgerufene Funktion.

Wollte man eine Methode oder Klasse umbenennen, so ging das bisher so: Umbenennen und Suchfunktion aufrufen und suchen, wo der alte Begriff überall vorkommt. Mit dem Menüpunkt **Refactoring · Umbenennen...** ist es mit Eclipse ein Kinderspiel. In einer Dialogbox kann zusätzlich ausgewählt werden, welche Arten von Referenzen umgetauft werden sollen und ob die Getter- und Setter-Methoden ebenfalls angepasst werden.

Möchte man eine Klasse innerhalb der Paketstruktur verschieben, so gelingt das mit **Refactoring · Versetzen...** Alle betroffenen Referenzen werden angepasst.

Folgender Menüpunkt ist zwar mehr Bequemlichkeit, aber hin und wieder doch sehr nützlich, vor allem wenn man so genannte Exceptions weiterreichen möchte. Es geht um die Erstellung von Methodensignaturen mittels Dialogbox über **Refactoring · Methodensignatur ändern...**

Anonyme Klassen sind Klassen, die direkt an der Stelle, wo man sie benötigt, auch definiert sind und folglich keine Instanziierung und keinen Objektnamen kennen. Meist benötigt man nur eine bestimmte Methode und erstellt deshalb eine solche Klasse. Als Beispiel die Closing-Methode des WindowAdapters:

```
addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
```

```

    {
        System.exit(0);
    }
});

```

Hinter **new WindowAdapter()** wird hier eine Klasse beschrieben. Sie ist zwar kurz aber hin und wieder störend und man möchte sie gerne ausprogrammieren. Warum soll das nicht Eclipse übernehmen? Mit der Funktion **Refactoring · Anonyme Klasse in verschachtelt konvertieren...** muss man nur noch einen Klassennamen angeben und es ist in einer Sekunde getan. Zwar wird die neue Klasse in unserem Beispiel die Klasse WindowAdapter erweitern, bleibt aber eine innere Klasse, also in der aktuellen Datei. Will man sie in eine eigene Datei auslagern, kann man dazu den nächsten Menüpunkt verwenden: **Refactoring · Membertyp in neue Datei versetzen...** Der Mauszeiger muss dazu auf dem Namen der Klasse stehen, die ausgelagert werden soll.

Die nächsten beiden Menüpunkte dienen dazu, Methoden innerhalb der Klassenhierarchie zu verschieben. Und zwar verschiebt **Refactoring · Herabsetzen...** zur Subklasse und **Refactoring · Heraufsetzen...** zur Superklasse. Es wird jeweils eine große Dialogbox geöffnet, in der die zu verschiebenden Methoden noch einmal ausgewählt werden können. Natürlich kann man keine Methoden aus einem Interface in eine Klasse schieben und umgekehrt.

Wie ist es, wenn man verschiedene Klassen hat, deren Aufbau man für weitere, ähnliche Klassen festlegen möchte? Man sollte eine Schnittstelle (Interface) generieren können, einfach in dem man festlegt, welche Methoden man aus der aktuellen Klasse in einem Interface festschreiben möchte. Genau das tut der Menüpunkt: **Refactoring · Schnittstelle extrahieren...** In einer Dialogbox können die gewünschten Methoden ausgewählt werden.

Refactoring · Typ generalisieren... Ein Typ wird auf eine seiner Superklassen umgesetzt. In einer Dialogbox werden die möglichen Typen für den aktuell markierten Typ angezeigt.

Einen etwas komplexeren Hintergrund hat der Menüpunkt **Refactoring · Nach Möglichkeit Supertyp verwenden...** Überall dort, wo der aktuelle markierte Typ verwendet wird (bei Parameterübergaben), wird er durch einen seiner Supertypen ersetzt. Welcher Typ verwendet wird, lässt sich in einer Liste der Superklassen auswählen. Die Funktion ist interessant, wenn man nachträglich Klassen ableitet, die ebenfalls von Methoden genutzt werden, wie eine bereits bestehende Klasse. Nun müssen alle Handlings der Klasse auf ihren Supertyp umgestellt werden, damit auch die neuen Klassen verwendet werden können.

Code zu integrieren, bedeutet nichts anderes als Variableninhalte direkt in dem Aufruf zu setzen. Auf diese Weise wird die Variable unnötig, der Code wird unter Umständen schneller. Es handelt sich also um eine Optimierung, die per **Refactoring · Integrieren...** automatisch durchgeführt werden kann.

Beispiel für eine Codeintegration:

```
int erg = 0;
int i = 10;
erg = errechneXBis(i);
```

Nach der Integration sieht der Code so aus:

```
int erg = 0;
erg = errechneXBis(10);
```

Wie oft haben Sie schon festgestellt, dass die eben getippten Zeilen vielleicht doch besser in einer eigenen Methode stehen würden, weil Sie den Code gerade noch einmal benötigen? Nichts einfacher als das, markieren Sie die Zeilen und rufen Sie den Menüpunkt **Refactoring · Methode extrahieren...** auf. Natürlich muss man noch angeben, wie die Methode heißen soll und schon ist die Arbeit getan.

Das Gleiche gibt es auch für lokale Variablen. Es kann positiv sein, das Ergebnis eines Therns in einer lokalen Variablen abzulegen. Das ist natürlich rasch getan, aber noch schneller geht es über: **Refactoring · Lokale Variable extrahieren...**

Ein Literal, das zum zweiten Mal vorkommt, erweckt vielleicht schon den Verdacht, man könnte eine Konstante definieren, beim dritten Mal überlegt man sich schon: »Wo waren nun die beiden anderen?«. Vielleicht hat man es früher sogar sein lassen, weil man sich vor der Suche scheute. Mit Eclipse und dem Menüpunkt **Refactoring · Konstante extrahieren...** ist es ein Kinderspiel. Die Konstante wird erzeugt und alle gleichen Literate durch sie ersetzt.

Ein Ausdruck wird mit **Refactoring · Parameter einführen** aus einer Methode in den Aufruf verlegt.

Eine Factory-Methode (Factory Designpattern) für eine Klasse, erzeugt die folgende Funktion: **Refactoring · Factory einführen...** Markieren Sie den Konstruktor der betreffenden Klasse vorher. Es wird eine **create <Klassennamen>**-Methode erstellt.

Die Funktion **Refactoring · Lokale Variable in Feld konvertieren...** macht ganz einfach aus einer lokalen Variablen eine Instanzvariable. Die lokale Wertzuweisung bleibt erhalten. Die Sichtbarkeit kann festgelegt werden.

Den umgekehrten Weg geht folgender Menüpunkt: **Refactoring · Feld kapseln...** Der direkte Zugriff auf eine Instanzvariable wird unterbunden, indem entsprechende Methoden angelegt und die Zuweisungen ersetzt werden.

Die Verwendung der eben besprochenen Funktionen wird noch klarer, wenn wir sie bei der Arbeit an unseren Beispielen verwenden.

6.4.7 Navigieren

Eine Unzahl an Möglichkeiten den Mauszeiger gezielt zu setzen oder externe Dateien einzusehen, bietet Eclipse natürlich auch.

Die Go-Funktionen bieten speziell mit Go To eine Reihe Möglichkeiten.

Hin und wieder gibt es Probleme sich in den Klammerebenen zurechtzufinden. **Strg+Shift+p (Navigieren · Gehe zu · Übereinstimmende eckige Klammer)** hilft in diesem Falle weiter. Man muss hinter der Klammer stehen, damit dieser Menüpunkt funktioniert.

Man nutzt eine Variable und fragt sich, wie sie eigentlich festgelegt wurde, public oder private? Ein Blick in die Outline View hilft nicht, weil dort die Sichtbarkeit gerade nicht angezeigt wird. Ein Klick dorthin bringt einen zur Deklaration, aber auch **F3** oder **Navigieren · Deklaration öffnen**. Dieser Menüpunkt kann aber auch noch mehr. Man ruft eine Methode auf, instanziiert eine Klasse, stets setzt diese Funktion den Mauszeiger in die Deklaration. **Navigieren · Typhierarchie öffnen** ist ein ganz besonderer Menüpunkt. Er hat nämlich eine eigene View, in der er die Ableitung einer Klasse im Vererbungsbaum der Klassen darstellen kann. Will man dagegen den Aufruf im aktuellen Projekt verfolgen, so gelingt das über: **Navigieren · Aufrufhierarchie öffnen**.

Falls eine Methode überschrieben werden soll oder schon ist und man dazu die ursprüngliche Methode konsultieren möchte, so ist das ein Einfaches mit **Navigieren · Übergeordnete Implementierung öffnen**.

Die vorigen Menüpunkte lassen sich am schnellsten über die RMT aufrufen.

Navigieren · Externes Javadoc öffnen kann so genutzt werden, wenn der Mauszeiger innerhalb des Editors steht. Es wird die Javadoc des aktuellen Projektes an der Stelle geöffnet, die auf die aktuelle Klasse oder Methode verweist, in der der Mauszeiger steht. Damit die Funktion verwendet werden kann, muss außerdem die Dokumentation zum Projekt bereits einmal erzeugt worden sein (**Projekt · Javadoc generieren...**).

Mit der folgenden Funktion **Navigieren · Typ öffnen...** kann man den Quellcode eines ausgewählten Typs öffnen. Das funktioniert natürlich nur so weit,

wie der Quellcode verfügbar ist. Was aber in jedem Fall funktioniert, ist die Anzeige im Klassenbaum, die über **Navigieren · Typ in Hierarchie öffnen...** aufgerufen wird.

Die Funktion **Navigieren · Ressource öffnen..** dient dagegen dazu, eine externe Datei zu öffnen.

Will man eine der Methoden oder Objekte aus dem Quellcode in einer der Navigationsansichten suchen, so gelingt das am besten über **Navigieren · Anzeigen in**.

Eine ganz spezielle Navigation bieten **Navigieren · Schnellgliederung** und **Navigieren · Schnelltyphierarchie**. Es ist eine kontextbezogene Suche in einer externen Hierarchiedarstellung. Man kann durch Eingeben der Anfangsbuchstaben die Anzeige entsprechend reduzieren und findet auf diese Weise eine bestimmte Deklaration, selbst in der unüberschaubarsten Datei.

Die nächsten Menüpunkte zählen zu den nützlichsten der gesamten Navigation. Es geht einmal um die Navigation durch die aufgetretenen Fehler. Jetzt kann man zur letzten editierten Position im Quellcode gehen oder eine bestimmte Zeile anspringen. Die Menüpunkte **Navigieren · Zurück · Weiter** durchlaufen die History der Mauszeigerpositionen und sind damit ebenfalls sehr nützlich.

6.4.8 Suchen

Im Gegensatz zu den **Bearbeiten · Suchen**-Funktionen haben diese Suchmethoden nicht nur die aktuelle Datei im Zugriff. Man kann über die Dateien des aktuellen Projektes oder anderer Projekte, aber auch über den ganzen Arbeitsbereich suchen. Selbst eine Suche über die Hilfedateien ist ohne weiteres möglich. Vieles versteht sich hier von selbst, so dass umfangreichere Erklärungen entfallen können. Es lohnt aber in jedem Fall, sich einmal die Zeit zu nehmen und sich all diese Suchfunktionen genauer anzuschauen.

6.4.9 Projekt

Bei dieser kurzen Spalte lässt sich unter **Projekt · Projekt öffnen, Projekt schließen** ein Projekt öffnen oder schließen. Zum Schließen muss das Projekt in der Navigationsansicht ausgewählt sein. Ein geschlossenes Projekt kann in der Navigationsansicht nicht mehr expandiert und auch nicht von anderen Projekten angezogen werden. Das heißt, es ist nicht mehr im »Java Build Path«. Mit **Projekt öffnen** kann es jederzeit wieder geöffnet werden. Der Menüpunkt **Projekt · Projekt erstellen** erzeugt eine neue Runtime-Version des aktuellen Projektes.

Projekt · Alle erstellen wird dagegen alle offenen Projekte kompilieren. Mit **Projekt · Arbeitsset erstellen** lassen sich in Form von Arbeitssets Filter über die Projektdateien legen. Mit **Projekt · Bereinigen...** lassen sich Projekte neu generieren und so Erstellungsfehler bereinigen. Wird **Projekt · Automatisch erstellen** aktiviert, erstellt Eclipse nach jedem Abspeichern die entsprechenden Klassendateien des Projektes neu, so dass beim Ablaufen lassen oft nicht neu kompiliert werden muss.

Mit dem Menüpunkt **Projekt · Javadoc generieren...** lässt sich die Java-Dokumentation des Projektes generieren. Man muss dazu über die Schaltfläche **Konfigurieren...** die Lage der javadoc.exe eingetragen haben. Nach unseren Konventionen sollte sie `C:\Software\Java\jdk1.5.0\bin\javadoc.exe` lauten. Im nächsten Dialog kann der Titel bestimmt und eine ganze Reihe anderer Voreinstellungen festgelegt werden.

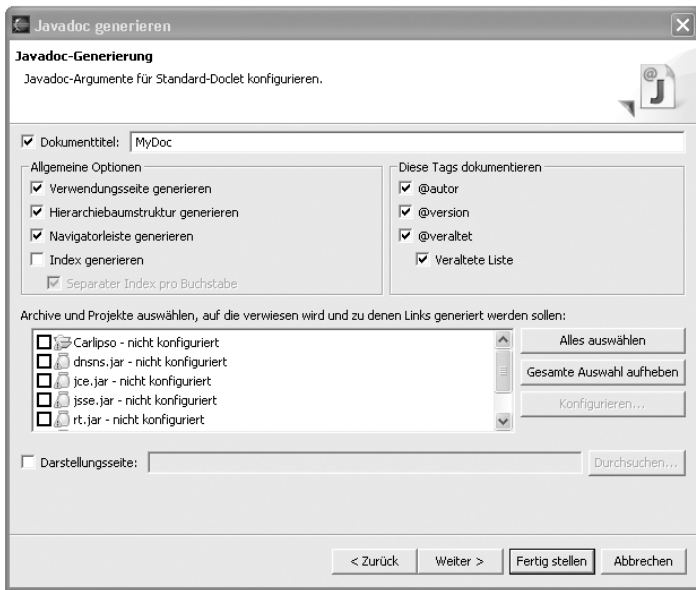


Abbildung 6.18 Javadoc einstellen

Die Generierung nimmt etwas Zeit in Anspruch, danach kann die erzeugte Dokumentation über **Shift+F2** aufgerufen werden.

Die Einstellungen, die beim Einrichten des Projektes vorgenommen wurden, lassen sich per **Projekt · Eigenschaften** ändern und erweitern. Dazu gehören auch Einstellungen zum Projekt, die unter **Fenster · Benutzervorgaben** vorgenommen werden und als Default für alle neuen Projekte dienen. Sie können hier projektspezifisch angepasst werden.

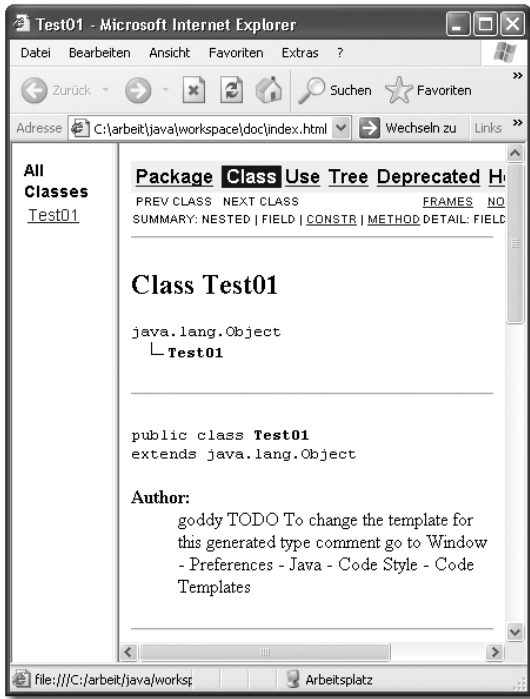


Abbildung 6.19 Javadoc aufrufen

6.4.10 Ausführen

Diese Spalte bietet alle Menüpunkte, die man sich im Rahmen eines Programm- laufs vorstellen kann. Hier geht es nicht nur um den einfachen Start, sondern auch ums Debugging. Zunächst einmal lässt sich der letzte Lauf oder Debug- Aufruf wiederholen. Die Funktionen im Menü heißen:

- ▶ **Ausführen · Zuletzt gestartetes Objekt ausführen**
- ▶ **Ausführen · Debug für zuletzt gestartetes Objekt**

Im Weiteren findet sich jeweils ein Menüpunkt als **Ausführungsprotokoll**, als **Ausführen...**, wodurch die große Dialogbox geöffnet wird und man eine neue Run-Konfiguration anlegen kann. Eine Auswahl aus bestehenden Konfigurationen ist ebenfalls möglich. Bei **Ausführen als...** lässt sich auswählen, wie das aktuelle Projekt gestartet werden soll. Das ist gut, weil sich Java-Programme schreiben lassen, die sowohl als Applet als auch als Applikation laufen. Setzt man Junit ein, ergibt sich eine weitere Möglichkeit, eben der Testlauf. Die **Laufzeit-Workbench** wird genutzt, um Plugins zu testen. Es folgen eine Reihe Menüpunkte zum Debugging, teilweise deaktiviert. Breakpoints lassen sich natürlich auch setzen, wenn das Debugging nicht aktiv ist. In der Debugging-

Perspektive ist die Run-Menüspalte durch die Debug-Befehle **Wieder aufnehmen, Step-Over, Ausführen bis Zeile** usw. erweitert.

