

Thomas Künneth



Eclipse 3.3

Einstieg in Eclipse 3.3

- Für Windows, MacOS X und Linux geeignet
- RCP-, Web- und Ajax-Anwendungen entwickeln
- Inkl. Ant, Refactoring, Debugging, Subversion, CVS, Plug-ins



Auf einen Blick

Einleitung	11
1 Hands on Eclipse	17
2 Arbeiten mit Eclipse	53
3 Arbeitsbereiche und Projekte	117
4 Funktionen mit Plug-ins erweitern	167
5 Fehlersuche und Test	205
6 Versionsverwaltung	253
7 Erstellung einer Anwendung mit grafischer Benutzeroberfläche	299
8 Web- und AJAX-Anwendungen	343
A Literaturverzeichnis	385
B Die Begleit-DVD	387
Index	389

Inhalt

Einleitung	11
------------------	----

1 Hands on Eclipse 17

1.1	Java und Eclipse installieren	17
1.1.1	Installation von Java	18
1.1.2	Installation von Eclipse	21
1.1.3	Der erste Start	24
1.2	Das erste eigene Projekt	28
1.2.1	Ein neues Projekt anlegen	28
1.2.2	Ein erster Blick auf die Projektverwaltung	36
1.3	Ein Rundgang durch die IDE	40
1.3.1	Die Hilfsfunktionen von Eclipse	40
1.3.2	Verfügbare Java-Laufzeitumgebungen anzeigen und bearbeiten	49
1.4	Zusammenfassung	52

2 Arbeiten mit Eclipse 53

2.1	Perspektiven, Sichten und Editoren	53
2.1.1	Die Workbench	54
2.1.2	Sichten	57
2.1.3	Editoren	61
2.1.4	Perspektiven	64
2.2	Java-Programme eingeben und bearbeiten	70
2.2.1	Einstellungen vornehmen	70
2.2.2	Der Java-Editor	74
2.2.3	Navigation	80
2.2.4	Komfortabel arbeiten	86
2.3	Suchen, Ersetzen und Refactoring	100
2.3.1	In und nach Dateien suchen	100
2.3.2	Suchen und Ersetzen im Quelltext	105
2.3.3	Refactoring	108
2.4	Zusammenfassung	114

3	Arbeitsbereiche und Projekte	117
3.1	Der Arbeitsbereich	117
3.1.1	Arbeitsbereiche anlegen und wechseln	118
3.1.2	Im Arbeitsbereich abgelegte Informationen	120
3.1.3	Verknüpfte Ressourcen	124
3.2	Die Projektverwaltung	126
3.2.1	Verschiedene Arten von Projekten	126
3.2.2	Projekte verwalten	131
3.2.3	Das Menü Project	134
3.3	Komplexe Projekte	142
3.3.1	Der Build Path	142
3.3.2	Bibliotheken	146
3.3.3	Launch Configurations	153
3.4	Ant und externe Tools	157
3.4.1	Ant	157
3.4.2	Externe Tools	163
3.5	Zusammenfassung	165
4	Funktionen mit Plug-ins erweitern	167
4.1	Plug-ins aus Anwendersicht	168
4.1.1	Manuelle Installation von Plug-ins	168
4.1.2	Installation über den Update Manager	169
4.1.3	Plug-ins verwalten	174
4.2	Die technische Infrastruktur	177
4.2.1	Die Eclipse-Plattform	177
4.2.2	Features, Plug-ins und Fragmente	179
4.3	Eigene Plug-ins entwickeln	181
4.3.1	Das Hello World-Plug-in	181
4.3.2	Editoren der Perspektive Plug-in Development	184
4.3.3	Sichten der Perspektive Plug-in Development	192
4.4	Eclipse RCP-Anwendungen	195
4.4.1	Ein kleines Beispiel	195
4.4.2	Branding und Verteilung	199
4.5	Zusammenfassung	204
5	Fehlersuche und Test	205
5.1	Visuelles Debuggen	206
5.1.1	Ein erstes Beispiel	206

5.1.2	Die Sichten der Perspektive Debug	210
5.2	Konzepte und Techniken	214
5.2.1	Architektur des Eclipse Debuggers	215
5.2.2	Breakpoints	219
5.3	Fortgeschrittene Debug-Techniken	229
5.3.1	Bedingte Programmunterbrechungen	229
5.3.2	Kontrollierte Einzelschrittverarbeitung	233
5.3.3	Änderungen vornehmen	240
5.4	Unit-Tests	243
5.4.1	JUnit im Überblick	244
5.4.2	Weitere JUnit-Funktionen	249
5.5	Zusammenfassung	251

6 Versionsverwaltung 253

6.1	Lokale Repositories aufsetzen	254
6.1.1	CVS einrichten	254
6.1.2	Subversion einrichten	258
6.2	CVS-Unterstützung	263
6.2.1	Mit Repositories arbeiten	263
6.2.2	Mit lokalen Kopien arbeiten	273
6.3	Zugriff auf Subversion-Repositories	277
6.3.1	Subclipse installieren und einrichten	277
6.3.2	Daten einchecken	280
6.3.3	Mit dem Repository arbeiten	285
6.3.4	Unterschiede analysieren und behandeln	290
6.4	Arbeiten im Team	294
6.4.1	Die Perspektive Team Synchronizing	295
6.4.2	Weitere Team-bezogene Funktionen	296
6.5	Zusammenfassung	297

7 Erstellung einer Anwendung mit grafischer Benutzeroberfläche 299

7.1	GUI-Editoren	300
7.1.1	Ein Überblick	300
7.1.2	Jigloo	303
7.1.3	Die Aufgabenverwaltung Do it!	307
7.2	Das Hauptfenster	309
7.2.1	Die Menüleiste	309
7.2.2	Der Darstellungsbereich	315

7.3	Die Dialoge	318
7.3.1	Aufgaben erfassen	318
7.3.2	Die übrigen Dialoge	328
7.4	Die Teile verbinden	334
7.4.1	Die Implementierung von Do It!	334
7.4.2	Die Anwendung paketieren	339
7.5	Zusammenfassung	341
8	Web- und AJAX-Anwendungen	343
8.1	Java-Web-Anwendungen mit Eclipse	344
8.1.1	Die Web Standard Tools	344
8.1.2	J2EE Standard Tools	353
8.2	Das Google Web Toolkit	360
8.2.1	Funktionsweise und Installation	360
8.2.2	Eine eigene Anwendung	363
8.2.3	Cypal Studio for GWT	368
8.3	Web- und AJAX-Frameworks	373
8.3.1	Die Rich Ajax Platform	373
8.3.2	AJAX Toolkit Framework	377
8.4	Zusammenfassung	382
	Anhang	383
A	Literaturverzeichnis	385
B	Die Begleit-DVD	387
	Index	389

In diesem Kapitel mache ich Sie mit wichtigen Arbeitstechniken unter Eclipse vertraut. Sie lernen Perspektiven, Sichten und Editoren kennen und erfahren, wie Sie die Eingabe und Bearbeitung Ihrer Java-Programme komfortabel und effizient gestalten.

2 Arbeiten mit Eclipse

Im vorangehenden Kapitel habe ich zahlreiche Aspekte der Arbeit mit Eclipse gestreift. Mein Ziel war, Ihnen einen möglichst schnellen Start zu ermöglichen. Denn nichts ist ermutigender als ein erstes Erfolgserlebnis. Allerdings setzt die routinierte Arbeit mit der IDE ein profundes Verständnis ihrer Kernideen und Konzepte voraus. Dieses Wissen möchte ich Ihnen in diesem Kapitel vermitteln.

Im ersten Abschnitt, *Perspektiven, Sichten und Editoren*, stelle ich Ihnen Perspektiven, Sichten und Editoren vor und zeige Ihnen, wie diese Konzepte in der Workbench ineinander greifen. Anschließend widme ich mich dem Java-Editor. Sie lernen dessen vielfältige Komfortfunktionen kennen, die die Eingabe und Pflege von umfangreichen Projekten erst möglich machen. Der dritte Abschnitt, *Suchen, Ersetzen und Refactoring*, erläutert wichtige Hilfsmittel zur Suche in Quelltexten, Dateien und auf Workspace-Ebene. Außerdem zeige ich Ihnen, wie Sie schnell und unkomplizierte Quelltextbereiche umstrukturieren, Bezeichner ändern und neue Elemente einführen.

2.1 Perspektiven, Sichten und Editoren

Bei Ihren ersten Experimenten haben Sie mit einer Reihe von sogenannten *Sichten* gearbeitet. Für ein erstes Verständnis hatte ich diese mit Unterfenstern eines Programm-Hauptfensters verglichen. Lange Zeit empfahl Microsoft eine solche, oftmals *Multiple Document Interface* (MDI) genannte, Darstellungsform als Benutzeroberfläche für Windows-Programme. Allerdings hinkt mein Vergleich. *Sichten* sind nämlich, wie Sie später noch sehen werden, normalerweise nicht beliebig auf dem Bildschirm oder innerhalb des Hauptfensters verschiebbar. Außerdem ist Eclipse keine MDI-Anwendung. In Zusammenhang mit den sogenannten *Perspektiven* bilden *Sichten* und *Editoren* aber das Kontrollzentrum für die Arbeit mit der IDE.

2.1.1 Die Workbench

In früheren Versionen war Eclipse in erster Linie eine durch Plug-ins erweiterbare Java-IDE. Mit Version 3 wurde die sogenannte *Eclipse Rich Client Platform* als mächtiger Rahmen für die Entwicklung von *Rich Client-Anwendungen* zum neuen Fundament. Mithilfe dieser Basis lassen sich alle möglichen Arten von Programmen realisieren. Die Java-IDE ist also, wenn Sie so möchten, nur eine Anwendung dieses Frameworks. Mit der Technik der Eclipse Rich Client Platform werde ich mich ausführlicher in Kapitel 4, *Funktionen mit Plug-ins erweitern*, beschäftigen. Wichtig ist an dieser Stelle der Begriff *Workbench*, weil er ein grundlegendes Instrument der Architektur beschreibt. Die Workbench integriert alle Bestandteile einer Anwendung innerhalb eines Hauptfensters. Wenn Sie Eclipse starten, wird *eine* Workbench angezeigt. Mit WINDOW • NEW WINDOW öffnen Sie weitere.

Elemente der Workbench

Neben der Menüleiste enthält die Workbench sogenannte *Perspektiven*. Eine *Perspektive* wiederum beinhaltet *Editoren* und *Sichten*. Als Sie Eclipse das erste Mal gestartet haben, war die *Java*-Perspektive aktiv und zeigte genau eine *Sicht*, nämlich den *Willkommensbildschirm*. Welche Perspektive gerade geöffnet ist, können Sie übrigens der Titelzeile des Hauptfensters entnehmen; der Fenstertitel beginnt nämlich mit dem Namen der Perspektive. Mithilfe der *Schnellzugriffsleiste* im rechten oberen Bereich der Workbench, die Sie in Abbildung 2.1 sehen, können Sie neue Perspektiven öffnen und zwischen bereits geöffneten umschalten. Sie erkennen die aktive Perspektive an einer anderen Hintergrundfarbe sowie an einem Rahmen um ihren Namen.

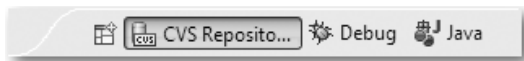


Abbildung 2.1 Schnellzugriff auf geöffnete Perspektiven

Eine Perspektive entspricht also einer Gruppe von Sichten und Editoren innerhalb eines Workbench-Fensters. Dieses wiederum kann eine oder mehrere Perspektiven enthalten. Jede Perspektive eines Fensters kann unterschiedliche Sichten beinhalten, allerdings teilen sich alle Perspektiven die gleiche Menge an Editoren. Hierzu ein Beispiel: Sie können den *Package Explorer* sowohl in der Perspektive *Java* als auch in *Debug* öffnen. Schließen Sie diese Sicht in der einen Perspektive, hat dies keinen Einfluss auf die andere. Anders verhält es sich mit einer geöffneten Java-Quelltextdatei. Diese wird in einem *Editor* angezeigt. Schließen Sie ihn, wirkt sich dies auf alle Perspektiven des entsprechenden Workbench-Fensters aus.

Sichten und Editoren sind visuelle Komponenten der Workbench. Sichten werden in der Regel verwendet, um Eigenschaften anzuzeigen oder durch baumarartige Strukturen zu navigieren. Änderungen in einer Sicht werden sofort gespeichert. Im Gegensatz dazu folgen Editoren dem Prinzip *Öffnen – Speichern – Schließen*. Sie müssen den Speichervorgang also explizit veranlassen und können im Gegenzug das ungewollte Übernehmen von Änderungen verhindern. Sichten und Editoren haben einige Gemeinsamkeiten. Sie können beispielsweise *aktiv* oder *inaktiv* sein. Innerhalb der Workbench gibt es allerdings stets nur ein aktives Element. Sie erkennen es an seiner hervorgehobenen Titelzeile. In Abbildung 2.2 ist beispielsweise der *Package Explorer* aktiv, die Sicht *Hierarchy* hingegen nicht. Die englischsprachige Eclipse-Dokumentation verwendet übrigens den Begriff *part*, wenn sie Editoren *oder* Sichten meint.

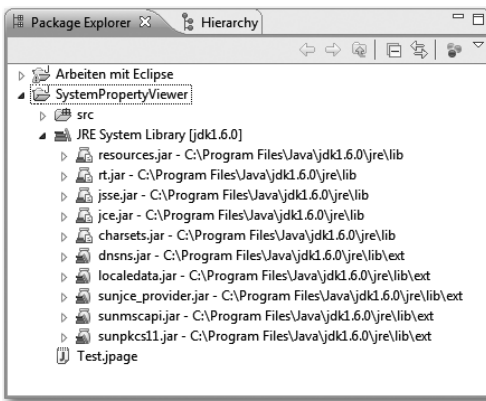


Abbildung 2.2 Der Package Explorer

Aktive Elemente sind Ziel gebräuchlicher Aktionen wie *Ausschneiden*, *Kopieren* oder *Einfügen*. Und sie bestimmen den Inhalt der *Statuszeile* des Workbench-Fensters. Ist beispielsweise der Java-Editor der aktive *part*, zeigt die Statuszeile unter anderem die Cursor-Position an. Wenn Sie hingegen den *Package Explorer* aktivieren, zeigt sie das dort zuletzt angeklickte Element an.

Darstellungsbereiche

In der Einleitung dieses Kapitels habe ich geschrieben, dass Sichten keine echten Fenster sind, weil sie sich nicht frei positionieren lassen. Die Workbench stellt aber eine Reihe von sogenannten *Darstellungsbereichen* (engl. *panes*) zur Verfügung, an die Sichten und Editoren angedockt werden können. Die Größe einer Sicht oder eines Editors richtet sich also nach der Größe des Bereichs, in dem die Komponente abgelegt wurde. Sie positionieren eine Sicht oder einen Editor, indem Sie ihre/seine Titelzeile anklicken und bei gedrückter Maustaste im Work-

bench-Fenster umherfahren. Der Bereich, der die zu verschiebende Komponente beim Loslassen der Maustaste aufnehmen wird, ist mit einem dicken Rahmen gekennzeichnet. Abbildung 2.3 zeigt das Verschieben der Sicht *Hierarchy*.



Abbildung 2.3 Positionieren von Sichten und Editoren

Während des Verschiebevorgangs ändert sich die Form des Mauszeigers. Auf diese Weise zeigt Ihnen Eclipse, an welcher Stelle relativ zum Element unter dem Mauszeiger die zu verschiebende Sicht andocken wird. In Tabelle 2.1 finden Sie eine Aufstellung der Mauszeigerformen und ihrer Bedeutung.