Der letzte Menüpunkt **Externe Tools...** betrifft den Start von externen Programmen. Bei einer Standardinstallation sollte es möglich sein, hier die Ant-Dateien zu starten.

6.4.11 Restliche Menüs





Die Menüpunkte der **Fenster**-Spalte sind zum großen Teil bereits besprochen worden, wo dies nicht der Fall ist, verstehen sie sich von selbst. Der größte Teil sind Funktionen zu den Perspektiven und der wichtigste Punkt ist eindeutig **Fenster · Benutzervorgaben**, wo die gesamten Properties-Einstellungen von Eclipse zusammengefasst sind.


Die **Hilfe**-Spalte beinhaltet nicht nur den Start der Online-Hilfe, sondern auch Update-Funktionen. So kann man beispielsweise mit **Hilfe · Software-Updates · Suchen und installieren** im Internet nachschauen lassen, ob ein neues Update für Eclipse oder für installierte Plugins bereitstehen. Daneben gibt es einen Konfigurations-Manager **Hilfe · Software-Updates · Konfiguration verwalten...**, der neben einem Installationsprotokoll die Updates aller Plugins verwaltet und mit dem man gezielt nach Updates suchen kann. Nähere Informationen zu den installierten Plugins kann man auch unter **Hilfe · Info über Eclipse-Plattform** finden.

6.5 Die Views

6.5.1 Die Navigator-View

Was zur Navigator-View zu sagen ist, gilt im Grunde genauso für den Paket-Explorer. Sie besitzt eine Kopfleiste mit Icons, ein Kontextmenü auf der RMT.

Symbol	Bedeutung
	Zurück (Back) Blättert in den Ansichten zurück
	Weiter (Forward) Blättert in den Ansichten vorwärts
	Nach oben (Up) Geht eine Stufe nach oben
	Alle Ebenen ausblenden (Collapse all) Minimiert die Baumansicht

Symbol	Bedeutung
	Mit Editor verlinken (Link with Editor) Synchronisiert Baum und Editor auf gleiche Objekte

Die ersten Icons dienen der Navigation innerhalb des Baumes. Wobei es nicht darum geht, im Baum auf- und abzustiegen, sondern darum, auf Ausschnitte zu reduzieren. Mit dem Menübefehl **RMT · Gehe in** gelangt man in das aktuelle markierte Verzeichnis, wobei der Baum oberhalb ausgeblendet wird. Mit dem Zurück-Icon kommt man wieder zurück in die Ansicht des Gesamtbaumes.

Mit **Zurück** und **Weiter** kann man innerhalb dieser verschiedenen Teilbaumansichten blättern.

Mit **Alle Ebenen ausblenden** lässt sich der Baum vollständig minimieren, so dass auch alle Unterverzeichnisse geschlossen sind.

Interessant wird es mit der Funktion **Mit Editor verlinken**. Meist hat man nämlich mehrere Dateien im Editor und wechselt über die Reiter von einer zur anderen. Hat man **Mit Editor verlinken** aktiviert, kann man die Datei im Editor in den Vordergrund setzen, indem man sie im Navigator anklickt. Da man sich im Navigator besser orientieren kann, ist dies eine Erleichterung. Das Menü das sich hinter dem Pfeil versteckt, bietet neben dem Kontextmenü weitere interessante Funktionen. Arbeitssets hatten wir bereits kurz angesprochen. Hier lassen sie sich aus- und abwählen sowie erstellen. Direkt unter den Funktionen baut Eclipse eine Liste der bereits genutzten Arbeitssets für die schnelle Anwahl auf.

Eine einfache Sortierung nach Typ oder Name ist mit dem Sortieren-Menüpunkt möglich. Während Arbeitssets eine gezielte Auswahl in jeder Ebene mit Ausnahme der einzelnen Dateien zulassen, kann man mit **Filter...** nach einem Dateityp filtern lassen.

Wenn Sie ein anderes Arbeitsset auswählen, den Navigationsbaum unberührt lassen und die RMT drücken, bekommen Sie ein kleines Kontextmenü zu sehen, das nur sehr wenig Menüpunkte aufweist. Über **Neu** können Sie alles Mögliche neu erstellen, **Importieren/Exportieren** sind bekannt. So bleibt nur noch **Aktualisieren**. Falls ein CVS genutzt wird, gleicht Eclipse beim Anklicken von **Aktualisieren** oder **F5** die externen Dateien mit der Workbench ab. Es wird jedoch nicht kontrolliert, ob eine Datei im Editor noch nicht gespeichert wurde. In erster Linie aber dient die **Aktualisieren**-Funktion dazu, die Dateien, die mit einem externen Programm bearbeitet wurden, mit den Dateien in Eclipse abzugleichen.

Wenn ein Projekt, ein Verzeichnis oder eine Datei im Navigatorbaum ausgewählt ist, sieht das RMT-Menü natürlich wieder anders aus. Jetzt ist es ein echtes Kontextmenü und bietet entsprechend viele Funktionen an. Allerdings unterscheiden sie sich etwas, je nachdem was ausgewählt wurde.

Die ersten Punkte sind mit Sicherheit geläufig und brauchen nicht erläutert zu werden.

Mit den **Importieren/Exportieren**-Funktionen lassen sich eine ganze Reihe von externen Dateien in die Workbench übernehmen oder Dateien erzeugen. Beim Export wird stets ein Dialog angezeigt, in dem das Projekt oder Teile davon zur Auswahl angezeigt werden.

Aktualisieren (F5) ist bereits besprochen worden.

Der Menüpunkt **Team** enthält eine Reihe von Funktionen zum Einsatz eines CVS.

Der Menüpunkt **Vergleichen mit** ist dagegen unabhängig von einem CVS und betrifft das **lokale Protokoll** (lokale Historie).

Diese Historie über die aktuelle Entwicklerarbeit wird entsprechend den Einstellungen unter **Fenster · Benutzervorgaben · Workbench · Lokales Protokoll** geführt (Abbildung 6.20). Dort gibt es eine Einstellung über die Länge der Historie in Tagen, die Anzahl der Einträge pro Datei und den Speicherplatz, den die Historie insgesamt beanspruchen darf. Mehr über die lokale Historie (und zur Funktion **Replace with...**) erfahren Sie im folgenden Kapitel zu den Editoren.

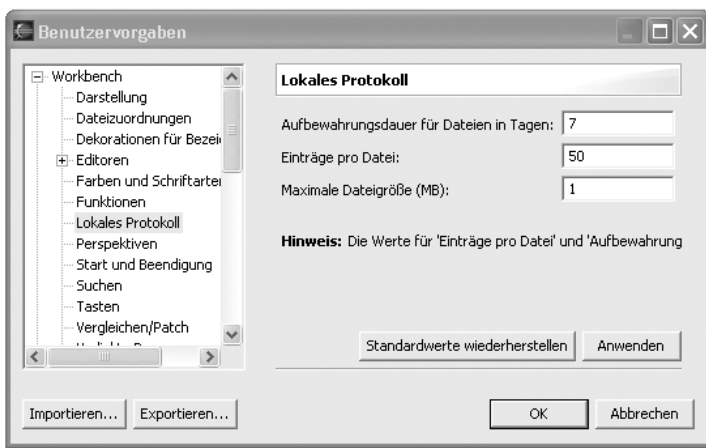


Abbildung 6.20 Einstellung zum lokalen Protokoll

Als letzten Punkt in der Spalte gibt es noch den Menüpunkt **Eigenschaften**. Hiermit lassen sich beispielsweise Dateien auf **Schreibgeschützt** setzen und Informationen zu den Dateien und Verzeichnissen einsehen.

6.5.2 Die Editoren

Die Kontrolle der Teilfenster, aus denen sich eine Eclipse-Oberfläche aufbaut, liegt zunächst bei der Navigations-View. Sie bestimmt, was der Editor anzeigt und dieser wiederum, was die Outline View und die Task View zeigen. Objekte, die im Paket-Explorer angezeigt werden, öffnen sich durch Doppelklick. Welches Programm sich öffnet ist durchaus nicht fest vorgegeben. Mit einem Rechtsklick kann man entscheiden, ob mit dem Systemeditor, das ist der Editor mit dem der Dateityp vom Betriebssystem her verknüpft ist, oder mit einem anderen Editor geöffnet wird (Menüpunkt: **Öffnen mit**).

Beispielsweise könnte sich der Effekt zeigen, dass XML-Dateien mit dem MS-Explorer geöffnet werden, weil er standardmäßig für die Dateierdung **.xml** verknüpft ist. Möchten Sie XML-Dateien jedoch direkt in Eclipse editieren, gehen Sie folgendermaßen vor:

Fenster · Benutzervorgaben · Workbench · Dateizuordnungen Nutzen Sie die obere **Hinzufügen**-Schaltfläche, um die Dateierdung **.xml** neu aufzunehmen. Weisen Sie mit der unteren **Hinzufügen**-Schaltfläche der Dateierdung den **Texteditor** zu. Nach dem Schließen des Dialoges mit **OK** sollte ein Doppelklick auf eine XML-Datei im **Paket-Explorer** die Datei mit dem Eclipse-Texteditor öffnen.

Tastenkürzel Tastaturbefehle erlauben ein zügigeres Arbeiten als das gewöhnliche Mäuserennen und Rumgeklicke. Sollte man zur Gattung der Tastaturmuffel gehören, bietet sich bei Eclipse die einmalige Gelegenheit einen flotteren Arbeitstil zu erlernen. Falls man Tastaturkürzel von Emacs kennt, lässt sich Eclipse durch einfaches Umschalten der so genannten Tastenkürzel (Keybindings) in einen äußerst komfortablen Emacs verwandeln.

Die wichtigsten Tastaturbefehle hatten wir bereits zu Beginn des Kapitels besprochen (Tabelle 6.1). Die Bedienung muss sich nicht auf diese Tastaturkürzel beschränken, man kann jederzeit eigene Tastenkürzel einfügen.

Fenster · Benutzervorgaben · Workbench · Tasten Die große Dialogbox erlaubt zunächst einmal die Anwahl der aktiven Konfiguration. Es lässt sich zwischen **Standardwert** und **Emacs** umschalten. Im Befehlfenster werden bei **Kategorie** Funktionsgruppen angezeigt, wie **Bearbeiten**, **CVS**, **Datei**, **Fenster** oder **Hilfe**. Man findet neben **Name** die verfügbaren Funktionen. Gibt es dort eine Tastenkürzelzuordnung wird sie in dem Feld **Zuordnungen** rechts dane-

ben angezeigt. Vorhandene Kürzel lassen sich auswählen und ändern. Man sollte jedoch bedenken, dass bei individuellen Einstellungen an einem Rechner die Bedienung an dem Rechner eines anderen Nutzers schwerer wird. Daher sollte man Teamweit die gleichen Kürzel nutzen. Neue Kürzel werden einfach durch Auswahl der Werte in den Optionfeldern erstellt. Mit der Schaltfläche **Hinzufügen** werden sie zugeordnet. Es wird angezeigt, welchem Kommando dieses Tastenkürzel bereits zugeordnet ist.

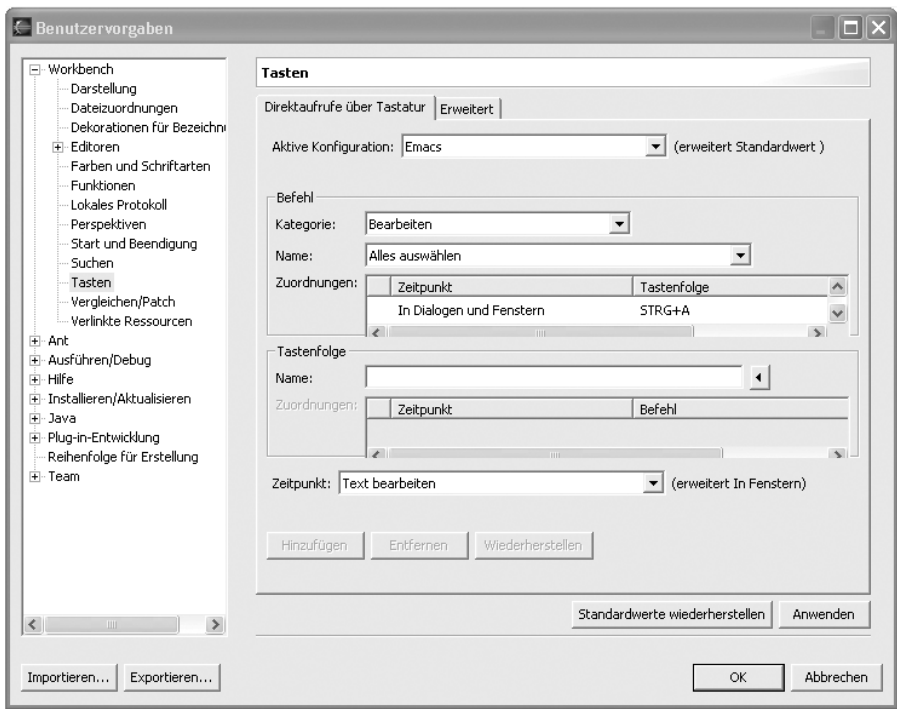


Abbildung 6.21 Einstellung der Tastenkürzel

Der Codeformatter

Es erübrigt sich, zu behaupten, gut lesbarer Code vereinfache die Arbeit. Man sollte, falls es irgendwie möglich ist, ein genau definiertes Styleguide für den Quellcode aufsetzen. (Wir werden in einem späteren Kapitel noch genauer darauf zu sprechen kommen.) Vor allem aber sollte man sich daran halten. Eclipse unterstützt dieses Unterfangen, so weit überhaupt automatisiert möglich, mit seinem Codeformatter. Sie finden ihn über **Fenster • Benutzervorgaben** im Baum mit **Java • Codedarstellung • Codeformatierungsprogramm**.

Wenn wir auf die Codekonventionen zu sprechen kommen, werden wir auch diese Benutzervorgaben genauer durchsprechen. Im Moment wollen wir uns

lediglich einen Überblick verschaffen. In der Dialogbox gibt es unter **Anzeigen...** die Registerkarten:

- ▶ Einrückung
- ▶ Geschweifte Klammern
- ▶ Leerzeichen
- ▶ Leerzeilen
- ▶ Neue Zeilen
- ▶ Steueranweisungen
- ▶ Zeilenumbruch
- ▶ Kommentare

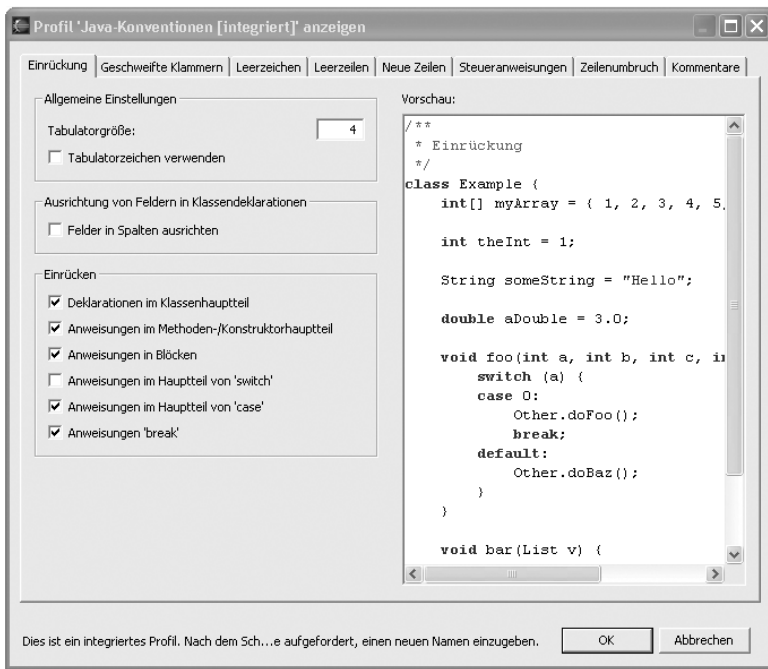


Abbildung 6.22 Einstellungen zur Codeformatierung

Ein Textfeld bietet bei allen Registerkarten die Möglichkeit, Änderungen an den Einstellungen direkt zu beurteilen.