Cursorform	Bedeutung
	Das Element erscheint oberhalb der Sicht/des Editors.
	Das Element wird unterhalb der Sicht eingefügt.
	Die Sicht erscheint linker Hand des Elements unter dem Mauszeiger.
	Das Element wird rechts neben der Sicht andockt.
	Die Sicht/der Editor erscheint als Registerkarte.
	An dieser Stelle kann das Element nicht andockt werden.
	Die Sicht ist nicht andockt.

Tabelle 2.1 Mauszeigerformen beim Verschieben von Sichten und Editoren

Möchten Sie einen Verschiebevorgang abbrechen, müssen Sie vor dem Loslassen der Maustaste nur die Taste `[ESC]` drücken.

Um die Größe eines Darstellungsbereichs zu verändern, fahren Sie mit der Maus an den Rand des Bereichs, den Sie vergrößern oder verkleinern möchten. Wenn eine Größenanpassung an dieser Stelle möglich ist, zeigt Ihnen dies Eclipse durch eine entsprechende Änderung der Mauszeigerform an.

Sie haben in diesem Abschnitt den Zusammenhang zwischen der Workbench, Sichten und Editoren kennengelernt. In den folgenden beiden Abschnitten beschäftige ich mich ausführlicher mit den beiden Gruppen visueller Komponenten und gebe Ihnen Informationen zum Stapeln von Sichten sowie zur Arbeit mit Darstellungsbereichen.

2.1.2 Sichten

Sichten werden häufig verwendet, um die Informationen in der Workbench zu ordnen oder zu gruppieren. Beispielsweise stellt der *Package Explorer* die Dateien und Elemente eines Java-Projekts als baumartige Struktur dar. Andere Sichten enthalten Detail-Informationen zu Objekten innerhalb eines Editors. So können Sie im Quelltext einer Klasse nach Doppelklick auf einen Klassennamen und Drücken der Taste **[F4]** in der Sicht *Hierarchy* die Vererbungshierarchie dieser Klasse ablesen.

Stapeln von Sichten

Sichten können das einzige Element einer pane sein oder aber diesen Darstellungsbereich mit anderen Sichten teilen. Die englische Dokumentation nennt diese Präsentationsform mittels Registerkarten *tabbed notebook*. Mehrere Sichten werden hierbei in einer pane gestapelt. In Abbildung 2.4 sehen Sie die drei Sichten *Package Explorer*, *Hierarchy* und *Outline* als Registerkarten.

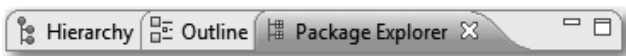


Abbildung 2.4 Drei Sichten als tabbed notebook

Die beiden Symbole am rechten Rand der Abbildung gehören übrigens nicht unmittelbar zu den Sichten, sondern sind streng genommen Bedienelemente des Darstellungsbereichs. Mit ihnen minimieren bzw. maximieren Sie seine Größe. Auch Editoren bieten entsprechende Funktionen an.

Das Kontextmenü von Sichten

Sichten stellen in der Regel zwei Menüs zur Verfügung. Das *Kontextmenü*, das in Abbildung 2.5 zu sehen ist, erreichen Sie durch einen Klick mit der rechten Maustaste auf den Namen der Sicht. Mit ihm kontrollieren Sie seine Größe und

Position. Praktisch ist es, mittels `MOVE • TAB GROUP` alle Sichten des Darstellungsbereichs gleichzeitig verschieben zu können.

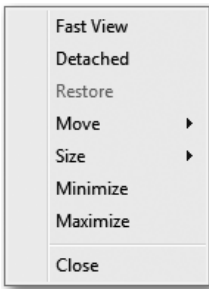


Abbildung 2.5 Das Kontextmenü einer Sicht

Mit dem Menüpunkt `DETACHED` lassen sich Sichten von der Workbench entkoppeln. Abbildung 2.6 zeigt den *Package Explorer* als frei schwebendes Fenster.

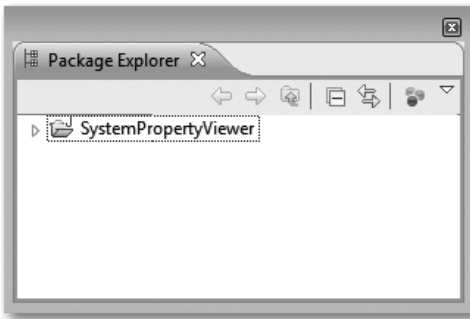


Abbildung 2.6 Der Package Explorer im nicht angedockten Zustand

Sichten erscheinen stets über dem Eclipse-Hauptfenster, können dessen Inhalt also verdecken. Um die Sicht wieder an einen Darstellungsbereich anzudocken, müssen Sie nur den mit einem Häkchen versehenen Menüpunkt `DETACHED` erneut anklicken. Das Fenster wird daraufhin geschlossen und die Sicht erscheint wieder in dem Darstellungsbereich, zu dem sie vor dem Entkoppeln gehört hat.

Klappmenüs

Das zweite Menü einer Sicht wird im Englischen *view pull-down menu* genannt. Im Gegensatz zum eben vorgestellten Kontextmenü enthalten diese *Klappmenüs* Sicht-spezifische Befehle. Die zur Verfügung stehenden Menüpunkte variieren also von Sicht zu Sicht. Sie beziehen sich normalerweise auf die Sicht als Ganzes oder auf alle angezeigten Informationen, nicht jedoch auf einzelne Elemente.

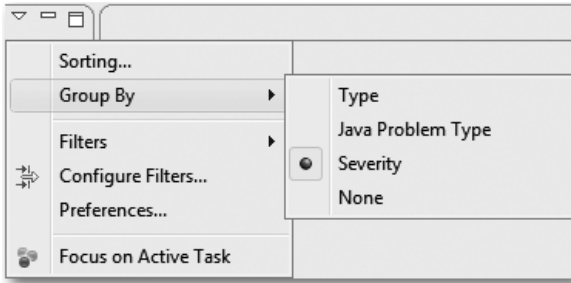


Abbildung 2.7 Klappenmenü der Sicht Problems

Typische Beispiele sind Filter-, Sortier- sowie Gruppierungsfunktionen. Abbildung 2.7 zeigt das Klappenmenü der Sicht *Problems*. Um es zu öffnen, klicken Sie bitte auf das Symbol mit dem nach unten zeigenden Dreieck, das sich links neben den Elementen zum Minimieren bzw. Maximieren der Sicht befindet.

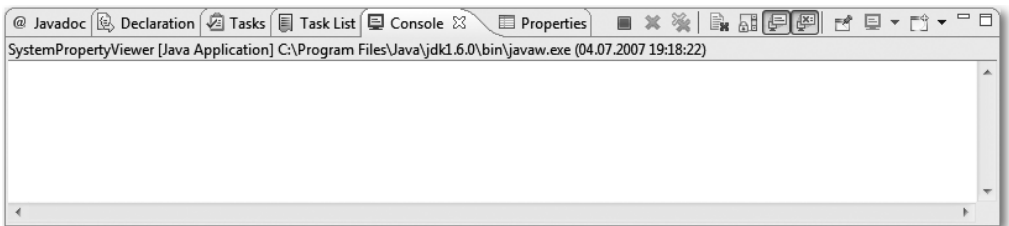


Abbildung 2.8 Die Sicht Console

Neben den Menüs lassen sich Sichten auch über Symbole steuern. Welche Symbole zur Verfügung stehen, variiert von Sicht zu Sicht. Abbildung 2.8 zeigt beispielsweise die Sicht *Console* mit ihren zahlreichen Symbolen. Grundsätzlich werden diese linker Hand der beiden Symbole zum Minimieren bzw. Maximieren der Sicht sowie, falls vorhanden, dem Symbol zum Öffnen des Klappenmenüs dargestellt. Falls der Platz hierfür nicht ausreicht, rutschen die Symbole der Reihe nach in eine neue Zeile. Falls ein Darstellungsbereich mehrere gestapelte Sichten enthält, werden die Symbole der jeweils aktiven Sicht angezeigt.

Fast Views

Sie werden sehr bald merken, dass Sie mit bestimmten Sichten sehr häufig arbeiten. Deshalb ist es wichtig, auf diese möglichst schnell zugreifen zu können. Allerdings wird es mit zunehmender Anzahl geöffneter Sichten schnell eng und unübersichtlich auf dem Bildschirm. Eclipse begegnet diesem Dilemma mit den sogenannten *Fast Views*.