In **Zeilenumbruch** wird beispielsweise lediglich festgelegt, wie lang die Zeilen sein sollen. Stellen Sie beispielsweise die Zeilenklänge auf 10 ein, so sehen Sie sofort die durchgeführten Umbrüche. Machen Sie die Zeilen nicht zu kurz, das verschlechtert die Lesbarkeit. Durch die Möglichkeit, jedes Teilfenster mit Dop-

pelklick auf die Kopfleiste zu maximieren, ist man nicht auf kurze Zeilen angewiesen. Es ist sogar angebracht, die Zeilenlänge relativ groß zu machen, um gut lesbaren Code zu erhalten. Beispielsweise ist 200 bis 400 ganz praktisch. Perfektionisten sollten feststellen, wie viel Zeichen bei ihrer Lieblingsauflösung beim maximierten Editor in eine Zeile passen und diesen Wert einstellen.

Neue Zeilen bringt mit seinen Einstellungen die meisten Änderungen. Auch hier sieht man sehr schön, was sich ändert. Was man einstellt, bleibt jedem selbst überlassen, weil Eclipse anbietet, über den Menüpunkt **Quelle · Formatieren** oder **Strg+Shift+F** die aktuelle Datei jederzeit umzuformatieren.

Auf der Registerkarte **Geschweifte Klammern** und **Steueranweisungen** wäre es gut, diejenigen Einstellungen vorzunehmen, die die geschweiften Klammern untereinander legt. Hierzu muss man fast alles aktivieren.

Auf der Registerkarte **Kommentare** kann man die Formatierung der Kommentare ganz abschalten oder nach bestimmten Gesichtspunkten (Html-Tags, Header, Leerzeilen) gestalten.

Nachdem der Codeformatter Ihren Vorstellungen gemäß eingestellt ist, steht er Ihnen jederzeit per **Strg+Shift+F** zur Verfügung.

Probleme mit der Codequalität

Falls Sie ein Projekt übernehmen und der Vorgänger unsauber programmiert hat, überschüttet Eclipse Sie womöglich mit Warnungen und Fehlermeldungen. Deshalb gibt es die Möglichkeit, über **Fenster · Benutzervorgaben** im Baum **Java · Compiler** auszuwählen und auf den Registerkarten **Darstellung** und **Erweitert** Meldungen für die bestimmten Problemgruppen zu aktivieren oder zu deaktivieren. Zunächst einmal sollte man versuchen, eine lauffähige Version zu erhalten. Dazu kann man die Meldungen alle deaktivieren und beginnend mit **Nicht verwendeter Code** Problemgruppe für Problemgruppe aktivieren, um sie ausmerzen zu können. So kann man sich langsam an die gewünschte Codequalität heranarbeiten. So kann die Qualität des eigenen Codes nach und nach verbessert werden.

Fehler

Fehler sind am linken Rand mit einem roten Kreis markiert. Von Fehler zu Fehler kann man mit den Tasten **Strg+** sowie **Strg+,** springen. Ist ein Fehler behoben worden, wandelt sich die Markierung in einen grünen Kreis. Hier können die ehemaligen Fehler weiterhin angesprungen und kontrolliert werden. Erst nach einem Abspeichern oder Kompilierprozess werden die Markierungen nicht mehr angezeigt.

Die lokale Historie (Protokoll)

Hin und wieder stellt man fest, dass die Änderungen nicht den gewünschten Erfolg bringen und man lieber den älteren Versionsstand wieder nutzen möchte. Nun versucht man sich zu erinnern ... Selbstverständlich wäre es besser gewesen, den alten Zustand irgendwo zwischenzuspeichern. Was man selbst versäumt hat, Eclipse hat es gemacht.

Dafür gibt es das so genannte »Lokale Protokoll« (nicht gerade eine gelungene Übertragung ins Deutsche. **Historie** statt **Protokoll** wäre passender). Man markiert den Codeteil (ganz wichtig: Ohne Markierung funktioniert es nicht. Es geht auch nicht, wenn es innerhalb des markierten Codes noch nie Änderungen gab), in dem die Änderungen vorgenommen wurden und nutzt die **RMT · Lokales Protokoll · Vergleichen mit...** (Abbildung 6.23).

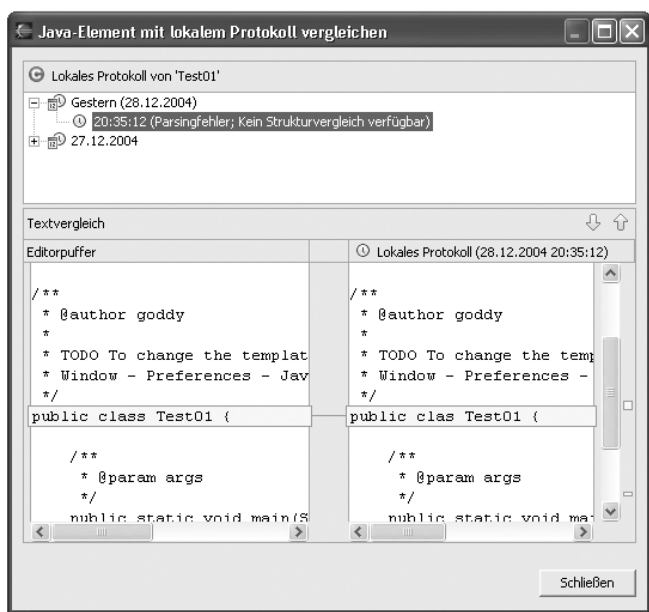


Abbildung 6.23 Textvergleich im lokalen Protokoll

In der Dialogbox werden die alten, gespeicherten Versionen in Form eines Baumes mit Tag und Uhrzeit angezeigt. Der aktuelle Zustand wird im linken Fenster gezeigt. Falls die zuletzt gemachte Änderung unbrauchbar war, kann der Code direkt durch den vorigen Stand ersetzt werden. Das gelingt mittels **RMT · Lokales Protokoll · Durch vorheriges ersetzen**. Möchte man nach dem Austausch den früheren Stand wieder haben, so gelingt das mit dem Rückgängigbefehl (**Bearbeiten · Rückgängig** oder **Strg-Z**), aber auch, indem man das **Durch vorheriges ersetzen** im **Lokalen Protokoll** erneut aufruft. Da der letzte Ersetzen-

vorgang ebenfalls in der History festgehalten ist, kann er auch so wieder rückgängig gemacht werden.

Es kann auch gezielt mit einer bestimmten Version ausgetauscht werden. Dazu dient die Funktion:

RMT · Lokales Protokoll · Ersetzen durch... Wobei aus dem Baum der gespeicherten Versionen die gewünschte Version ausgewählt werden kann. Das Programm zeigt den jeweiligen Zustand in der Gegenüberstellung an. Es gibt noch eine weitere Möglichkeit in der Reihe der Protokoll-Funktionen. Wenn Sie **RMT · Lokales Protokoll · Wiederherstellen von...** aufrufen, bietet Ihnen Eclipse einen Überblick der verworfenen Methoden der aktuellen Klasse an. Das sind die Methoden, die irgendwann aus der Klasse gelöscht wurden. Mit dieser Funktion lassen sie sich wieder reaktivieren.

Ein Teil der **Lokales Protokoll**-Funktionen kann auch mit der RMT direkt auf der Klasse (Datei) aufgerufen werden.

Die Kontext-Info

Steht der Mauszeiger auf einem Objektnamen oder auf einer Methode, wird nach einer Sekunde die genaue qualifizierte Bezeichnung mit Typen in einem gelben Kontextmenü angezeigt. Bei Klassen wird außerdem der Javadoc-Kommentar gezeigt. Ein überzeugender Grund dafür, den Code entsprechend sinnvoll zu kommentieren.

Hält man die Strg-Taste gedrückt, während man den Mauszeiger auf einen Namen setzt, zeigt die Kontext-Info bei Klassen die ersten fünf Zeilen des Codes einer Klasse an. Außerdem funktioniert der Name wie ein Link, der mit der Maus angeklickt werden kann.

Interessant am Editor ist die darin befindliche Funktion **Lesezeichen hinzufügen....** Zunächst einmal ist zu sagen, dass man Lesezeichen in erster Linie im Editor setzen kann, sie werden jedoch im Repository mit abgespeichert und können jederzeit wieder eingesehen werden. Im Editor befindet sich an der linken Seite ein graues Band, auf dem die RMT ein Menü mit dem Punkt **Lesezeichen hinzufügen...** öffnet. Also kann hier gezielt ein Lesezeichen an eine Zeile gehängt werden. Um die Lesezeichen selbst zu verwalten, öffnet man über **Fenster · Sicht anzeigen · Andere... · Basis · Lesezeichen**.

Die Gliederungs-View

Die Gliederungs-View bietet einen sehr gelungenen Überblick über den Quellcode und seine Bestandteile. Es gibt eine Reihe von Symbolen für die Bestandteile des Quellcodes. In der Kopfleiste sind folgende Symbole angeordnet:







Symbol	Bedeutung
	Sortieren (Sort) Sortiert die Elemente alphabetisch
	Felder ausblenden (Hide Fields) Die Funktion blendet alle Instanzvariablen aus.
	Statische Felder und Methoden ausblenden (Hide Static Members). Blendet alle statischen Elemente aus
	Nicht öffentliche Member ausblenden (Hide Non-Public Members) Alle Elemente mit eingeschränkter Sichtbarkeit werden ausgeblendet. Dadurch bleibt nur noch die Schnittstelle der Klasse sichtbar.
	Lokale Typen ausblenden (Hide local Types) Alles, was lokal ist, wird nicht mehr angezeigt.
	Mit Editor verlinken (Link with Editor) Setzt im Editor das aktuelle Element aus dem Outline View in den Vordergrund.

Table 6.2 Die Symbolleiste der Gliederungs-View

Innerhalb der Outline View gibt es natürlich auch Kontextmenüs, deren Funktionalität aber im Wesentlichen dem bisher besprochenen entspricht.

Eine sehr nützliche Funktionalität im Zusammenhang mit der **Gliederungs-View** ist die Funktion **Nur Quelle des ausgewählten Elementes anzeigen**, die hinter diesem Symbol versteckt ist:



Wählt man in der **Gliederung** ein Element an, ob Variable oder ganze Klasse ist egal, und ruft diese Funktion auf, wird im Editor alles, bis auf dieses Element, ausgeblendet. So ist übersichtliches Arbeiten gewährleistet.

Die Task View (Bookmarks – und Error View)

Die Task View ist eine einfache Liste, die Sortier- und Filterfunktionalität aufweist (Abbildung 6.24). Mit der RMT-Taste öffnet sich ein Kontextmenü, worin es Punkte gibt, um Eintragungen als erledigt zu markieren. Erledigte Einträge lassen sich auf einmal löschen. Natürlich kann man zu den Einträgen im Quellcode springen sowie neue Einträge aufnehmen.

Die Lesezeichen-View zeigt tabellarisch alle gesetzten Lesezeichen an. Eine Zeile besteht dabei aus dem Lesezeichentext, der Datei und dem Ordner, in dem die Datei liegt; auch wird die Zeile angegeben, in der das Lesezeichen gesetzt ist.

Beschreibung	Ressource	Im Ordner	Position
TODO To change the template for this ...	Test01.java	Carlipso/src	Zeile 4
TODO To change the template for this ...	Test01.java	Carlipso/src	Zeile 11

Abbildung 6.24 Task View

Die Fehler-View ist ähnlich aufgebaut. Sie zeigt eine genaue Beschreibung des Fehlers, es folgen der Dateiname und das Verzeichnis auf die Zeile.

Die Symbolleisten (Toolbars)

Die Symbolleisten liegen unterhalb der Menüleiste und sind frei anzuordnen. Sie enthalten Funktionen, die genauso in den Menüs vorkommen. Von daher sind sie nicht unabdingbar, sondern stellen lediglich eine Arbeiterleichterung dar. Als Anwender sollte man darauf achten, dass auch nur solche Toolbars eingebunden sind, die man wirklich benötigt.

Mit **RMT · Symbolleisten sperren** lassen sich die Symbolleisten fixieren, das heißt, sie lassen sich nun nicht mehr verschieben. Mit der **RMT · Perspektive anpassen...** kann man sie für die jeweilige Perspektive aus- oder abwählen.

Die Registerleisten

Registerleisten werden immer sichtbar, wenn mehrere Views übereinander gelegt werden. Mit ihrer Hilfe kann man in den Views blättern.

Die Statusbar

Die Statusbar bildet den unteren Rand der Workbench. Es ist eine übliche Statusbar, in der die Position im Text in Zeilen und Spalten angezeigt wird, wenn der Mauszeiger in einem Editor steht. Steht er in einer anderen View, wird hier der Name des markierten Elementes angezeigt.

6.6 Projekte übernehmen

6.6.1 Bestehende Projekte und Dateien in ein Eclipse-Projekt übernehmen

Wie soll man vorgehen, wenn man die Beispiele von der CD-ROM oder aus dem Internet in Eclipse zum Laufen bringen möchte? Oder wie übernimmt man Projekte aus einer anderen IDE, z. B. dem JBuilder von Borland?

Unter Umständen gibt es in der Firma sogar ältere JDK-Projekte, die mit einem einfachen Editor erstellt wurden?

6.6.2 JDK-Projekte und einzelne Dateien übernehmen

Falls man irgendwo Dateien liegen hat, die in einem Eclipse-Projekt von Bedeutung sind, lassen sie sich einfach in die Eclipse-Workbench übernehmen. Man markiert die Java-Datei im Windows-Datei-Explorer und kopiert über **RMT · Kopieren** und wechselt in Eclipse auf das Verzeichnis im Paket-Explorer, wo die Datei eingefügt werden soll: **RMT und Einfügen**. Gegebenenfalls muss in der Datei die Paket-Zeile angepasst und die Imports der Datei zu den Bibliotheken hinzugefügt werden.

Falls ein ganzes Paket aus dem Projekt einer anderen IDE übernommen wird, geht man ähnlich vor. Das Verzeichnis wird mittels des Datei-Explorers in das Eclipse-Projekt kopiert. Anpassungen sind unter Umständen gar nicht erforderlich. Zum Starten benötigt man lediglich einen neuen Eintrag unter **Ausführen als Java Anwendung**.

6.6.3 Ganze Projekte von fremden IDEs übernehmen

Das man ganze Projekte von anderen IDEs problemlos nach Eclipse übernehmen kann (z.B. von JBuilder – siehe Abbildung 6.25) fördert die Produktivität. So lassen sich beispielsweise Swing-Projekte sehr einfach mit dem JBuilder realisieren; für Swing findet man bei Eclipse bisher wenig Plugins. Soll ein ganzes Projekt übernommen werden, wenn kein relevantes Projekt in Eclipse existiert, geht man folgendermaßen vor:

Datei · Neu · Projekt... · Java · Java Projekt Einen Namen für das Projekt festlegen und das Projekt erzeugen. Wichtig ist vor allem das Sourcecode- und das Output-Verzeichnis (/src und /bin).

Im Menü **Datei · Import... · Dateisystem auswählen**. Im **Aus Verzeichnis:-** Feld wird mit der **Durchsuchen**-Schaltfläche der Ordner ausgewählt, der das Projekt beinhaltet. Die Übersicht zeigt den Inhalt des Projektes als Baumstruktur an.

Es werden die Unterverzeichnisse ausgewählt, die für das neue Eclipse-Projekt von Bedeutung sind. Ins Feld **In Ordner** wird mittels der **Durchsuchen...** - Schaltfläche das eben erstellte Projekt eingestellt. Je nach Aufbau kann direkt das Projekt oder das SRC-Verzeichnis gewählt werden (Abbildung 6.26).

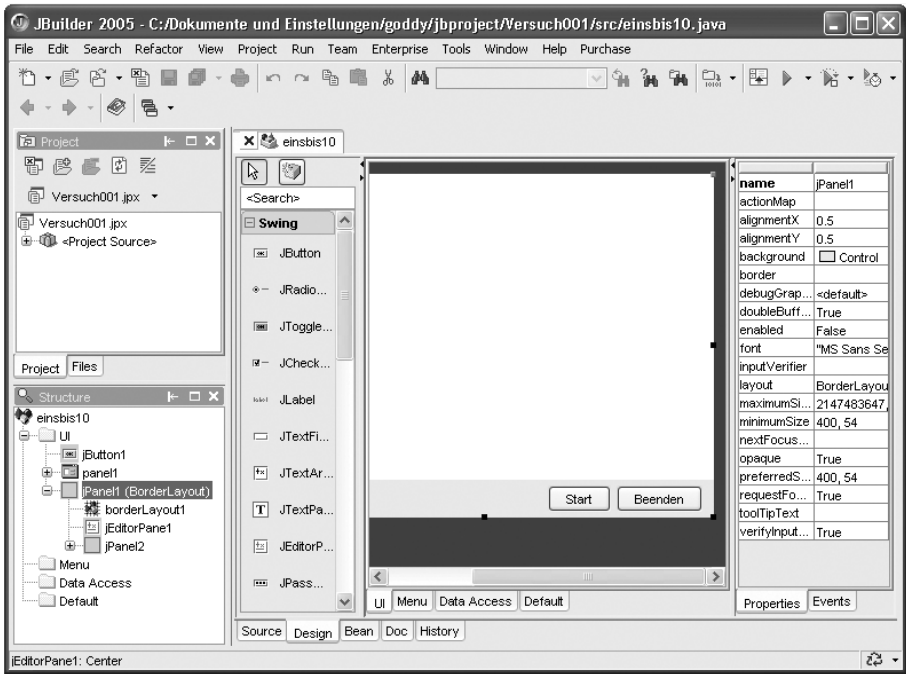


Abbildung 6.25 Swing-Projekt im JBuilder

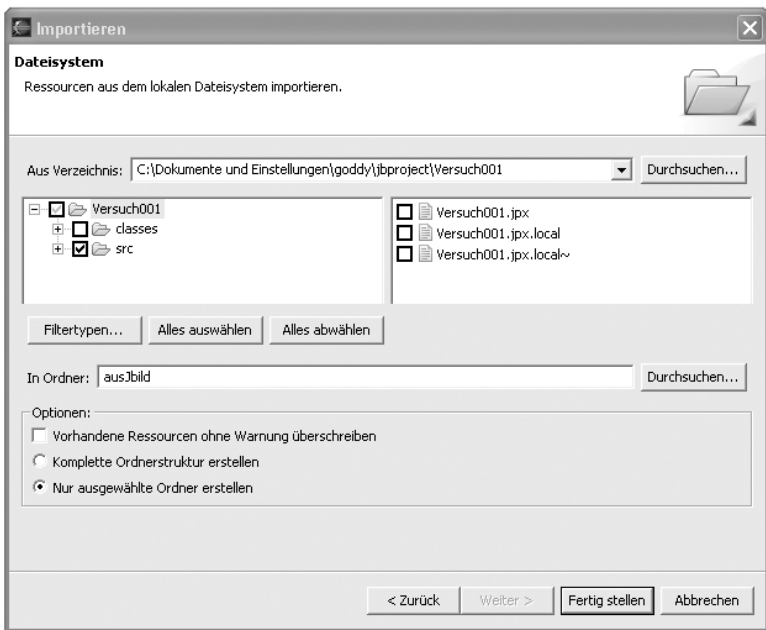


Abbildung 6.26 Projekt importieren

Sollte der Verzeichnisbaum des alten Projektes falsch übernommen worden sein, so kann er an beliebiger Stelle genommen und in der Paketstruktur verschoben werden. Die verwaisten Strukturen werden danach einfach gelöscht.

6.6.4 Andere Eclipse-Projekte übernehmen

In diesem Fall ist es nicht erforderlich, ein neues Projekt anzulegen. Wichtig ist, dass im Quellprojekt-Ordner eine Datei **.project** existiert. Eclipse entnimmt daraus die projektspezifischen Einstellungen und legt das Projekt mit allen Verzeichnissen und Metadaten neu an. Insbesondere die Projekte von der CD-ROM des Buches können Sie auf diese Weise auf Ihren Rechner übertragen.

RMT im Paket-Explorer oder der **Navigationsicht** • **Importieren...** • **Vorhandenes Projekt in den Arbeitsbereich**. Mit der Durchsuchen-Schaltfläche das temporäre Verzeichnis mit dem Projekt auswählen. Der Projektname sollte automatisch angezeigt werden. Nachdem der Vorgang abgeschlossen ist, ist das Projekt voll verwendbar, ohne das man zusätzliche Einstellungen vornehmen muss. Die Import/Export-Funktion bietet eine Reihe weiterer Dateiformate an. So lassen sich JAR-Dateien erzeugen oder Javadocs exportieren. Mit **Komprimierte Datei (Zip-File)** lassen sich beispielsweise gepackte Dateien direkt in ein Arbeitsbereichsverzeichnisse entpacken.

6.6.5 Änderungen an der Paketstruktur bestehender Projekte

Im Zusammenhang mit der Übernahme von bestehenden Projekten oder Teilen ist es oft erforderlich, Klassen aus Paketen zu entnehmen oder innerhalb der Struktur zu verschieben. Das gelingt ganz gut im Paket-Explorer durch Anklicken und Verschieben (Drag-and-Drop). Es funktioniert aber auch mit **RMT** • **Refactoring** • **Versetzen...** Man braucht in der folgenden Dialogbox lediglich das Zielverzeichnis auszuwählen. Mit der Schaltfläche **Neu...** kann aber auch ein neues Verzeichnis angelegt werden.

Neben einer Verschiebung ist das Umbenennen ein häufiger Vorgang bei der Übernahme von anderen Projekten. Es kann nicht direkt durchgeführt werden, sondern es geht über **RMT** • **Refactoring** • **Umbenennen...** Der neue Name wird im entsprechenden Feld eingegeben. Sollen alle Verweise auf ihn ebenfalls geändert werden, was fast immer ratsam ist, lässt man **Verweise aktualisieren** aktiv. Die Voranzeige zeigt alle anstehenden Änderungen an, bevor man sie durchführen lässt (Abbildung 6.27).

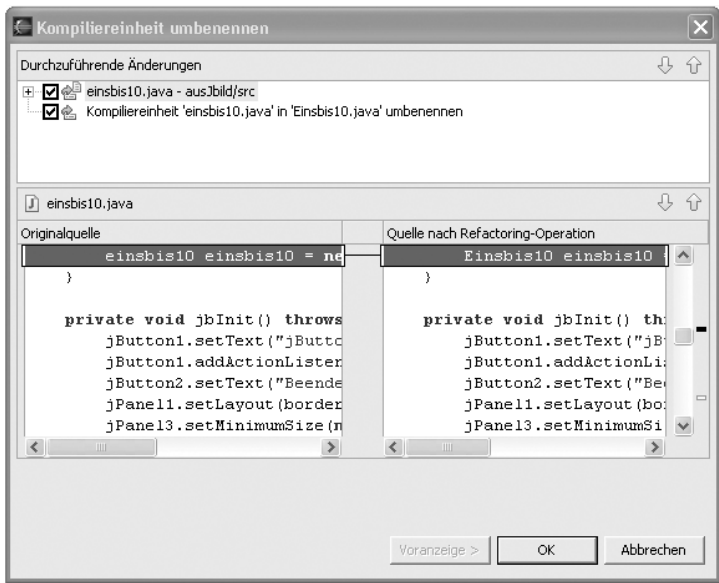


Abbildung 6.27 Voransicht beim Umbenennen

6.7 Zusammenfassung

Eclipse hat eine kaum überschaubare Vielfalt an Oberflächenelementen. Dieses Kapitel sollte einen Überblick und Hinweise für die Bedienung der einzelnen Bausteine geben. Zunächst haben wir uns eine Tabelle der Tastenkürzel angesehen. Tastenkürzel sind die schnellste Art Eclipse zu bedienen, deshalb sollte man sich möglichst früh daran gewöhnen.

Eclipse besteht aus Perspektiven, die sich aus Sichten und Editoren aufbauen. Die Anordnung der Sichten ist nicht willkürlich, sondern sie werden um einen Editor angeordnet, der in der Mitte positioniert ist.

Die erste View, die wir uns genauer angesehen haben, war die so genannte Scrapbookseite. Es ist eine Art Editor. Er ist der Sprache Smalltalk nachempfunden. Dieser Editor schafft es einzelne Anweisungen auszuführen, ganz wie bei einem Interpreter. Diese Eigenschaft macht ihn interessant, wenn man Java lernen möchte.

Eclipse kann man mehrmals öffnen. Das ist besonders an einem Arbeitsplatz interessant, der über zwei Monitore verfügt. Wir haben ein Projekt angelegt, um zu sehen, welche Möglichkeiten Eclipse bietet Projekte zu verwalten. Später haben wir uns angesehen, wie man eine Perspektive nach eigenen Vorstellungen abändert und wie die einzelnen Bestandteile der Perspektiven verwendet werden.

6.8 Aufgaben zum Kapitel

Theorie

1. Aus welchen grundlegenden Bestandteilen ist die Benutzeroberfläche von Eclipse aufgebaut?
2. Wie werden Sichten in Eclipse angeordnet?
3. Wie geht man vor, wenn man die Konsolen-Sicht öffnen möchte, die im Moment in Eclipse nicht sichtbar ist?
4. Was hat es mit der **Scrapbookseite** auf sich?
5. Startet man Eclipse ein zweites Mal, bekommt man die Meldung, dass Eclipse nicht zweimal gestartet werden kann. Wie ist es trotzdem möglich, mit zwei Eclipse-Instanzen zu arbeiten?
6. Welche Menüs gibt es in der Menüleiste?
7. Welchen Nutzen haben Arbeitssets bei Eclipse?
8. Nenne wenigstens vier Details (Bedienelemente, Views o.Ä.) der Eclipse-Oberfläche.
9. Was ist der Unterschied zwischen **Bearbeiten · Suchen und Suchen · Suchen...**?
10. Welche Möglichkeiten gibt es mit Eclipse andere Projekte zu übernehmen?

Praxis

Auf einem kleineren Monitor sind die vielen Views der üblichen Plugins verwirrend. Erstellen Sie für die Programmierfähigkeit eine Perspektive, die in erster Linie aus dem Java-Editor besteht. Auf der rechten Seite sollen der »Paket-Explorer« und die »Outline View« übereinander liegen. Insbesondere soll es am unteren Rand keine Konsole oder eine Task View geben. Die Console wird immer automatisch geöffnet, wenn sie durch eine Ausgabe (Fehlermeldung oder Sysout) angesprochen wird.

6.9 Webseiten zum Kapitel

URL	Beschreibung
http://www.3plus4software.de/eclipse/	Tutorials
http://de.wikipedia.org/wiki/Eclipse_(IDE)	Lexikoneintrag zu Eclipse
http://eclipsewiki.editme.com/	Community-Seite
http://www.eclipseproject.de/	Deutsche Seite

URL	Beschreibung
http://www.eclipse-magazin.de	Eclipse Zeitschrift
http://www.oneclipse.com	Umgang mit Eclipse lernen (engl.)
http://www.myeclipseide.com/	IDE mit Eclipse
http://www.visual-paradigm.com/	Werkzeuge für OOP
http://www.microtool.de/objectif/de/prod_eclipse.asp	Deutsches UML-Werkzeug für Eclipse
http://www.java-tutor.com/java/eclipse-plugins/plugin.html	Entwicklungstools Eclipse
http://www.eclipseproject.de/	Allgemeine Infos zu Eclipse
http://www.oio.de	OOP- Competence Center

12 Datenbanken in Eclipse

12.1	Einstieg in das Thema.....	465
12.2	Spielereien mit Access	466
12.3	Die Einrichtung von DbEdit.....	473
12.4	Allgemeine Betrachtungen zu Datenbanken.....	475
12.5	MySQL.....	478
12.6	Eclipse und MySQL.....	485
12.7	Zusammenfassung	488
12.8	Aufgaben zum Kapitel.....	488
12.9	Webseiten zum Kapitel.....	489

- 1 **Was ist Java?**
- 2 **Die Installation von Java**
- 3 **Was ist Eclipse?**
- 4 **Die Installation von Eclipse**
- 5 **Konzepte der OOP**
- 6 **Mit Eclipse arbeiten**
- 7 **Java: Ein Baukasten für Objekte**
- 8 **OOP in der Praxis**
- 9 **Java-Oberflächen**
- 10 **Ausnahmen und Threads**
- 11 **Codekonventionen**
- 12 **Datenbanken in Eclipse**
- 13 **Datenbanken und Java**

12 Datenbanken in Eclipse

Unter Persistenz eines Objektes versteht man sein Überleben, wenn der Rechner einmal ausgeschaltet wird. Bisher haben wir unsere Objekte nur im Hauptspeicher gehegt und gepflegt. In diesem Kapitel befassen wir uns mit der Problematik, die Daten persistent zu machen. Allerdings erst einmal in Bezug auf die Nutzung von Datenbanken. Zu Java und Datenbanken hat das nächste Kapitel einiges zu sagen.

12.1 Einstieg in das Thema

Mit Eclipse haben wir das Plugin DbEdit installiert. Es dient dazu, von Eclipse aus eine Datenbank zu pflegen. Um mit Datenbanken halbwegs professionell umgehen zu können, benötigt man allerdings einiges Wissen, das wiederum außerhalb von dem liegt, was man wissen muss, um Java programmieren zu können. Unter Datenbanken stellt man sich zunächst einmal so etwas wie eine Festplatte vor. Das ist nicht ganz falsch. Bei Datenbanken geht es hauptsächlich darum, Daten zu speichern.

Es gibt drei Typen von Datenbanken, die traditionelle Version, relationale und objektorientierte Datenbanken.

Heute sind relationale Datenbanken vorherrschend. Während die ersten Datenbanken in erster Linie Techniken hatten, die eine optimale Speicherung von Datensätzen auf Festplatten realisierten, bieten heutige Datenbanken eine anwendungsorientierte Datenhaltung. Man arbeitete früher mit Zeigern, die Verbindungen zwischen den Datenpaketen darstellten, heute sind relationale Datenbanken tabellenorientiert. Gleichzeitig kam mit den relationalen Datenbanken **SQL** auf.

Der Vorläufer hieß **SEQUEL** (Structured English Query Language), daraus wurde die **Structured Query Language** SQL, zu Deutsch: Strukturierte Abfragesprache. Der Name sagt uns, dass sie vor allem dazu dienen soll, Daten aus einer Datenbank zu entnehmen. SQL ist relativ einfach zu erlernen, sie ist aber doch so umfangreich, dass wir hier nicht viel mehr als eine kurze Einführung geben können. Programmieren, ohne SQL zu können, ist eigentlich ein Unding. Sollte es also erforderlich sein, können Sie sich recht einfach anhand von Literatur oder Internetseiten (siehe Ende dieses Kapitels) SQL selbst beibringen. Es gibt zwar objektorientierte Datenbanken und sie sind für die OOP ideal, weil man bei ihnen direkt die ganzen Objekte ablegen kann. Aber sie hal-

ten sehr viele Daten mehrfach und sind deshalb unperformant. Eigentlich spielen sie im praktischen Einsatz keine große Rolle.

Ein schönes praktisches Beispiel für eine relationale Datenbank ist das Produkt MS-Access. Falls Sie über das Programm verfügen, sollten Sie es nicht versäumen, folgende Ausführungen damit praktisch nachzuvollziehen.

12.2 Spielereien mit Access

Grundlage der relationalen Datenbanken sind, wie erwähnt, die Tabellen. Eine Tabelle **KUNDE** beispielsweise kann die Spalten Name, Vorname, Wohnort usw. enthalten.

Nach dem Start von Access wählt man entweder **Leere Datenbank erstellen** oder man lässt sich von einem **Assistenten** bei der Erzeugung der Datenbank **KundenDB** helfen (Abbildung 12.1).



Abbildung 12.1 Erstellen einer leeren Access-Datenbank

Bei der Erstellung der Tabelle hat man wieder mehrere Möglichkeiten. In der **Entwurfsansicht** gibt man die Spaltennamen und die gewünschten Typen an (Abbildung 12.2). Ist man fertig, wird die Tabelle auf den Namen **KUNDE** getauft und ein Primärschlüssel erstellt.