Sie können jede Sicht zu einer solchen *Fast View* machen, indem Sie den Menüpunkt *Fast View* im Kontextmenü der Sicht anklicken. Die Sicht wird daraufhin ausgeblendet und ihr Symbol erscheint in der Leiste *Fast Views*, die sich standardmäßig im linken Bereich der Statuszeile am unteren Rand der Workbench befindet.



Abbildung 2.9 Die Symbolleiste Fast Views

Sie können die in Abbildung 2.9 gezeigte Symbolleiste übrigens an andere Bereiche des Workbench-Fensters andocken. Klicken Sie hierzu auf die gepunktete Linie am linken Rand der Leiste, halten die linke Maustaste gedrückt und verschieben Sie sie an ihre neue Position. Noch bequemer ist ein Klick auf die rechte Maustaste. Er öffnet das in Abbildung 2.10 gezeigte Kontextmenü *DOCK ON*, mit dem Sie ebenfalls die neue Position festlegen können.

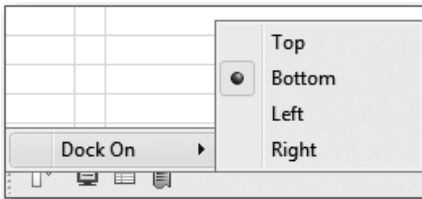


Abbildung 2.10 Neue Position der Symbolleiste Fast Views festlegen

Wie aber funktionieren Fast Views? Sobald Sie auf das Symbol einer Fast View klicken, wird die entsprechende Sicht geöffnet. Sie können nun wie gewohnt mit ihr arbeiten. Um die Sicht wieder verschwinden zu lassen, können Sie erneut auf das Symbol in der Fast Views-Leiste klicken, eine andere Fast View aktivieren oder das Symbol zum Minimieren der Sicht anwählen.

Ob eine Fast View von links nach rechts oder von unten nach oben eingeblendet wird, können Sie mit dem Untermenü *ORIENTATION* des Fast View-Kontextmenüs einstellen. Dieses Menü öffnen Sie durch Klick mit der rechten Maustaste auf das Symbol der entsprechenden Sicht in der Fast Views-Leiste. Sie können die Breite oder Höhe (je nachdem, welche Ausrichtung Sie eingestellt haben) einer Fast View übrigens wie die Darstellungsbereiche ändern.

Mithilfe der Fast Views können Sie Ihr Workbench-Fenster also übersichtlicher gestalten. Nicht benötigte Sichten bleiben ausgeblendet, sind aber mit nur einem Mausklick erreichbar.

2.1.3 Editoren

In diesem Abschnitt möchte ich Ihnen die *Editoren* als weitere visuelle Komponenten der Workbench vorstellen. Wie Sie gleich sehen werden, haben auch diese einige Schmankerl zu bieten, die den Bedienkomfort für Sie als Entwickler deutlich erhöhen. Generell verwenden Sie Editoren, um Ressourcen zu bearbeiten oder zu durchsuchen. Anders als bei Sichten werden Änderungen in einem Editor nicht unmittelbar gesichert, sondern erst nach expliziter Aufforderung durch den Benutzer. Sie erkennen nicht gesicherte Modifikationen an einem * vor dem Titel des betreffenden Editors. Ein weiterer, wesentlicher Unterschied ist, dass Sie mehrere Instanzen eines Editors öffnen können.

Stapeln von Editoren

Wie Sichten können Editoren einen Darstellungsbereich für sich alleine haben oder sich diesen mit anderen Editoren teilen. Dabei kommen die Ihnen schon bekannten *Registerkarten* zum Einsatz.

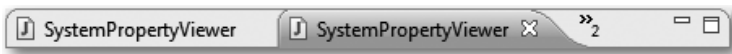


Abbildung 2.11 Vier gestapelte Editoren

Abbildung 2.11 zeigt zwei solcher Editoren sowie die beiden zum Darstellungsbereich gehörenden Symbole zum Minimieren und Maximieren des Darstellungsbereichs. Durch Anklicken des Editor-Titels und Festhalten der Maustaste können Sie Editoren in andere Bereiche der Workbench verschieben. Hierbei zeigt Ihnen Eclipse durch unterschiedliche Mauszeigerformen, wo Sie den Editor andocken können. Eine Aufstellung, was die unterschiedlichen Symbole bedeuten, finden Sie im Abschnitt *Darstellungsbereiche*. Sie können die Verschiebefunktion übrigens auch dazu nutzen, die Reihenfolge der Registerkarten Ihren Wünschen entsprechend zu sortieren. Dies funktioniert nicht nur bei Editoren, sondern auch bei Sichten.

Haben Sie das Symbol neben der rechten Registerkarte in Abbildung 2.11 bemerkt? Wenn die Breite eines Darstellungsbereichs nicht ausreicht, um alle Registerkarten anzuzeigen, werden die rechts liegenden Reiter verdeckt. Die Zahl im Innern des Symbols gibt an, wie viele Editoren momentan nicht zu sehen sind.

Ein Klick auf dieses Symbol öffnet ein Menü, das Sie in Abbildung 2.12 sehen. Mit ihm können Sie gezielt zu einem Editor wechseln. Halbfette Einträge kennzeichnen sichtbare Editoren. Die Eingabezeile im oberen Randbereich des Menüs ist übrigens ein Filter. Tippen Sie beispielsweise ein S, verschwinden alle Ein-

träge, die nicht mit diesem Buchstaben beginnen. Das Symbol steht übrigens auch bei Sichten zur Verfügung.

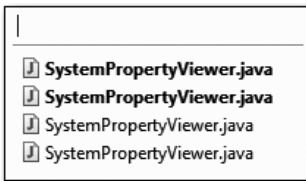


Abbildung 2.12 Auswählen von Registerkarten

Das Kontextmenü von Editoren

Auch Editoren bieten ein Kontextmenü an, das sich nach einem Klick mit der rechten Maustaste auf den Titel eines Editors öffnet. Wie Sie in Abbildung 2.13 sehen, sind viele Funktionen identisch mit denen des Kontextmenüs von Sichten.

Allerdings können Sie mehr Einfluss auf das Schließen von Editoren nehmen. Sehr praktisch ist beispielsweise, alle außer dem aktiven Editor schließen zu können. Äußerst nützlich ist auch **NEW EDITOR**. Mit dieser Funktion *klonen* Sie den aktuellen Editor. Sie bearbeiten dieselbe Datei, können in den beiden Editoren aber an unterschiedlichen Positionen arbeiten. So behalten Sie auch in großen Dateien den Überblick, weil Sie nicht ständig zwischen zwei weit auseinander liegenden Bereichen hin und her scrollen müssen. Allerdings ist es hierbei ratsam, den zweiten Editor in einen anderen Darstellungsbereich zu verschieben, um nicht zwischen den beiden Registerkarten wechseln zu müssen.

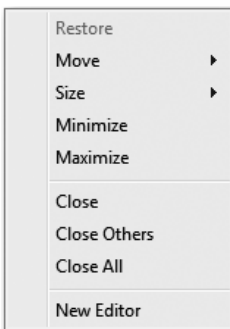


Abbildung 2.13 Das Kontextmenü eines Editors

Ich habe stets »den Editoren« geschrieben, obwohl Sie bisher nur mit einem, nämlich dem Java-Editor gearbeitet haben. Im folgenden Abschnitt zeige ich Ihnen, welche anderen Editoren die IDE zur Verfügung stellt.

Interne und externe Editoren

Bisher haben Sie Dateien stets im Rahmen eines Projekts bearbeitet. Da dessen Klassen im *Package Explorer* zu sehen sind, müssen Sie sich nicht merken, wo Sie die Dateien physikalisch abgelegt haben. Allerdings kann es notwendig sein, einen Quelltext zu editieren, der keinem Projekt zugeordnet wurde oder dessen Projekt Sie geschlossen haben. Mit dem Menüeintrag FILE • OPEN FILE können Sie solche Dateien öffnen.

Um diese Funktion auszuprobieren, schließen Sie bitte zunächst das Projekt *SystemPropertyViewer* im *Package Explorer*, falls Sie es noch geöffnet haben. Wählen Sie nun FILE • OPEN FILE und suchen Sie die Datei *SystemPropertyViewer.java* in der Dateiauswahl. Sie müssen hierzu in den *Arbeitsbereichsordner* wechseln, den Sie beim allerersten Start von Eclipse angegeben haben. Das Projekt *SystemPropertyViewer* ist ein Unterverzeichnis von ihm. Dies wiederum enthält *src*. Hierin befinden sich die Quelltexte des Projekts. Denken Sie bitte daran, dass die Java-Paket-Struktur auf Dateisystemebene durch Unterverzeichnisse abgebildet wird. Das Paket `systempropertyviewer` entspricht also dem Verzeichnis *systempropertyviewer*.

Ein kleiner Tipp: Falls Sie sich nicht mehr an den Ablageort des Arbeitsbereichsordners erinnern können, brechen Sie die Dateiauswahl zunächst ab. Rufen Sie anschließend FILE • SWITCH WORKSPACE • OTHER auf. Sie sehen daraufhin den Dialog *Workspace Launcher*, der den Speicherort preisgibt.

Haben Sie *SystemPropertyViewer.java* ausgewählt, wird die Datei im Ihnen bereits bekannten Java-Editor angezeigt. Auf diese Weise können Sie jede beliebige Datei öffnen. Sofern Eclipse deren Inhalt anhand der Dateieindung oder des MIME-Typs erkennt, wird ein passender interner Editor verwendet. In allen anderen Fällen wird ein externes Programm gestartet, das mit dem Dateityp verknüpft ist. Bitte probieren Sie dies aus, indem Sie eine beliebige Grafik oder Bilddatei mittels FILE • OPEN FILE laden.

Windows-Nutzer können unter Umständen von einem besonderen Schmankerl profitieren. Gestattet die aufgerufene Anwendung die Einbettung mittels *Object Linking and Embedding (OLE)*, wird das Dokument in einem Arbeitsbereich der Workbench angezeigt. Sofern Sie beispielsweise *Word* oder *Excel* installiert haben, werden die Dokumente dieser Anwendungen also auf die gleiche Weise wie Java-Quelltexte eingebunden.

Nachdem Sie nun Editoren und Sichten im Detail kennengelernt haben, stelle ich Ihnen im folgenden Abschnitt die sogenannten *Perspektiven* ausführlicher vor.

2.1.4 Perspektiven

Wie Sie bereits wissen, bestehen *Perspektiven* aus Editoren und Sichten. Es liegt auf der Hand, dass sich aus der Auswahl der vorhandenen Sichten ein bestimmter Blickwinkel auf die mithilfe der Workbench dargestellten Informationen ergibt. Hierzu ein Beispiel: *Package Explorer*, *Problems* und *Tasks* werden Sie vor allem während der Eingabe von Java-Programmen benötigen. Denn der bereits bekannte *Package Explorer* zeigt Ihnen die Struktur Ihrer Anwendung und die Sicht *Problems* weist auf Fehler wie fehlende oder nicht auffindbare Importe sowie Syntaxfehler im Quelltext hin. Hingegen sind *Console*, *Breakpoints* und *Variables* vor allem während der Fehlersuche interessant, weil Sie mit diesen Sichten einen Einblick in das gerade ablaufende Programm erhalten. Perspektiven helfen Ihnen also bei Ihrer Arbeit, indem sie diejenigen Sichten ausblenden, die für die Bewältigung einer bestimmten Aufgabe unnötig sind. Anders formuliert: Während Ihrer Arbeit schalten Sie immer auf diejenige Perspektive um, die für eine bestimmte Aufgabe besonders gut geeignet ist.

Perspektiven öffnen und schließen

Der Zugriff auf *Perspektiven* erfolgt über das Menü WINDOW, das Sie in Abbildung 2.14 sehen. Mittels OPEN PERSPECTIVE können Sie bei Bedarf neue Perspektiven öffnen. Falls die benötigte Perspektive nicht im Menü angezeigt wird, klicken Sie bitte auf *Other* und wählen im Dialog *Open Perspective*, den Abbildung 2.15 zeigt, die gewünschte aus.

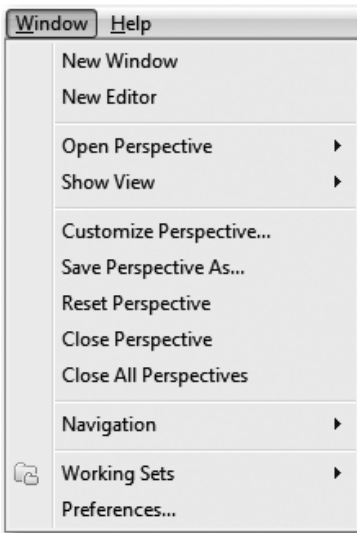


Abbildung 2.14 Das Menü Window

Mit der *Schnellzugriffsleiste* im rechten oberen Bereich der Workbench schalten Sie zwischen bereits geöffneten Perspektiven um. Sie sehen diese Leiste in Abbildung 2.15. Die Leiste kennzeichnet die derzeit aktuelle Perspektive durch eine andere Hintergrundfarbe sowie einem Rahmen um den Namen der Perspektive. Mit **WINDOW • CLOSE PERSPECTIVE** schließen Sie diese. Um alle Perspektiven zu schließen, verwenden Sie **WINDOW • CLOSE ALL PERSPECTIVES**.

Einige Funktionen sind übrigens nicht nur über das Menü **WINDOW** erreichbar. Wenn Sie in der Schnellzugriffsleiste für Perspektiven mit der rechten Maustaste auf den Namen einer Perspektive klicken, öffnet sich ein Kontextmenü, das Sie in Abbildung 2.16 sehen. Es bietet unter anderem die Möglichkeit, die Schnellzugriffsleiste in andere Bereiche der Workbench zu verschieben, sie zu schließen und sie an die eigenen Bedürfnisse anzupassen. Ich werde auf diesen Punkt gleich noch ausführlicher eingehen. Außerdem können Sie die Namen der Perspektiven ausblenden, indem Sie das Häkchen vor **SHOW TEXT** durch Anklicken entfernen. Um sie wieder anzuzeigen, müssen Sie nur diese Funktion erneut aufrufen.

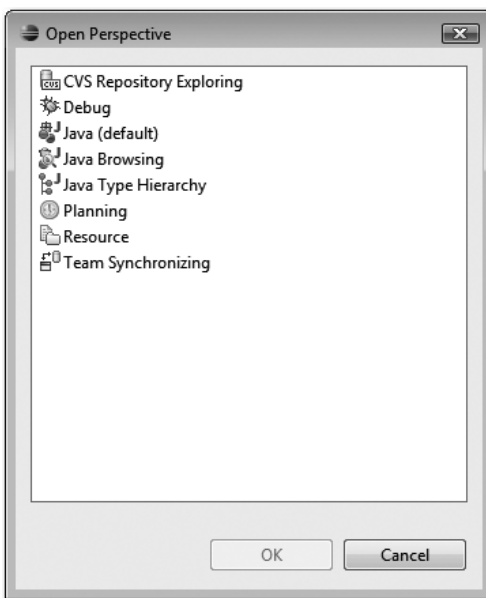


Abbildung 2.15 Dialog zum Auswählen einer Perspektive

In jeder Perspektive sind bestimmte Sichten standardmäßig geöffnet. Ferner hat jede Perspektive ein Grundlayout. Es legt fest, welcher Darstellungsbereich welche Sicht(en) beinhaltet, wie groß diese Bereiche sind und wo innerhalb der Workbench sie angeordnet werden. Der Kontextmenübefehl **RESET** stellt diesen Zustand jederzeit wieder her.