Schon hat man eine Tabelle, in die man Werte eingeben kann. Das Ganze kann man auch in SQL beschreiben. Es sieht dann etwa so aus:

```
CREATE TABLE KUNDE (  
ID                NUMBER NOT NULL,  
NAME              CHAR (24) NULL,  
VORNAME          CHAR (16) NULL,
```



```

PLZ           CHAR (8) NULL,
ORT           CHAR (24) NULL,
STRASSENr    CHAR (24) NULL,
TEL           CHAR (24) NULL,
HANDY        CHAR (24) NULL,
EMAIL        CHAR (40) NULL
)

```

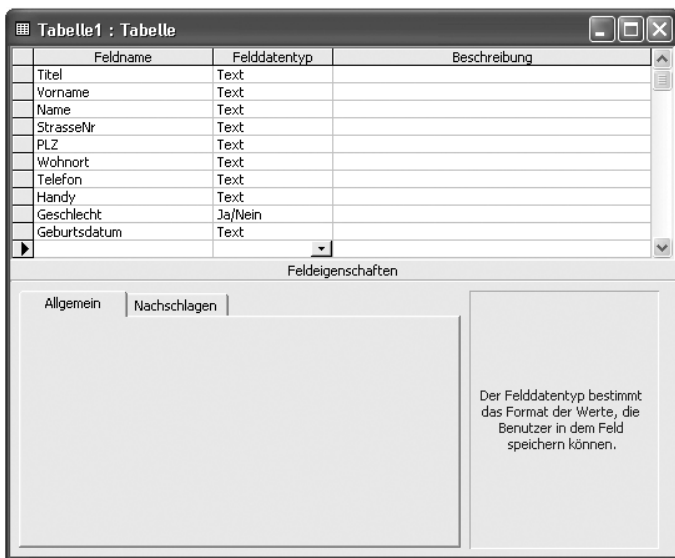


Abbildung 12.2 Felder der Tabelle festlegen

Sie werden sich fragen, wie Sie aus diese Anweisungen mit Access eine Tabelle erstellen sollen. Das ist eigentlich relativ einfach. Sie müssen die SQL-Anweisung nur als **Datendefinitionsabfrage** ausführen. Hierzu wählen Sie zunächst einmal im Menü **Einfügen · Abfrage** aus. Wechseln Sie in die **Entwurfsansicht**. Schließen Sie den folgenden Dialog (falls ein Dialog kommt) und wechseln Sie auf die SQL-Ansicht: **Ansicht · SQL-Ansicht**. In dem großen Editor-Fenster geben Sie die SQL-Anweisung ein. Falls Sie bereits eine Tabelle Kunde haben, schreiben Sie einfach:

```
CREATE TABLE KUNDE2 (
```

Danach schließt man den Editor einfach und lässt die Abfrage speichern. Wählt man nun **Ansicht · Datenbankobjekte · Abfragen** und klickt sie mit der RMT an, kann man sie ausführen lassen und die gewünschte Tabelle wird erzeugt (Abbildung 12.3).

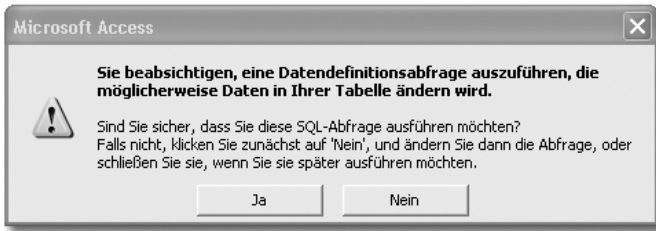


Abbildung 12.3 Erzeugen einer neuen Tabelle per Abfrage

Man findet sie in der Tabellenübersicht und sie sollte alle angegebenen Spalten besitzen.

Jetzt haben wir bereits unsere erste SQL-Anweisung kennen gelernt. Um genau zu sein, es ist eine **DDL** (Data Definition Language)-Anweisung. SQL trennt man nämlich eigentlich in zwei Sprachen, in die erwähnte DDL und in eine **DML** (Data Manipulation Language). Die Namen sind selbsterklärend und deuten an, was man mit den Sprachen machen kann.

Die wichtigsten Befehle der DDL wären:

Befehl	Bedeutung
create table	Erzeugt eine Tabelle
create index	Erstellt einen Index
drop table	Löscht eine Tabelle
drop index	Löscht einen Index
truncate	Löscht alle Zeilen einer Tabelle
alter table add constraint	Setzt eine Integritätsregel für eine Tabelle fest
alter table	Ändert Spalten, fügt Spalten an

Tabelle 12.1 Die wichtigsten DDL-Befehle

Die wichtigsten Befehle der DML sind:

Befehl	Bedeutung
insert	Fügt eine Zeile in eine Tabelle ein
update	Ändert die Zeile/n einer Tabelle
delete	Löscht eine Zeile/n einer Tabelle

Tabelle 12.2 Die wichtigsten DML-Befehle

Befehl	Bedeutung
select	Liest in der Datenbank Daten aus Tabellen oder Views
commit work	Schreibt alle Änderungen auf die Platte
rollback	Setzt alle Änderungen seit dem letzten commit zurück

Tabelle 12.2 Die wichtigsten DML-Befehle (Forts.)

Access lässt sich übrigens direkt aus Eclipse starten. Man kopiert die Datenbankdatei (z. B. KundenDB) ins src-Verzeichnis des Projektes, das auf die Datenbank zugreifen soll. In Eclipse refreshet man das Projekt:

RMT auf das Projekt · Refresh oder **[F5]**. Will man eine Änderung an der Datenbank durchführen, startet man Access einfach mit einem Doppelklick auf die Datenbankdatei im Projekt (Abbildung 12.4).

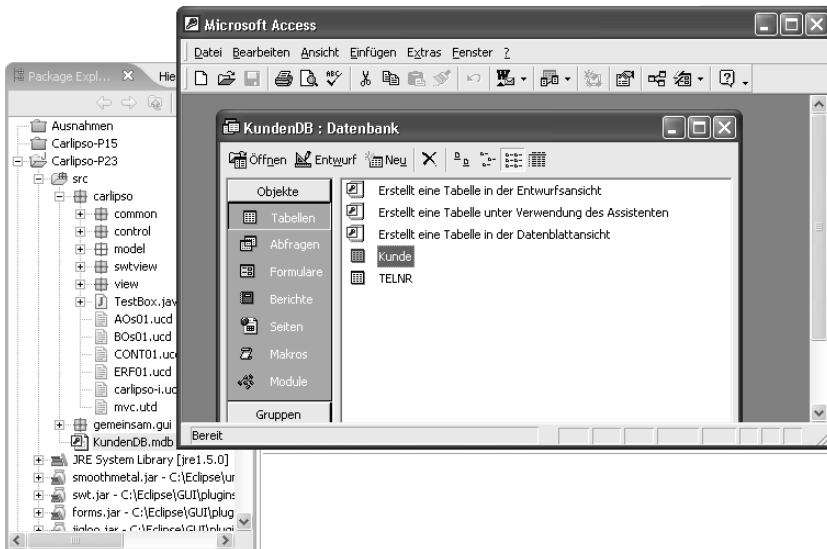


Abbildung 12.4 Access direkt aus Eclipse starten

Nachdem wir nun die erste Tabelle haben, geht es weiter. Daten bleiben nicht auf ewig festgeschrieben, die Strukturen leider auch nicht. Wenn nun neue Kommunikationsmöglichkeiten entstehen oder wir beispielsweise eine Telefonnummer vorgesehen haben, viele Kunden jedoch mehr und mehr zwei Nummern besitzen, müssen wir die Tabelle erweitern.

Das heißt zunächst einmal eine Spalte anfügen. Machbar ist das, wie aus unserer Tabelle (Tabelle 12.1) hervorgeht, mit **alter table**. Es heißt aber, dass alle bestehenden Datensätze um einen Wert in der neuen Spalte ergänzt werden

müssen. Versäumen wir das, kann es zu schweren Problemen in unseren Programmen kommen, die auf die Datenbank zugreifen.

Genau an dieser Stelle versagten die ersten Datenbanken und lieferten die relationalen Systeme eine gangbare Lösung. Man setzt einfach die einzelnen Tabellen miteinander in Beziehung, bildet Relationen zwischen den Daten der Datenbank. Ein Kunde kann beispielsweise zwei Telefonnummern haben. Tragen wir sie in die Tabelle **Kunde** ein, müssen wir zwei Datensätze Kunde schreiben und alle Daten, bis auf die Telefonnummer, verdoppeln. Ein sehr unrentables Verfahren. Erzeugen wir dagegen eine neue Tabelle **TELNR**, können wir die Kunde-Tabelle und das ist der wichtigste Punkt, völlig unverändert lassen.

Erstellen wir also eine neue Tabelle. Ihre erste Spalte soll die ID der Kundentabelle enthalten, damit wir eine Beziehung aufbauen können. Also heißt sie KUNDE_ID. Dann folgt die Spalte für die Telefonnummer. Versuchen wir es einmal alleine mit SQL:

```
CREATE TABLE TELNR (  
ID          NUMBER NOT NULL,  
KUNDE_ID   NUMBER NULL,  
TELEFON    CHAR (24) NULL  
)
```

Nachträglich müssen wir allerdings in Access das ID-Feld mit AutoWert belegen. So haben wir das Problem natürlich noch nicht gelöst. Wir müssen eine Beziehung zwischen der ID in der Tabelle Kunde und dem Feld Kunde_ID in der Tabelle TelNr herstellen. Das gelingt in Access mittels

Extras • Beziehungen... In einem Dialog können wir die beiden Tabellen auswählen oder wir klicken mit der RMT in das leere Feld und nutzen **Tabelle anzeigen**. Beide Tabellen werden markiert und mittels der Schaltfläche Hinzufügen aufgenommen.

Nun klicken wir die ID in der Tabelle Kunde an und ziehen sie zu KUNDE_ID. Im Dialog **Beziehungen bearbeiten** klicken wir die Schaltfläche **Verknüpfungstyp...** an. Als Eigenschaft wird ausgewählt: »Beinhaltet alle Datensätze aus 'Kunde' und nur die Datensätze aus 'TELNR', bei denen die Inhalte der verknüpften Felder gleich sind« (Abbildung 12.5). Die Auswahl wird bestätigt.

Nun sollte die Verknüpfungslinie eine Pfeilspitze bei **Kunde_ID** haben. Wir schließen und bestätigen, danach kann die Verknüpfung genutzt werden. Wir öffnen die Tabelle Kunde über **Ansicht • Datenbankobjekte • Tabellen** und klicken **Kunde** doppelt an. Tragen Sie einen ersten Kunden ein. Zum Eintragen

der Telefonnummer klickt man vorne auf das kleine Pluszeichen und trägt die Nummer ein. Wie zu erkennen ist, kann man nun beliebig viele Nummern eingeben (Abbildung 12.6).

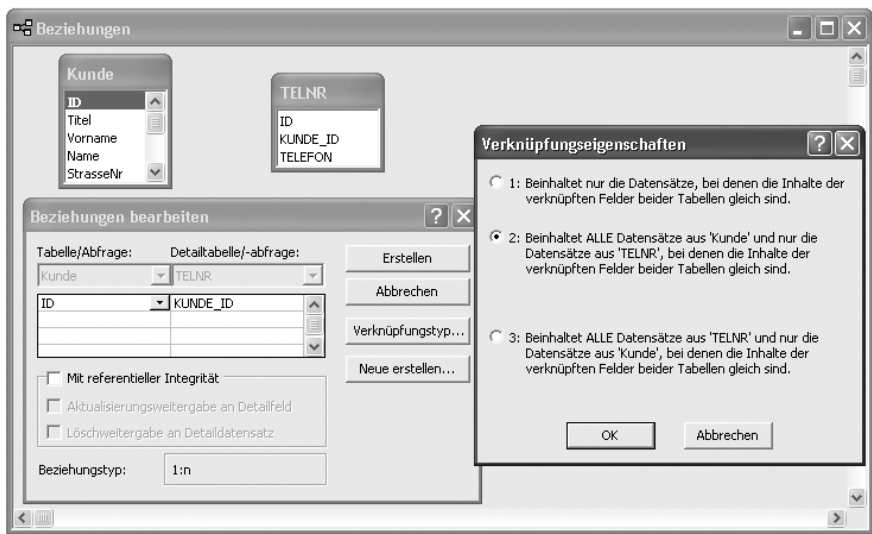


Abbildung 12.5 Verknüpfung zwischen zwei Tabellen aufbauen

Die Access-Datenbank **KundenDB** findet man auf der CD-ROM unter **\Zum_Buch\kap-0012\Access**.

ID	Titel	Vorname	Name	StrasseNr
1	Dr	Peter	Albert	Hauptchaussee
ID	TELEFON			
1	04345363434			
2	03526326323			
* (AutoWert)				

Abbildung 12.6 Verknüpfte Tabellen füllen

Um Access mit Java nutzen zu können, müssen wir einen ODBC-Treiber einrichten. Dazu wählt man **Start · Systemsteuerung** und weiter mit dem Symbol **Leistung und Wartung · Verwaltung · Datenquellen (ODBC)**

Hinzufügen... anklicken und Microsoft Access-Treiber (*.mdb) auswählen. **Fertigstellen** anklicken und als Datenquellenname **CarlippoDB** eintippen. Man wählt die Datenbank **KundenDB.mdb** aus, die man mit Access eben erstellt hat. Name und Passwort braucht man nicht anzugeben. Nach den Bestätigungen ist die ODBC-Verbindung zur Datenbank erzeugt (Abbildung 12.7).

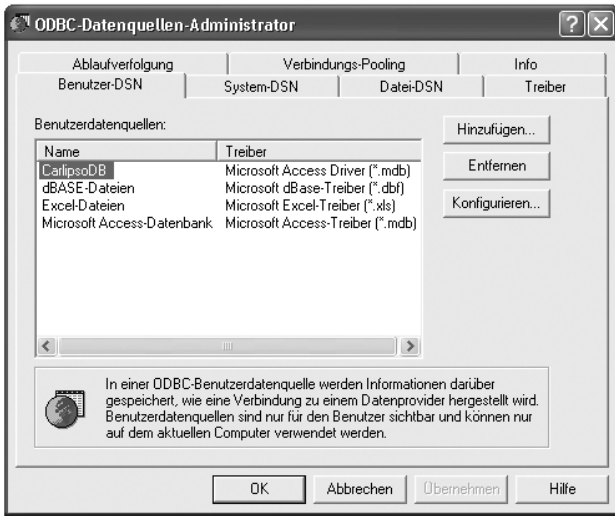


Abbildung 12.7 ODBC-Treiber einrichten

Nun kann man daran gehen, auf die Datenbank aus Eclipse mit DbEdit zuzugreifen. Da DbEdit bereits installiert ist, ruft man die Eclipse-Installation auf, die DbEdit beinhaltet. Das Plugin hält sich auf den ersten Blick diskret im Hintergrund, man kann allerdings überprüfen, ob es installiert ist, indem man **Hilfe · Info über Eclipse Plattform** aufruft.

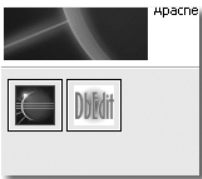


Abbildung 12.8 DbEdit ist installiert.

Ein kleines Symbol zeigt jetzt, dass DbEdit verfügbar ist (Abbildung 12.8). DbEdit stellt eine Reihe Views und eine Perspektive zum Umgang mit Datenbanken zur Verfügung. Die Perspektive ist etwas unhandlich. Um DbEdit problemlos nutzen zu können, sollte man sich eine eigene Perspektive erstellen: **Fenster · Perspektive öffnen · Andere... und wählt DbEdit aus.**

Die **Instant SQL-View** wird unter den Editor gezogen (Abbildung 12.9). Das wird gesichert mit **Fenster · Perspektive speichern als...** Der Name wird mit **DataBase** festgelegt.

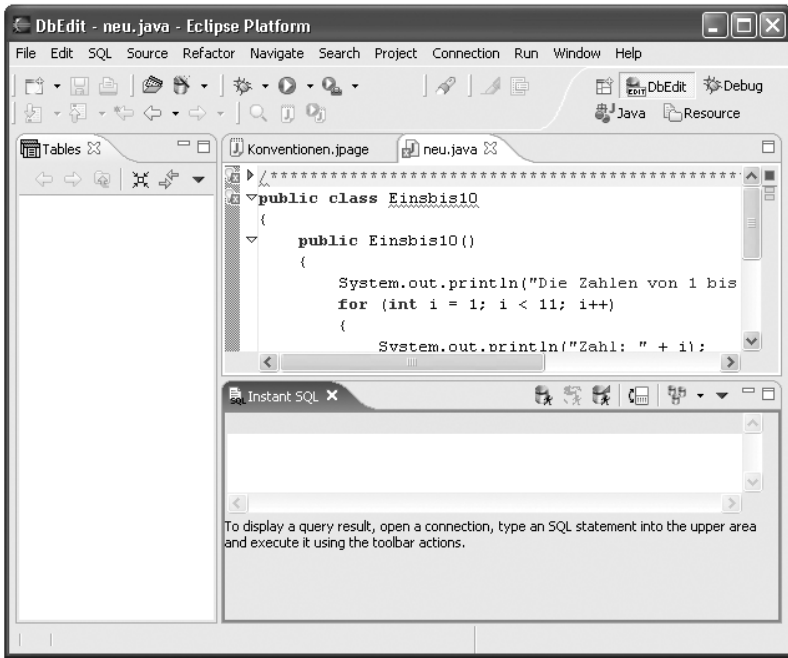


Abbildung 12.9 Neue DbEdit-Perspektive einrichten

12.3 Die Einrichtung von DbEdit

Somit ist die Oberfläche hergerichtet, aber anzeigen kann DbEdit noch nichts, weil die Konfiguration fehlt.