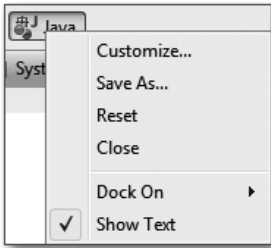


Abbildung 2.16 Das Kontextmenü einer Perspektive

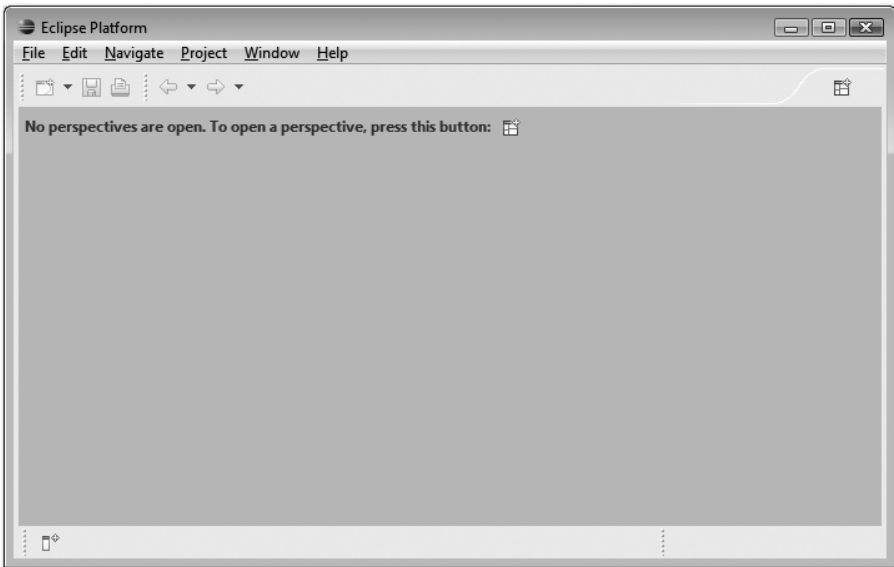


Abbildung 2.17 Das Workbench-Fenster ohne geöffnete Perspektiven

Bitte machen Sie sich nun mit dem Öffnen und Schließen von Perspektiven vertraut, indem Sie einige Perspektiven öffnen und schließen. Probieren Sie hierbei auch die Funktion *Close All Perspectives* aus. Ihr Workbench-Fenster sollte dann Abbildung 2.17 entsprechen.

Eigene Perspektiven erstellen

Wenn Sie Änderungen am Layout (also Lage und Größe der Darstellungsbereiche) einer Perspektive vornehmen, bleiben diese auch bei nachfolgenden Programmstarts wirksam. Wie Sie bereits wissen, können Sie beliebig viele weitere Sichten öffnen. Auch diese werden bei späteren Sitzungen automatisch geöffnet. Beides gilt allerdings nur, bis Sie eine Perspektive schließen. Nach dem erneuten

Öffnen sind die zusätzlichen Sichten ebenso wie Änderungen am Layout verschwunden. Stattdessen gelten wieder die von mir eben angesprochenen Grundeinstellungen. Zu diesen können Sie auch im laufenden Betrieb zurückkehren, indem Sie **WINDOW • RESET PERSPECTIVE** oder **RESET** im Kontextmenü der Perspektive aufrufen und die anschließende Rückfrage bestätigen.

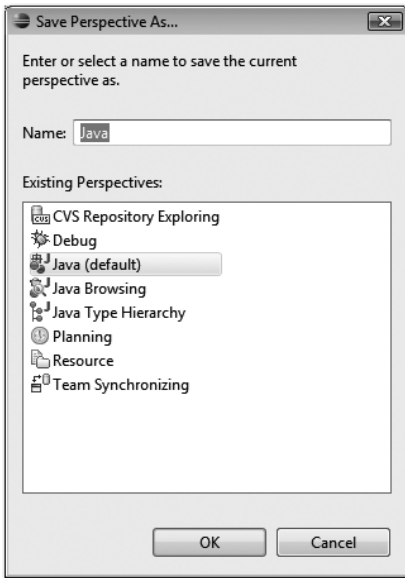


Abbildung 2.18 Dialog zum Speichern von Perspektiven

Was aber, wenn Sie Anpassungen an einer Perspektive dauerhaft speichern möchten? Haben Sie sich beispielsweise an die Sicht *Tasks* gewöhnt, ist es lästig, sie nach dem Öffnen einer Perspektive von Hand aufrufen zu müssen. In diesem Fall ist **WINDOW • SAVE PERSPECTIVE AS** äußerst nützlich. Mit dieser Funktion können Sie nämlich benutzerdefinierte Perspektiven erzeugen sowie Änderungen an Standard-Perspektiven dauerhaft sichern.

Um dies zu testen, wechseln Sie bitte in die *Java*-Perspektive oder öffnen Sie diese. Fügen Sie der Perspektive nun eine Sicht Ihrer Wahl hinzu und positionieren diese nach Belieben auf der Workbench. Öffnen Sie anschließend den in **Abbildung 2.18** gezeigten Dialog *Save Perspective As*, indem Sie **WINDOW • SAVE PERSPECTIVE AS** aufrufen.

Sie haben die Möglichkeit, Ihrer Perspektive einen eigenen Namen zu geben oder eine bestehende zu überschreiben. Klicken Sie hierzu bitte in der Liste der existierenden Perspektiven auf *Java (default)* und schließen den Dialog durch Anklicken von **OK**. Eclipse wird Sie nun darauf aufmerksam machen, dass eine Per-

spektive mit dem eingegebenen Namen bereits existiert. Klicken Sie auf *Yes*, um das Überschreiben dieser Perspektive zu bestätigen.

Rufen Sie den Dialog *Save Perspective As* bitte erneut auf, wählen dieses Mal allerdings einen eigenen, nicht existierenden Namen, zum Beispiel *meine Java Perspektive*. Sie werden feststellen, dass der Name sofort in der Schnellzugriffsleiste erscheint. Außerdem wird Ihre eigene Perspektive im Dialog *Open Perspective* aufgeführt, den Sie durch Anklicken von **WINDOW • OPEN PERSPECTIVE • OTHER** erreichen. Löschen können Sie eigene Perspektiven übrigens im Dialog *Preferences* in dessen Zweig **GENERAL • PERSPECTIVES**.

Sie können nun eigene Perspektiven erstellen sowie Änderungen an bestehenden vornehmen. Allerdings haben wir uns bisher auf Sichten sowie das Layout innerhalb der Workbench beschränkt. Eclipse bietet jedoch viel weiterreichende Einflussmöglichkeiten auf Perspektiven. Mit diesen möchte ich mich nun beschäftigen.

Perspektiven anpassen

Die Wahl einer Perspektive hat nicht nur Einfluss darauf, wie Ihnen Informationen präsentiert werden, sondern auch, welche Funktionen in der Menüleiste sowie der *Toolbar* am oberen Rand der Workbench verfügbar sind. Ihnen ist vielleicht aufgefallen, dass der Inhalt der Menüs **WINDOW • OPEN PERSPECTIVE** und **WINDOW • SHOW VIEW** abhängig von der gerade aktiven Perspektive ist. Welche Menüpunkte zu sehen sind, stellen Sie im Dialog *Customize Perspective* ein, den Sie in Abbildung 2.19 sehen.

Auf dessen Registerkarte *Shortcuts* gibt es den Bereich *Submenus*, in dem Sie den Menüs **NEW**, **OPEN PERSPECTIVE** und **SHOW VIEW** Elemente hinzufügen können. Jedes dieser Elemente ist einer Kategorie zugeordnet. Hierzu ein Beispiel: Für das Menü **SHOW VIEW** gibt es unter anderem die Kategorie *Java*, zu der beispielsweise die Elemente *Package Explorer*, *JUnit* und *Javadoc* gehören. Standardmäßig sind *Declaration*, *Javadoc* und *Package Explorer* mit einem Häkchen versehen. Die entsprechende Sicht erscheint also, wenn Sie das Menü in der Menüleiste aufklappen. **JUNIT** hingegen ist nicht zu sehen. Um auch diese Sicht anzuzeigen, müssen Sie das gleichnamige Häkchen setzen.