Fenster · Perspektive öffnen und **DataBase. RMT** in das Tables-Fenster · **Connection · Configure**. Es wird eine neue Verbindung mit der Schaltfläche **New** angelegt. Sie bekommt den Namen **Carlipso**.

Nun muss ein JDBC-Treiber angegeben werden. Hiermit ist das Gegenstück zum ODBC-Treiber gemeint. Auf diese Weise bekommt Java Zugriff auf den ODBC-Treiber und über diesen Treiber auf die Datenbank. Der JDBC-Treiber wird mit dem JRE mitgeliefert und befindet sich in einer JAR-Datei, genauso wie die API-Pakete, mit denen man die Java-Software entwickelt. Die Datei, in der er enthalten ist, ist die **rt.jar**. Sie befindet sich unter **jdk1.5.0\jre\lib**.

Man wählt die Registerkarte **Classpath** und macht sich mit der Schaltfläche **Add Archive** auf die Suche nach **rt.jar**. Nach der Auswahl ruft man mit der Schaltfläche **Apply** DbEdit dazu auf, den Treiber im Archiv zu suchen. Das sollte dann auch melden, dass es einen Treiber gefunden hat.

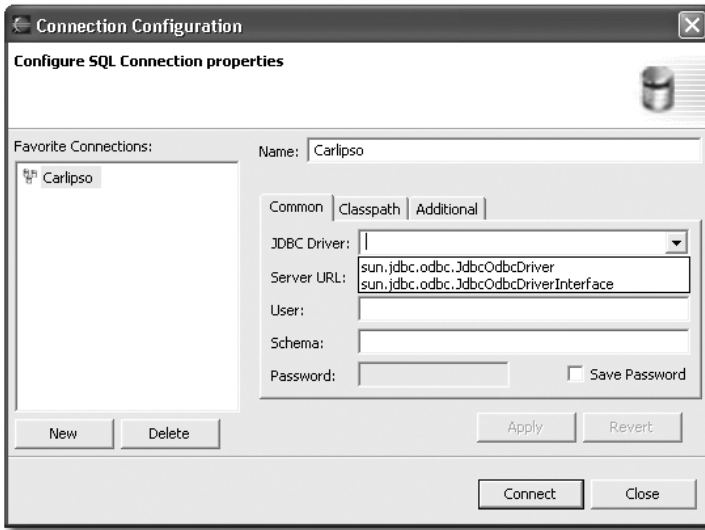


Abbildung 12.10 JDBC-Treiber auswählen

Die Combobox JDBC-Treiber kann nun **sun.jdbc.odbc.JdbcOdbcDriver** ausgewählt werden (Abbildung 12.10). Als Server-URL gibt man **jdbc:odbc:CarlipsoDB** ein.

Mittels der Schaltfläche **Connect** wird DbEdit mit der Datenbank verbunden. Schon zeigt sich in der View Tables ein Symbol mit der Beschriftung Carlipso. Die View funktioniert wie ein Datei-Explorer. Die existierenden Tabellen und Views werden als Symbole angezeigt und können geöffnet werden, indem man darauf doppelt klickt. Zur View gehört ein Editor, in den die Tabellen angezeigt werden. Man kann sie darin sogar editieren. Möglicherweise unterstützt der MS ODBC-Treiber diese Funktionalität nicht und man bekommt bereits beim Klicken auf das Datenbanksymbol eine Fehlermeldung. Das ist nicht weiter schlimm. Alle folgenden Funktionen sollten jedoch ausführbar sein.

DbEdit verfügt über eine weitere View, die für uns von Interesse ist, die so genannte **Instant SQL View**. Es ist die Sicht, die wir unter den Editor gescho-ben haben. Sie dient dazu, SQL-Anweisungen an die Datenbank zu senden.

Versuchen wir es mit einer ganz einfachen Anweisung:

```
select * from Kunde
```

Machen Sie es wie im Scrapbook, markieren und mit **RMT | Execute**. Sollte die Datenbank nicht verbunden sein, versuchen wir es mit dem Symbol, das drei Datenbanken zeigt, in der Kopfleiste der Instant SQL View.

Die SQL-Abfrage soll uns nun den gesamten Inhalt der Tabelle **Kunde** ausgeben. In der Tat wird der Result View die Tabelle angezeigt (Abbildung 12.11).

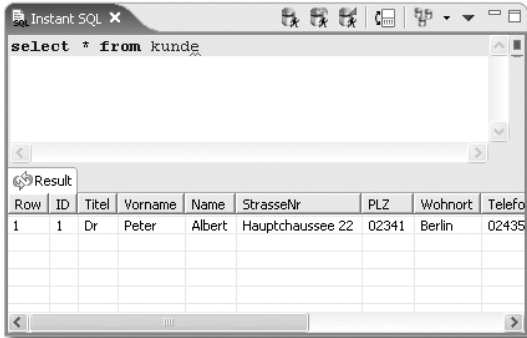


Abbildung 12.11 Per SQL-Statement Daten aus Tabelle lesen

Versuchen wir etwas mehr. Tragen wir doch einmal einen neuen Kunden mittels SQL ein:

```
INSERT into Kunde values (5, 'Prof', 'Berger', 'Herbert',
    'Frankfurter Str 12', '02524', 'Schlangenbad', '034634634',
    '87438278', -1, '23.2.1961')
```

Ein erneutes `select * from Kunde` sollte den neuen Kunden ebenfalls zeigen (Abbildung 12.12).

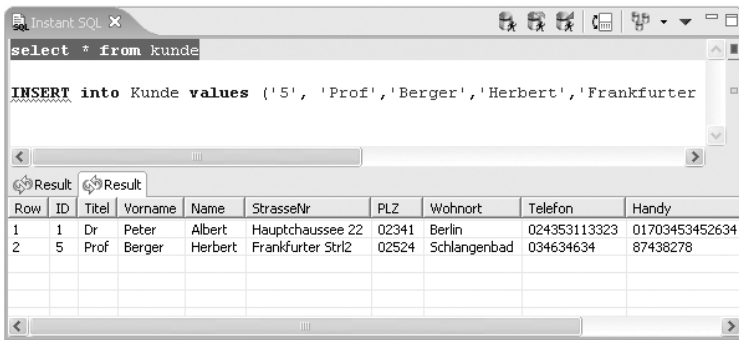


Abbildung 12.12 Kunden eintragen und ansehen

12.4 Allgemeine Betrachtungen zu Datenbanken

Wenn auch der erste Eindruck sein könnte, bei relationalen Datenbanken handle es sich nur um Tabellen mit Daten darin, so ist dem aber nicht so. Datenbanken sind weit mehr. Neben den eigentlichen Daten bestehen sie aus Benutzerrechten, Regeln, Funktionen, Zugriffsbedingungen und -berechtigungen,

neben den Tabellen und Beziehungen zwischen den Tabellen, die wir bereits kennen gelernt haben. Datenbanken dienen dazu, umfangreiche Datenmengen zu speichern und bereitzuhalten, aufbereitet zurückzuliefern und das für mehrere Anwender gleichzeitig.

Datenbanken sind also wahrhaftig nicht einfach nur Datenspeicher, sondern verwalten auch Regeln, die in Beziehungen zu den Daten stehen. So kann man beispielsweise sicherstellen, dass nur bestimmte Benutzer gewisse Daten lesen oder ändern dürfen. So kann man auch sicherstellen, dass die Daten Konsistenzbedingungen genügen; beispielsweise, dass die gespeicherten Adressen entweder gar keine oder eine fünfstellige Postleitzahl haben, die nur aus Ziffern besteht.

So genannte Datenbank-Managementsysteme (DBMS) verwalten die Daten der Datenbanken. Nur über das DBMS wird auf die Datenbank zugegriffen. Das DBMS prüft und kontrolliert dabei und führt komplexe Operationen im Auftrag der Anwendungen aus. Wie das DBMS dabei vorgeht, interessiert die Anwendungsprogramme überhaupt nicht. Sie haben nicht einmal direkten Kontakt zum DBMS, weil meist noch ein ODBC-Treiber dazwischensitzt, der die Wünsche der Anwendungsprogramme entsprechend umsetzt. DBMS nutzen eigene Regeln, um Redundanzen in den Daten gering zu halten. Erst bei der Rücklieferung der Daten werden sie wieder so zusammengestellt, wie es sich gehört.

12.4.1 Zugriffssysteme und Datenbankserver

Datenbanken werden heute viel häufiger in Form so genannter Datenbankserver genutzt, als in der Form von Standalone-Datenbanken, wie wir bis jetzt Access genutzt haben.

Wie kann man sich das vorstellen? Im Grunde ist es nur eine Datenbank mit DBMS und anhängendem Server, der dazu dient, die Daten übers Netz und vor allem zahlreichen Nutzern zur Verfügung zu stellen.

Wenn wir von Datenbanken sprechen, meinen wir eigentlich den Plattenplatz auf dem ein DBMS die Daten einer Datenbank ablegt oder wir meinen das DBMS selbst. Schnittstellen zu einer Datenbank sind eigentlich immer nur Schnittstellen zum DBMS. Man spricht jedoch üblicherweise nur von Datenbankschnittstellen.

Die Schnittstellen zu den Datenbanken über das DBMS sind aus verschiedenen Schichten (Layer) aufgebaut. Viele dieser Schnittstellen sind Remoteschnittstellen, also netzwerkfähig. DBMS verwenden dabei meist das TCP/IP-Protokoll, das uns vom Internet her längst geläufig ist.

Genau wie unter Access werden auch über die Netzwerk-Connection die Befehle oder Kommandos in Datenbanksprachen abgesetzt. Um über das Netz Daten zu senden, müssen sie serialisierbar sein. Tabellen sind aber nichts Serielles. SQL aber kann man als fortlaufenden Zeichenstrom darstellen.

Natürlich verfügt jede Datenbank über eine eigene Schnittstelle, das heißt, der genaue Aufbau der Kommandos und vor allem die Übertragung über das Netzwerk unterscheiden sich erheblich – selbst bei der Verwendung von SQL. Wir werden das noch feststellen, wenn wir bald SQL-Befehle für MySQL erzeugen. Sie unterscheiden sich ein klein wenig von dem SQL, das von Access verstanden wird.

Zu den Datenbankservern gibt es Dienstprogramme, die über diese Schnittstellen mit der Datenbank kommunizieren können. Wir werden für MySQL eine ganze Reihe Dienstprogramme benötigen, die alle über einen Remotezugriff auf die Datenbank zugreifen. Meist ist ein Dienstprogramm für bestimmte Aufgaben gedacht, beispielsweise für die erste Einrichtung. Andere Dienstprogramme bieten Zusatzfunktionen, die man für die Administration benötigt, beispielsweise für das Anlegen von Benutzern. Manche Datenbanken bieten solche Zusatzfunktionen auch über SQL an. Bei der Datenbank Oracle findet man beispielsweise einen so erweiterten SQL-Dialekt. Die Kommandos sind so aufgebaut, dass sie den SQL-Standards genügen und ihnen nicht widersprechen. Es handelt sich dabei um so genannte SQL-Erweiterungen. Da man mit Java jedoch datenbankunabhängig programmieren kann, sollte man solche speziellen SQL-Befehle möglichst wenig einsetzen. Man erspart sich so Probleme bei der Nutzung anderer Datenbanken. SQL sorgt für eine gewisse Unabhängigkeit auf der Anwendungsebene, die aber nur gewährleistet ist, wenn man die genormten Befehle einsetzt, die auch von allen Datenbanken verstanden werden.

Die Unabhängigkeit auf der Programmierenebene wird durch eine andere Art von Schnittstellen gewährleistet. Es sind eine Reihe von Funktionen, die aus einer Programmiersprache aufgerufen werden können. Die Schnittstellen, die diese Vereinheitlichung bewirken, sind ODBC und JDBC. ODBC, OpenDataBaseConnectivity, beschreibt, wie Programme mit der Datenbank (eigentlich dem DBMS) kommunizieren sollen. ODBC ist eigentlich eine Funktionsbibliothek, die SQL bereits voraussetzt. ODBC regelt aber auch eine Ebene tiefer den Verbindungsaufbau zu einer Datenbank und berücksichtigt dabei die Benutzerauthentifizierung und andere Sicherheitsmechanismen. JDBC ist, wie wir bereits wissen, das Gegenstück zu Java.

Selbst, wenn man aufgrund des Einsatzes von ODBC und JDBC davon ausgeht, dass alles Genormte problemlos zusammenpasst, ist das in der Praxis eigentlich

nie so. Falls Sie den Fehler beim Zugriff von DbEdit auf unserer Access-Datenbank auch hatten, haben Sie diese Praxis bereits eingehend kennen gelernt. Leider ist es oft so und es kostet oft mehr Mühe als erforderlich, die Datenbankzugriffe halbwegs brauchbar zu realisieren.

12.5 MySQL

12.5.1 MySql installieren

Auf der CD-ROM befindet sich ein Version von MySQL für Windows. Es ist zwar nicht so nutzerfreundlich wie Access, entspricht aber weit mehr dem üblichen Einsatz einen Datenbankserver zu nutzen, als ein System wie Access. Die Installationsroutine befindet sich unter `\software\datenbank` und heißt: **mysql-essential-4.1.9-win32.msi**.

Man startet sie mit einem Doppelklick. Zuerst gibt es den üblichen Begrüßungsdialog zu sehen (Abbildung 12.13).



Abbildung 12.13 MySQL Setup

Nun kann man auswählen, ob man eine Installation mit den üblichen Funktionalitäten (Standard) durchführt, oder ob man alles installiert oder eine benutzerdefinierte Installation durchführt (Abbildung 12.14).

Wir wählen die Standardinstallation. Die Installation wird vorbereitet und die Dateien kopiert (Abbildung 12.15).

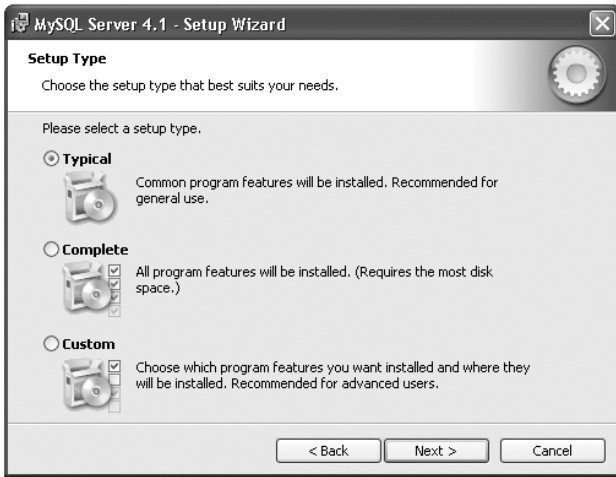


Abbildung 12.14 Auswahl des Setup-Typs



Abbildung 12.15 Installation läuft

Jetzt bietet MySQL an, einen Account auf MySQL.com einzurichten, damit man immer Zugriff auf die neusten Versionen hat. Das ist zwar durchaus sinnvoll, wird hier jedoch nicht näher beschrieben.

Der nächste Dialog fragt, ob man eine Konfiguration des MySQL Servers durchführen möchte (Abbildung 12.16).



Abbildung 12.16 Konfiguration starten

Man lässt den Merkerkasten aktiviert und wählt die Schaltfläche **Finish** an. Es wird in einem weiteren Dialog auf den Start der Konfiguration hingewiesen. Anschließend folgt eine Abfrage, ob eine detaillierte Konfiguration oder eine Standardkonfiguration durchgeführt werden soll (Abbildung 12.17). Hier wird die Standardversion der Konfiguration gewählt.

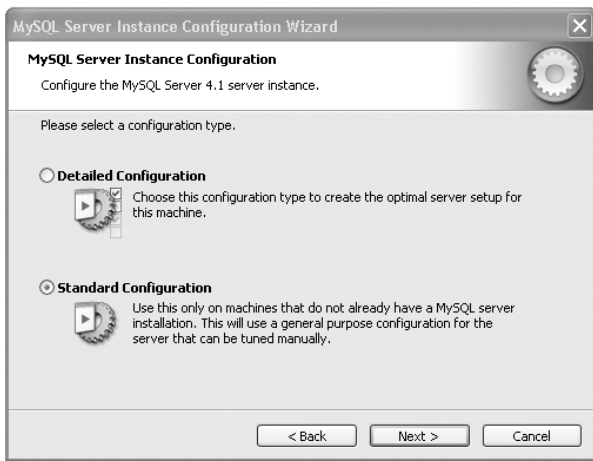


Abbildung 12.17 Auswahl der Standardkonfiguration

Des Weiteren fragt MySQL, ob es als Windows-Service laufen soll und wie der Service heißt. Außerdem kann man den Path setzen lassen, damit man die Client- und Service-Routinen direkt im Kommandozeilenfenster starten kann, ohne die Verzeichnisnamen angeben zu müssen. Das wird natürlich aktiviert (Abbildung 12.18).