Neben Menüs beeinflussen Sie mit dem Dialog *Customize Perspective* auch, welche Elemente in der *Toolbar* am oberen Rand der Workbench sowie in der Menüleiste angezeigt werden. Wechseln Sie hierzu bitte auf die Registerkarte *Commands*, die Sie in Abbildung 2.20 sehen.

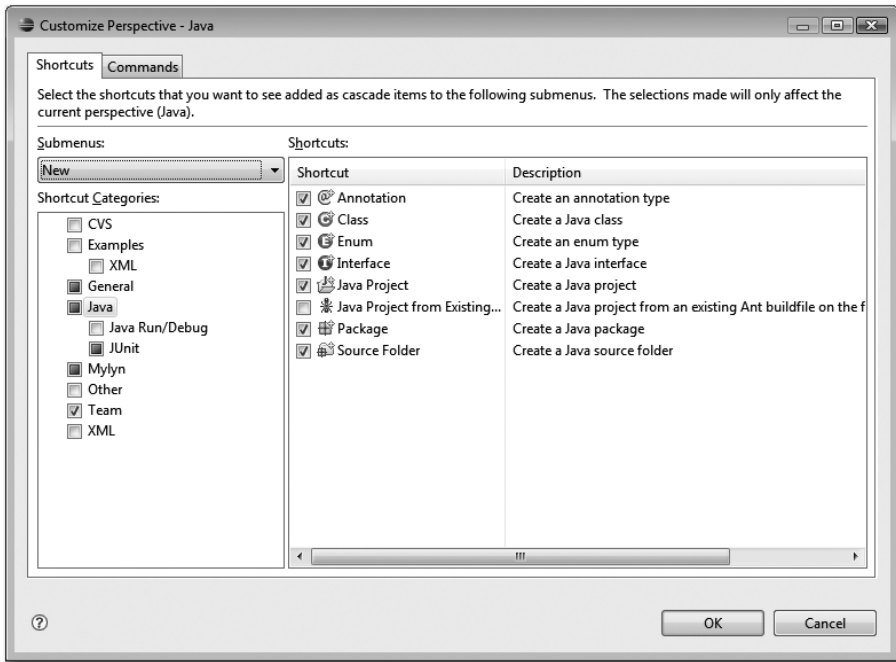


Abbildung 2.19 Dialog zum Anpassen von Perspektiven

Sie ist in die drei Bereiche *Available command groups*, *Menubar details* und *Toolbar details* unterteilt. *Command groups* fassen Menüpunkte und Toolbar-Elemente zusammen, die eine Sinneinheit ergeben. Beispielsweise gehören zu *Team* alle Menüeinträge und Symbole, die sich auf Funktionen zur Arbeit im Team beziehen. Standardmäßig ist diese command group deaktiviert. Um sie einzuschalten, setzen Sie bitte ein Häkchen vor *Team*.

Durch Anklicken einer command group legen Sie also fest, ob die Ihr zugeordneten Menü- und Toolbareinträge angezeigt werden, falls die betreffende Perspektive aktiv ist. Das gezielte Ein- oder Ausblenden bestimmter Elemente einer solchen Gruppe ist übrigens nicht möglich.

Nach dieser Einführung in die Bedeutung und Funktionsweise von Perspektiven wende ich mich im nächsten Abschnitt dem Java-Editor zu. Ich zeige Ihnen, wie Sie ihn an Ihre Bedürfnisse anpassen und wie Sie seine fortgeschrittenen Funktionen nutzen, um effizient Java-Quelltexte einzugeben und zu bearbeiten.

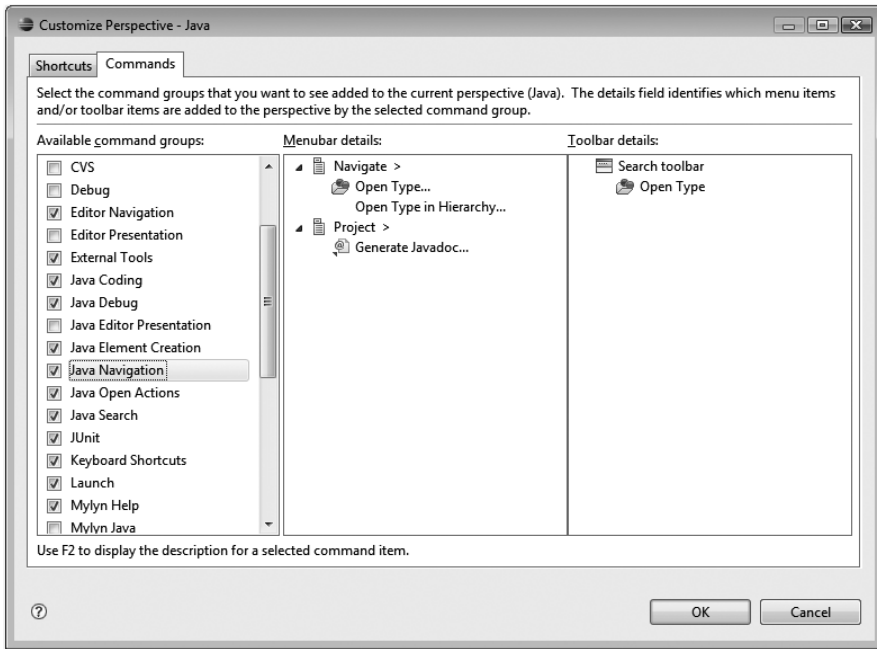


Abbildung 2.20 Registerkarte Commands des Dialogs Customize Perspective

2.2 Java-Programme eingeben und bearbeiten

Die Eingabe und Pflege von Java-Quelltexten bildet einen Schwerpunkt Ihrer Arbeit mit Eclipse. Aus diesem Grund ist ein routinierter Umgang mit dem Java-Editor wichtig. Die hierfür notwendigen Grundlagen möchte ich Ihnen in diesem Abschnitt vermitteln. Wichtig ist aber, dass Sie die hier erlernten Funktionen kontinuierlich einsetzen und üben.

2.2.1 Einstellungen vornehmen

Sie haben in Kapitel 1, *Hands on Eclipse*, bereits ein Java-Programm geschrieben. Die ersten Schritte mit dem Java-Editor liegen also schon hinter Ihnen. Aus diesem Grund zeige ich Ihnen im Folgenden, wie Sie ihn an Ihre Bedürfnisse anpassen können.

Zeilennummern einblenden

Wie Sie wissen, blendet Eclipse am unteren Rand der Workbench eine Statuszeile ein, die in Abbildung 2.21 zu sehen ist. In der Standardeinstellung enthält sie unter anderem die sogenannten *Fast Views*, die Sie aus Abschnitt 2.1.2, *Sichten*,

kennen. Außerdem können Sie die *Cursor-Position* ablesen, sofern ein Editor geöffnet und aktiv ist. Die erste Zahl gibt in diesem Fall die aktuelle *Zeilennummer* an. Die zweite nennt die Spalte, in der sich der Cursor gerade befindet.



Abbildung 2.21 Statuszeile mit Fast Views und Anzeige der Cursor-Position

Noch mehr Übersicht erhalten Sie, wenn Sie die Zeilennummern unmittelbar im Eingabebereich anzeigen lassen.