Abbildung 12.18 Als Service einrichten

Im nun folgenden Account-Dialog gibt man ein Passwort für den Root-Zugang zur Datenbank an (selbstverständlich gut merken). Vor allem darf man nicht vergessen, den **Root-Zugang** über **localhost** zu aktivieren, weil man wahrscheinlich nur einen Rechner zur Verfügung hat, um die Datenbank zu nutzen. Zusätzlich muss man den anonymen Zugang aktivieren (Abbildung 12.19).



Abbildung 12.19 Das Root-Passwort festlegen

Im nächsten Schritt wird die Konfiguration letztendlich ausgeführt. Am Ende muss noch einmal die Schaltfläche **Finish** angeklickt werden und die Installation ist abgeschlossen.

12.5.2 Die ersten Experimente mit einem Datenbankserver

So weit so gut. Nun müssen wir irgendwie Kontakt zu dem Server auf unserem Rechner aufnehmen. Um zu testen, ob der Server aktiv ist, öffnet man die Eingabeaufforderung und gibt folgendes Kommando ein:

```
mysqladmin -u root -h -p localhost ping
```

Zurück kommt zunächst nur ein:

```
Enter password:
```

Geben Sie das festgelegte Passwort ein.

Als Antwort von MySQL sollte nun die Meldung »mysqld is alive« kommen. Sollte eine Fehlermeldung kommen, ist irgend eine Einstellung nicht in Ordnung. Es kann aber auch sein, dass der Server gar nicht läuft. In Windows heißt das, dass man bei den Diensten nachsehen muss: **Start · Systemsteuerung · Leistung und Wartung · Verwaltung · Dienste**.

In der Liste der Dienste sollte sich ein Eintrag MySQL finden lassen. Ist er nicht gestartet, kann er mit den Links neben der Tabelle gestartet werden. Hier lässt er sich auch herunterfahren, wenn einmal Änderungen an den Einstellungen (ini-Datei) erforderlich sind. Der Server kann aber auch per mysqladmin heruntergefahren werden. Lassen Sie die Dienste offen und geben Sie in der Eingabeaufforderung folgendes Kommando ein:

```
mysqladmin -u root -h -p localhost shutdown
```

Wenn Sie nun im Dienstdialog den Menüpunkt **Aktion · Aktualisieren** wählen, sollte das »Gestartet« im Status verschwinden. Starten Sie den Server wieder; dies kann in der Eingabeaufforderung durchgeführt werden mit

```
mysqld
```

Praktischer ist jedoch der Start über den Dienstdialog. Was mysqladmin noch bietet, ist ein kleiner Status, der die nötigsten Informationen zum Server liefert. Man ruft ihn auf mit

```
mysqladmin -u root -h -p localhost status
```

Wir werden zwar in erster Linie mit DbEdit aus Eclipse heraus mit dem Server arbeiten, es ist aber ganz gut, wenn man auch die Tools kennt, die für die Pflege des Systems gedacht sind. Ganz wichtig ist der so genannte Client **mysql**. Er dient dazu, remote auf die Datenbank zuzugreifen. Dabei funktioniert er wie

ein zeilenorientiertes Programm und erlaubt uns, SQL-Befehle an die Datenbank zu schicken.

```
mysql -u root -h localhost -p
```

Statt des »DOS-Prompts« erhält man ein Prompt, das anzeigt, dass man sich im Clientprogramm befindet.

```
mysql>
```

Vor dem Prompt stehen noch einige Begrüßungsworte, aus denen hervorgeht, dass wir jeden SQL-Befehl mit einem Semikolon abschließen müssen. Mit der Eingabe von **help** kann man sich einen Überblick über die verfügbaren Kommandos machen.

Doch wir wollen uns natürlich die Datenbank selbst ansehen. Ist überhaupt eine Datenbank vorhanden? Die Entwickler von MySQL haben natürlich eine kleine Testdatenbank mitgeliefert. Man kann sich die Datenbank **test** und die Datenbank **mysql** ansehen. Um auf eine Datenbank zuzugreifen, muss man sie zuerst aktivieren, das gelingt mit dem Befehl USE:

```
mysql> USE mysql;
```

Der SQL-Befehl, um sich eine Datenbank, oder besser gesagt, eine Liste der darin enthaltenen Tabellen anzusehen, heißt:

```
mysql> SHOW TABLES;
```

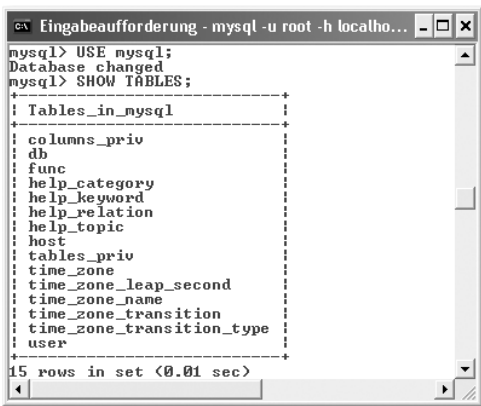


Abbildung 12.20 Tabellen der Datenbank mysql

12.5.3 Datenbanken erzeugen

Wir wollen nun zunächst eine eigene Datenbank erzeugen:

```
mysql> CREATE DATABASE Carlipso;
```

Um diese neue Datenbank zu nutzen, müssen wir sie wieder aktivieren.

```
mysql> USE carlipso;
```

Eine Anzeige der Tabellen mit SHOW TABLES darf natürlich noch nichts anzeigen. Auch das wollen wir ändern. Legen wir die Tabelle Kunde an. Der Editor funktioniert so, dass man mittels Return eine neue Zeile beginnen kann, die mit einem Semikolon abgeschlossen wird.

```
mysql> CREATE TABLE kunde(  
-> id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,  
-> titel CHAR(16) NULL,  
-> name CHAR(24) NULL,  
-> vorname CHAR (24) NULL,  
-> strassenr CHAR (24) NULL,  
-> plz CHAR (5) NULL,  
-> ort CHAR (25) NULL,  
-> tel CHAR(25) NULL,  
-> handy CHAR (25) NULL,  
-> geschlecht BINARY,  
-> geburtsdatum CHAR (25) NULL,  
-> PRIMARY KEY(id),  
-> INDEX(name, plz, ort));
```

Die einzelnen Typen findet man in der API-Dokumentation unter JDBC sehr schön erklärt (in Englisch). Der Pfad innerhalb der Dokumentation ist [\docs\guide\jdbc\getstart\GettingStartedTOC.fm.html](#).

```
mysql> desc kunde  
-> ;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned		PRI	NULL	auto_increment
titel	char(16)	YES		NULL	
name	char(24)	YES	MUL	NULL	
vorname	char(24)	YES		NULL	
strassenr	char(24)	YES		NULL	
plz	int(5) unsigned	YES		NULL	
ort	char(25)	YES		NULL	
tel	char(25)	YES		NULL	
handy	char(25)	YES		NULL	
geschlecht	binary(1)	YES		NULL	
geburtsdatum	char(25)	YES		NULL	

```
11 rows in set (0.00 sec)
```

Abbildung 12.21 Neu angelegte Tabelle anzeigen

Die Tabelle ist rasch angelegt (Abbildung 12.21). Mit **desc kunde** kann man sie sich anschauen. Mit **drop Table kunde**; hat man die Tabelle aber auch genauso rasch wieder gelöscht.

Aber wir wollen wenigstens einen Datensatz eintragen. Versuchen wir es mit unserem Herrn Berger:

```
mysql> INSERT into kunde values (  
-> 1, 'Prof', 'Berger', 'Herbert',  
-> 'Frankfurter Str 12',  
-> '02524', 'Schlangenbad',  
-> '034634634', '0187438278',  
true, '23.2.1961');
```

Diesen ersten Datensatz kann man sich bereits mit **select * from kunde**; anschauen.

Es lassen sich auch mehrere Datensätze mit einer into-Anweisung einstellen.

```
mysql> INSERT INTO kunde VALUES  
-> (1, '', 'Brunninger',...),  
-> (2, '', 'Müller', ...),  
-> (3, 'Ing', 'Beerg', ...);
```

Gelöscht werden Datensätze mit:

```
mysql> DELETE FROM kunde WHERE id = 2;
```

oder alle Datensätze in einer Tabelle:

```
mysql> DELETE FROM kunde;
```

12.6 Eclipse und MySQL

Da wir DbEdit bereits mit Access genutzt hatten, sollte es jetzt ein Einfaches sein, mit dem Plugin auf den Datenbankserver aufzusetzen. Öffnen wir die von uns erzeugte Perspektive **Fenster · Perspektive öffnen** und **DataBase**.

Für MySQL benötigen wir allerdings einen anderen JDBC-Treiber. Er befindet sich auf der CD-ROM zum Buch unter **\Software\Datenbank\Driver** und heißt **mysql.jar**. Man kopiert ihn am besten in das Eclipse-Verzeichnis der Eclipse-Installation mit dem DbEdit-Plugin. Bei der Konfiguration wird er unter Classpath angegeben, und in der Registerkarte **Common** sollte man im Feld **JDBC Driver** nun **org.gjt.mm.mysql.Driver** auswählen können. Ins Feld Server URL kommt **jdbc:mysql://localhost:3306/carlipso**.

Mit der Schaltfläche **Connect** wird der Verbindungsaufbau gestartet. Das Passwort braucht nicht eingegeben zu werden. Zunächst wird in der linken Tables View die Datenbank mit der Tabelle **kunde** angezeigt. Die Tabelle kann wie ein Verzeichnis aufgeklappt werden. Mit einem Doppelklick auf kunde wird der Inhalt der Tabelle in der Editor-View angezeigt (Abbildung 12.22).

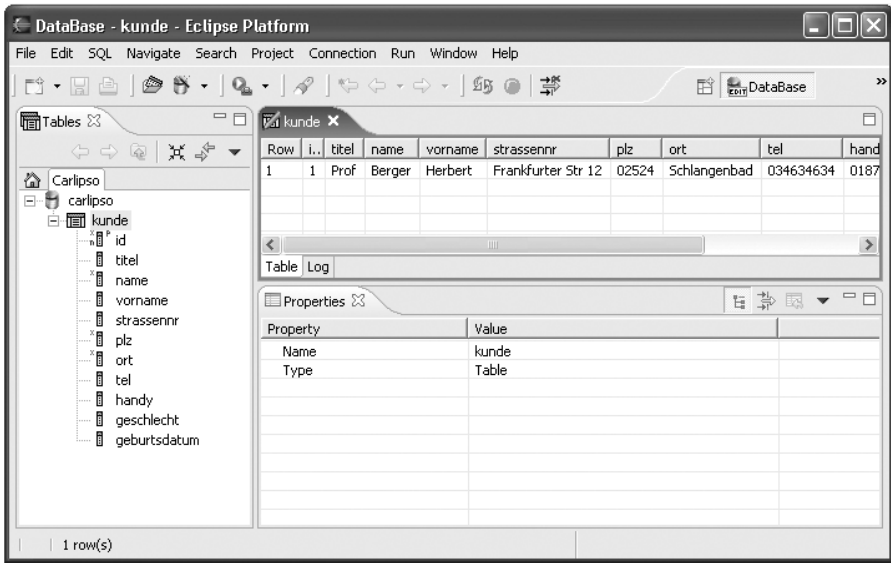


Abbildung 12.22 Eclipse riskiert einen Blick auf MySQL

Um einen Kunden zu erfassen, klickt man mit der **RMT** auf die Tabelle im **Editor • Insert Row**. Eine neue Zeile wird eingefügt und kann mit den entsprechenden Daten gefüllt werden. Nach der letzten Spalte wird sie abgespeichert. Auch nachträgliche Korrekturen sind noch mit einem Doppelklick auf dem entsprechenden Feld möglich.

DbEdit bietet weitere Editiermöglichkeiten. Klickt man mit der **RMT** auf den Datenbanknamen kann man über das Menü eine neue Verbindung (Connection) einrichten, ein Schema, eine Tabelle oder ein SQL-Datei aufbauen. Wir richten eine weitere Tabelle ein und nennen sie Kfz. Nach dem Namen müssen in jedem Fall Spalten eingefügt werden, das gelingt über den Reiter **Columns**. Es ist möglich, dass die vorliegende DbEdit-Version hier etwas Probleme macht. Nach einem Eclipse-Neustart lässt sie sich in jedem Fall ausführen.

Erzeugen Sie die Spalten: ID, Marke, Modell, Fahrzeugart, Baujahr, Leistung, Kilometerstand, Getriebeart, Kraftstoff, Farbe, Ausstattung. Überlegen Sie jeweils genau, welchen Typ Sie nehmen müssen. Die Abbildung 12.23 zeigt eine mögliche Lösung.

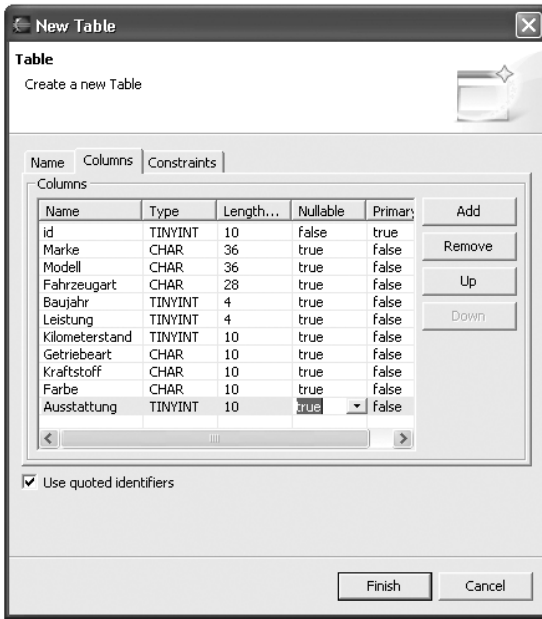


Abbildung 12.23 Tabelle Kfz anlegen

Diese Lösung benötigt eine weitere Tabelle für die **Ausstattung**. Sie muss die ID des Kfz, dazu einen Namen und eine Beschreibung sowie den Preis als Felder enthalten (Abbildung 12.24).

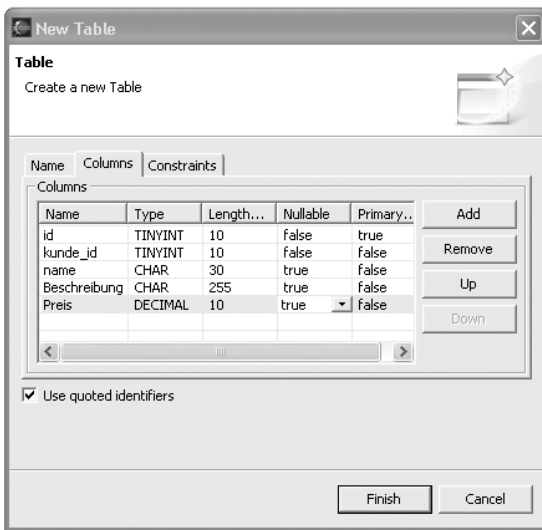


Abbildung 12.24 Tabelle Ausstattung erzeugen

12.7 Zusammenfassung

Relationale Datenbanken gehören heute fast zu jeder Anwendung. Man speichert in ihnen alles, von einfachen Ganzzahlen bis zu Bildern und Objekten. Es gibt zwar objektorientierte Datenbanken, aber sie haben sich nie richtig durchsetzen können. Als gutes Beispiel für eine nutzerfreundliche relationale Datenbank kann Access dienen. Es erlaubt auf einfache Weise eine recht detaillierte Konstruktion von Datenbanken.

Wir haben so eine Kundendatenbank aufgebaut. Sie enthielt zwar lediglich eine Tabelle, hat uns aber gezeigt, wie man Tabellen anlegt und befüllt. Vor allem aber diente sie zu einer Einführung in SQL, der weltweit eingesetzten Datenbanksprache. Eigentlich besteht sie aus zwei Sprachen, der DDL und der DML. Die wichtigsten Befehle der beiden Sprachen haben wir uns in Übersichten angeschaut.

Nachdem mit Access eine minimale Datenbank aufgebaut war, wurde das Plugin DbEdit in Eclipse konfiguriert, so dass man hier auf diese Datenbank zugreifen konnte. Ganz entscheidend ist dabei die Einrichtung eines JDBC-Treibers, weil er die Verbindung zum ODBC-Treiber von Access aufbaut.

Nachdem wir uns mit Datenbanken im Allgemeinen beschäftigt haben, wurde ein Datenbankserver eingerichtet. Er nennt sich MySQL und lag in einer Version für Windows vor. Datenbankserver werden über eine Remoteverbindung gewartet, deshalb benötigen sie zusätzliche Routinen. Für MySQL gibt es den Client **mysql.exe** und das Wartungs-Tool **mysqladmin.exe**.

Auf dem Server richteten wir die Datenbank Carlipso ein und realisierten auf ihr auch eine Tabelle Kunde. In einem letzten Schritt verbanden wir auch hier das DbEdit-Plugin mit der Datenbank und pflegten sie aus Eclipse heraus. Das ging bis zum Anlegen von neuen Tabellen und der Eintragung von Datensätzen. Im nächsten Kapitel werden wir aus unserem Java-Programm heraus auf die Datenbanken zugreifen.

12.8 Aufgaben zum Kapitel

Theorie

1. Welche Arten von Datenbanken haben wir in diesem Kapitel genannt?
2. Mit welchen Mitteln arbeiten relationale Datenbanken?
3. Warum werden objektorientierte Datenbanken kaum verwendet?
4. Was bedeuten die Buchstaben SQL?
5. Wozu dient SQL?

6. Aus welchen beiden Sprachen ist SQL aufgebaut?
7. Der ODBC-Treiber ist für die Kommunikation zwischen Java und Access wichtig. In wieweit ist diese Aussage richtig?
8. Was ist die Aufgabe des JDBC-Treibers?
9. Welche JDBC-Treiber haben wir kennen gelernt?
10. Wie kann man DbEdit mit MySQL verbinden?

Praxis

1. Sie ahnen es schon? Natürlich haben wir wieder mit Absicht die Kundennummer weggelassen. Versuchen Sie eine Spalte Kundennummer mittels Eclipse in der Tabelle Kunde einzufügen.
2. Erzeugen Sie mit Eclipse eine Tabelle Preis, die drei Felder enthält, eine Kfz-Id, eine Kategorie als Ziffer und ein Feld für den Preis.

12.9 Webseiten zum Kapitel

URL	Beschreibung
http://www.mysql.de/	Deutsche MySQL-Homepage
http://www.mysql.com/	MySQL-Homepage
http://www.worldserver.com/mm.mysql/	JDBC-Treiber
http://www.uni-koeln.de/dv/erfahrung/beratungsdb/vortrag1.html	Vortrag Java und Datenbanken
http://www.fh-wedel.de/~si/projekte/ss98/Ausarbeitung/HTTPEndJDBC/javadb/ausarbeitung/jdbcb.htm	Ausarbeitung zu Java und Datenbanken
http://www.codefutures.com/java-mysql/	MySQL und Java
http://tecfa.unige.ch/guides/tie/html/java-mysql/java-mysql.html	MySQL und Java
http://www.webservertalk.com/MySQL_Java_277.htm	MySQL und Java
http://forum.java.sun.com/thread.jspa	Sun-Forum MySQL
http://rs50.bv.tu-berlin.de/lehre/seminare/JDBC/	TU Berlin zu Datenbanken und Java
http://www.java.de/forum/forumlist/1/	Java User Group Forum allgemein
http://www.java.de/forum/messagelist/8/	Java User Group Forum JDBC
http://jdbc.postgresql.org/	Postgres JDBC-Treiber

URL	Beschreibung
http://www-db.stanford.edu/~ullman/fcdb/oracle/or-jdbc.html	JDBC-Infos
http://web.f4.fhtw-berlin.de/hartwig/JDBC/jdbc.html	Einführung in JDBC von der FHTWBerlin

Index

3-Tier Modell 495

A

abstrakt 139, 218
abstrakte Klassen 152, 216
abstrakte Methode 216
Alan Turing 25
Algol 60 26
Amigos 125
Analyse 269
Anekdote 23
Anwendungsfall 269
Anwendungsobjekte 129, 271
anzeigen 290
API 127, 212
API-Doku 312
API-Dokumentation 54, 437
APL 47
Apple iMac 23
Applikation 269
Arbeitsfläche 139
Arbeitsset 169, 174
Archivart 99
ArrayList 284
Arrays 229
ASCII 308
Assembler 26
Ast 124
atomare Typen 226
AtomicInteger 228
AtomicLong 228
Attribute 122, 272, 299
Aufbau 176
auflösen 214
Aufrufstelle 150
ausführen 189
Auslieferung 91
Ausnahmefälle 401
AWT 84
AWT-Oberfläche 217

B

Basisklassen 124
Baum 124
Bauplan 121

Bausteine 163
Bedingungen 258
Bertrand Russell 119
Betriebssystemebene 163
Bezeichnungen 124
Beziehungen 124
Bibliotheken 173
BigDecimal 228
BigInteger 228
Blöcke 445
Booch 125
boolean 224
Build-Vorgang 174
Business-Objekte 271
Businessobjekte 129
byte 224
Bytecodedatei 97

C

C 81, 86
C# 85
C++ 84, 87
CAD 85
CAD-System 163
Calendar 281
CASE 79
Castings 24
catch 325
CDT 86
CeBit 24
Chain of Responsibility 430
char 224
Child 169
class 122
ClassNotFoundException 404
Cobol 24, 85
Code Assistant 323
Codekompletion 90
Codekonventionen 427
Combobox 59
Compiler 234
Component 217, 218
Computer 25
connect 474, 496
Connection 486

Constructor Summary. 235
Custom Setup Dialoges 62
CVS 90, 96
CVS-Sichten 170

D

daemon 414
Dämon 45, 414
Darwin 121
Datei 180
Dateiaufbau 429
Dateiendungen 431
Dateiname 128, 429
Datenbank 89, 465
Datenbankserver 476
Datenbankzugriffe 493
Datendefinitionsabfrage 467
Datenkapselung 297
Datenkonstrukte 432
Datenquellenname 471
Datentypen 226
DbEdit 106, 465
DBMS 476
DDL 468
Deaktivierung 111
Debuggen 407
Debugger 81, 91, 166, 250
Dechriffierung 25
Decompiler 41
default 238
Defaultverzeichnis 171
Define 88
definiert 233
Definition 146
Deinstallation 73, 112
Deklarationen 442
deklariert 233
Design 88
Designpattern 430
Deutsch 104
Develop 88
Diagramm 133, 151, 283
Dialog 288
Dialogbox 43, 129, 171
Dialogfeld 214
DML 468
do-Schleife 254
double 225

do-while 447
Drag-and-Drop 203
DVD-Player 25

E

Early Binding 149
Ebenen 191
Eclipse starten 101
Editor verlinken 191
Editoren 164, 176
Eiffel 85
Eigenschaften 102, 122, 135, 213
Eingabeaufforderung 71
Einrückung 435
Einzeiliger Kommentar 437
Emacs 79
Embeded System 30
EMF 107
Endemarke 437
Enigma 25
Enterprice Java Beans 55
entkoppeln 217
Entwicklerwerkzeug 162
Entwurfsansicht 466
Entwurfsmuster 288, 430, 500
Entzipper 97
erben 136, 218
erfassen 290
erlernen 47
error-handling 325
Erscheinungen 119
erweitern 305, 309
EVA 290
Eventbehandlung 401
Exception 403
Exception-Handling 404
Exceptions 401
exist 244, 248
extends 145
External JARs 173

F

Fachliteratur 30
Fall-Throughs 448
Fallunterscheidung 150
Fehlermeldung 213, 232, 407
Fehlersymbol 137, 214
Fenster 179

Fernsteuerung 30
Festplatte 97
Fields 220
final 235, 246
finalize 318
Flag 412
float 225
Formatierprogramm 428
for-Schleifen 254, 447
Forth 47
Frame 321
Framework 46, 85, 163
frühe Bindung 149
funktionale Sprache 43

G

Ganzzahl 44
Garbage-Collection 303
GByte 38
Gebäudesteuerung 25
gecastet 313
GEF 107
gelistete Attribute 280
Generalisation 137
Geschäftsobjekte 129, 130
Geschäftsvorfall 285
getLength 44
getter-Methoden 182, 401, 509
Green 30
Griechenland 119
Grundausrüstung 272
Grundsystem 104
GUI 15
GUI-Applikationen 45
GUI-Builder 80
GUI-Klassen 217

H

Hardware 96
Hardwarevoraussetzungen 54
Hauptteil 433
Hauptthread 409
Header-Datei 34
Hersteller 125
Hilfdatei 236
Hinweise 15
Historie 90, 434
Homepage 104

HotJava 31
HTML 25

I

IBM 104
IDE 15, 79
Ideenlehre 119
IllegalThreadStateException 414
imperative Sprache 43
implementieren 218
importieren/exportieren 192
Index 212, 312
Innere Klassen 274
innert class 272
Installation 53, 96
Installation von Plugins 106
Installationsprogramm 59
instanceof 511
Instanz 125
instanzieren 295
Instanziierung 36, 125, 140, 278
Instanz-Initialisatoren 418
int 223, 225
Interface 141, 216, 218, 453
InterruptedException 325
invoke 509
isDaemon 416

J

J++ 85
Jacobson 125
Jar-Dateien 211
Java Web Start 53
Javacompiler 42
Javadoc-Kommentar 433
Java-Handy 24
Java-Lebenslauf 31
Java-Oberflächen 217
Java-Pakete 127
Java-Perspektive 164, 493
Java-Story 30
Java-Trainer 72
JBuilder 80, 88
JDBC 493
JDBC-Aufrufe 495
JDBC-Treiber 473
JDK 53
JDK EE 55

JFace 84
JFrame 291
Jigloo-GuiBuilder 106
JRE 53
JSP-Anwendungen 169
JVM 23

K

Kaffeessorte 23
Key 229
Kfz 124
Klassen 223
Klassenbezeichnung 123
Klassendiagramm 269
Klassenkopf 221
Klassenkörper 279
Klassennamen 323, 430
Klassifizierung 143
Kommandozeilen-Programme 45
Kommentare 241, 273, 433
Kommentarzeichen 215
Kommunikation 251
Kompiler 79
Komplexe Datentypen 223, 227
Komprimierung 273
Konfiguration 481
konkret 139
konkrete Klassen 216
Konsole 97
Konsolen-Ansicht 233
Konsortium 84
Konstanten 234, 453
Konstruktor 122, 134, 277, 504
Konstruktormethode 236
Konsumelektronik 30
Konventionen 427
Kunden 209

L

Language-Pack 104
Late Binding 149
Latest Release 96
Laufvariablen 443
Lebenszyklus 429
Leerzeichen 450
Leerzeilen 449
Lesbarkeit 429
lesen 498

Libraries 53
Library 211
Linné 121
Linux 23, 228
Liste 313
Listen 229
Literale 35
Lizenz 59
LMT 15
long 225

M

main-Methode 247, 275
Manage 88
Markierung 231
Maschinencode 26
Mathematik 37
Mauszeiger 61
mehrdeutig 497
mehrzeiliger Kommentar 437
Meldung 403
Menubar 179
Menüleiste 179
Menüs 165
Methoden 134, 216, 273, 453
Methodenkörper 153, 236
Methodik 171
Micro-Edition 55
Microprozessor 39
Middleware 494
Milestones 95
Millisekunden 412
Modell 125, 499
Model-View-Controller 288
Müllbeseitigung 303
MultiThreading 409
MVC-Prinzip 430
MySQL 479, 480

N

Nachrichten 134, 220
Namenskonventionen 451
Navigatorview 190
Navigieren 186
neuronale Netze 44
new 276
Nomenklatur 121

null 277
NumberFormatException 402

O

Oak 30
Oberklassen 124
Object 291
Object-relational mapping 494
Objekt 120, 122
objektorientierte Sprache 27
Objektvariablen 148
ODBC-Bridge 497
ODBC-Verbindung 471
offene Attribute 280
Officesystem 163
OMG 125
Omondo UML 106
Omondo-UML-Plugin 126
OOP 15, 147, 288
Open Source 30, 83
Operationen 124, 134
Operatoren 35
Ordner 126
OTI 83

P

Package Explorer 294, 312
Paket 129, 212, 319
Paket-Explorer 127, 128, 131
Paket-Strukturen 211
Paradigma 121
Parameter 300
parseInt 402
PDA 55
Perl 84, 86
Perspectives 164
Perspektive
 öffnen 177
 schließen 178
 zurücksetzen 178
Perspektiven 164
Pfade 173
Phasenmodell 269
PHP 85
PL/1 24
Plato 119
Plattformunabhängigkeit 42
Plugins 44, 83

Polymorphie 294
Polymorphismus 148
PowerPC 23
Primitive Datentypen 223
println 34, 317
private 133, 143, 238, 290
Programmautoren 299
Programmiersprachen 16
Programmierung ohne Eclipse 70
Projekt 89, 171, 187
Projekt löschen 175
Projektdesign 270
Projektname 167
Projektnamen 211
Projektverwaltung 89
proprietäre Sprache 28
protected 238
Protokollnamen 497
Prozedur 150
prozedurale Programmierung 209
prozedurale Sprache 27
public 238
Python 85

Q

QNX 84
Qualifizierung 277
Quellcode 26, 79, 273, 283, 438
Quelle 181
Quelle-Registerkarte 173
Quellverzeichnis 126
Quick C 81

R

Rational 83
Rechenoperationen 308
Redundanzen 143, 297
Refactoring 153, 183, 293
Reflection-Klassen 297
referenzieren 300, 307
Registerleisten 200
relationale Datenbanken 465
Relationen 470
resume 412
RMT 15
Ruby 85
Rückgabewerte 456
Rumbaugh 125

run 409
Runnable 409
Runtime Environment 73

S

SAP 84
Schleife 253, 313
schließende Klammer 220
Schlüsselwörter 35
Scrapbook Page 229, 275
Scrapbookseite 166
screenSize 323
SE 55
Seiteneffekte 440
Semantik 34
Semikolon 151
SEQUEL 465
Serializable 141
Servlets 169
setLength 44
setLocation 324
Setter 182, 401
Setup 59
SGML 25
short 224
Shortcuts 161
Sicht 288
Sichtbarkeit 239
Sichten 164
Signal 402
Signal-Werfen 403
Signatur 236, 302
Singleton 500
sleep 324
Smalltalk 81, 166
Smalltalk 80 34
Snippeteditor 166, 229
Softwareobjekt 122, 209
Softwarewelt 24
Sonderausstattung 272
Spaltennamen 466, 510
späte Bindung 149
Speicher 220
Speicherbereinigung 302
Spezialisierung 137
Splash-Screen 296, 321
Sprachinstallation 104
SQL 465

SQL-Befehle 483
SQL-Statements 494
Squeak 34
Stacks 229
Standardkonstruktor 306
Startmarke 437
static 235, 246
Statusbar 200
steuern 290
Steuerung 288
stop 412
Stream Stable Builds 96
String 168, 227
Subjekte 120
Subklassen 124
SUN 23
Sun World 31
SUN-Workstation 23
super 138
Superklasse 124, 213, 279, 305
suspend 412
Swing 84
switch 448
Symbole 125
Symbolleiste 165
synchron 288
Syntax 31
Sysouts 403

T

Tabelle 466, 510
Tabellenkalkulation 163
Tastenkürzel 161, 193
Team 192
Test 88
Testen 275
Texteditor 193, 212
Textverarbeitung 85, 163
Thread 228, 324
throw 403
Throwable 403
toString 226
Treiber 496
try 325
TRY-/CATCH-Block 182
try-catch 401
Try-catch-Block 404
Turing-Maschine 26

Turing-Test 26
Tutorials 103
Typstrenge 297

U

überladen 305, 307, 317
überschreiben 217, 305
UML 15, 125, 209
UML-Diagramme 125
UML-Plugin 499
Umwandlung in eine Zahl 401
Undo 90
UNICODE 308
Unified Modeling Language 125
Unikat 120
Unterfenster 169
Unterklasse 148, 305
Untersuchen 243
URL 15
useCases 125

V

Variablen 220, 453
VBA-Routinen 45
Verbindung 486, 498
Vererbung 121, 144, 294
Vererbungspfeil 145
Verknüpfungslinie 470
verlinken 283
Versionen 95
Verteilte Anwendungen 46
Verzeichnisse 173
Verzeichnisstruktur 127
Views 164
View-Verzeichnis 173
visibility 241
VM 15
VOB-Verzeichnis 173
void 247
Von Neumann-Rechner 39
Vorgehensmodell 269

W

Wrapperklassen 226
Wartung 429
Webseite 64
WebSphere 80
Werkbank 164

while-Schleife 254, 447
Windows 163
Windows2000 23
Workbench 164
Workspace 102
Wrapperklassen 37

X

XML 25, 53

Z

Zahlenformat-Ausnahme 402
Zeichenkette 37
Zugriff 306
Zusicherungen 136
Zuständigkeitskette 430
Zuweisungen 455