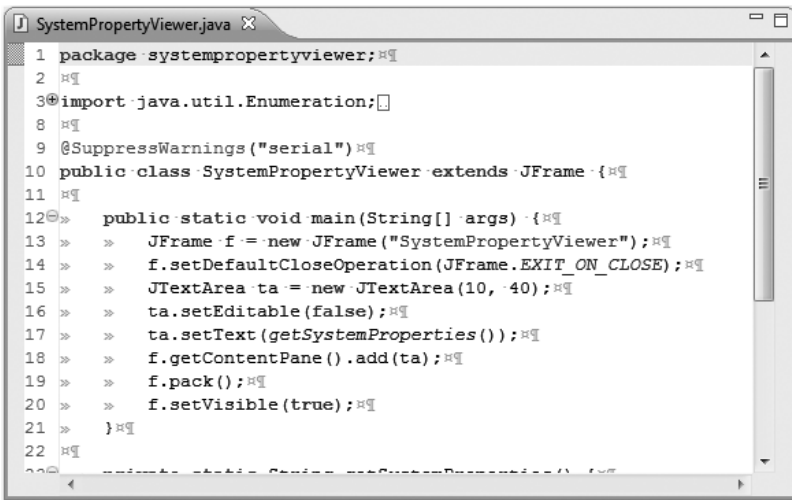


Abbildung 2.22 Java-Editor mit Zeilennummern und nicht druckbaren Zeichen

Öffnen Sie hierzu bitte den Dialog *Preferences* und navigieren Sie zum Knoten GENERAL • EDITORS • TEXT EDITORS. Setzen Sie bitte jeweils ein Häkchen vor *Show Line Numbers* und *Show whitespace characters*. Wie Sie in Abbildung 2.22 sehen, erleichtert die zweite Funktion das Auffinden von nicht druckbaren Zeichen, beispielsweise Tab-Stopps und Zeilenumbrüchen. Um während der Eingabe zu einer bestimmten Zeile zu springen, rufen Sie bitte NAVIGATE • GO TO LINE auf oder drücken Sie die Tasten `Strg`-`L`.

In Folgenden möchte ich Ihnen zeigen, wie Sie die Formatierung des Quelltextes an Ihre Bedürfnisse anpassen.

Formatierung des Quelltextes

Es gibt unzählige Meinungen und Konventionen, wie ein Java-Quelltext formatiert werden sollte. Dies beginnt bei relativ einfachen Fragestellungen, etwa, ob

die Tab-Weite zwei oder vier Leerzeichen entspricht. Gerne diskutiert wird auch, ob die öffnenden geschweiften Klammern von Methodenrümpfen in der Zeile der Methodensignatur oder alleine für sich stehen müssen.

Wenn Sie keine Mitstreiter haben, ist es letztlich eine Frage des Geschmacks, für welche der unzähligen Varianten Sie sich entscheiden. Anders verhält es sich bei der Arbeit im Team. Hier sind feste Vorgaben unerlässlich. Zum einen ist es ärgerlich, ständig sich ändernde Schreibweisen vorzufinden. Zum anderen behindert es die Arbeit mit Werkzeugen zur Versionsverwaltung. Für solche Tools macht es nämlich einen großen Unterschied, wenn sich die Lage von geschweiften Klammern ändert. Sie als Entwickler haben dann mit vermeintlichen Unterschieden zu kämpfen, die letztlich gar keine sind.

Wie üblich nehmen Sie solche Einstellungen im Dialog *Preferences* vor. Navigieren Sie hierzu bitte zu *JAVA • CODE STYLE • FORMATTER*. Die Formatierung geschieht auf der Basis sogenannter *Profile*. Wie sich ein solches Profil auf den Quelltext auswirkt, können Sie in der Vorschau unterhalb der Klappliste *Active profile* in Abbildung 2.23 sehen. Um ein Gefühl für die Möglichkeiten zu bekommen, probieren Sie bitte die vorhandenen Profile aus.

Wenn Sie Änderungen vornehmen möchten, überlegen Sie bitte zunächst, welches der vorhandenen Profile Ihren Vorstellungen am ehesten entspricht. Klicken Sie anschließend auf die Schaltfläche *New*, woraufhin sich der Dialog *New Profile* öffnet. Er ist in Abbildung 2.24 zu sehen. Nachdem Sie einen Profilnamen vergeben und das Profil, auf dem ihre Einstellungen basieren sollen, ausgewählt haben, können Sie den Dialog mit *OK* schließen. Achten Sie bitte darauf, dass *Open the edit dialog now* mit einem Häkchen versehen ist.

Auf den ersten Blick wirkt der Dialog zum Bearbeiten eines Profils in Abbildung 2.25 sicher beeindruckend. Aber bedenken Sie, dass Ihr eigenes Profil auf einer soliden Grundlage aufbaut, sodass Sie vermutlich nur an wenigen Einstellungen Änderungen vornehmen werden. Ein paar davon möchte ich Ihnen nun zeigen.

Die von mir angesprochene Tab-Breite, also wie viele Leerzeichen einem Tab-Stopp entsprechen, stellen Sie auf der Registerkarte *Indentation* ein. Als *Tab Policy* wählen Sie *Tabs only*. Der Wert, den Sie bei *Tab size* eingeben, wird nach kurzer Verzögerung automatisch in der Vorschau dargestellt. Probieren Sie dies ruhig aus. Wie Klammern angeordnet werden, stellen Sie auf der Registerkarte *Braces* ein. Um beispielsweise die Klammern in der Zeile unter dem ersten Modifier einer Methodensignatur erscheinen zu lassen, wählen Sie für *Method declaration* den Wert *Next line*.

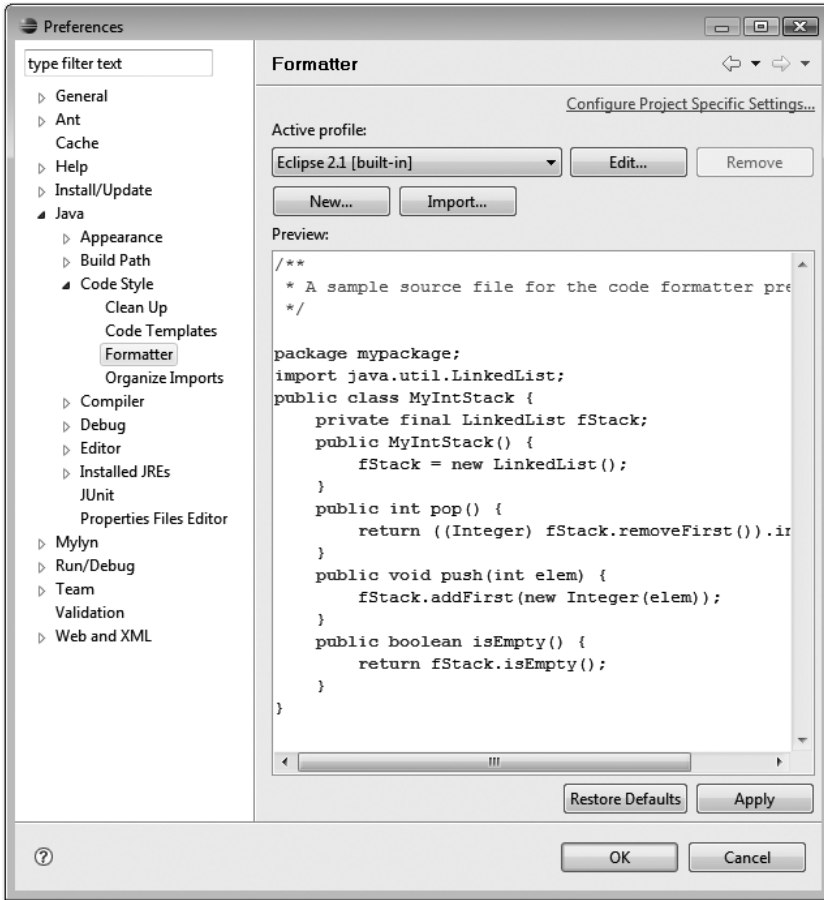


Abbildung 2.23 Die Seite Formatter des Dialogs Preferences

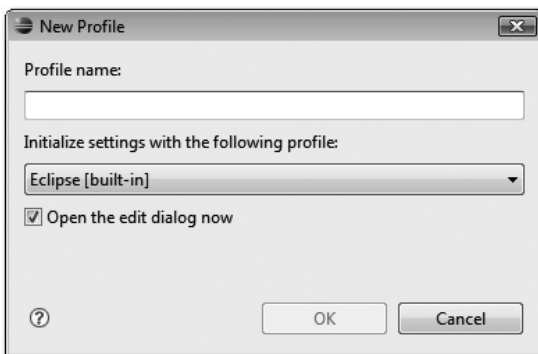


Abbildung 2.24 Dialog zum Anlegen eines neuen Profils

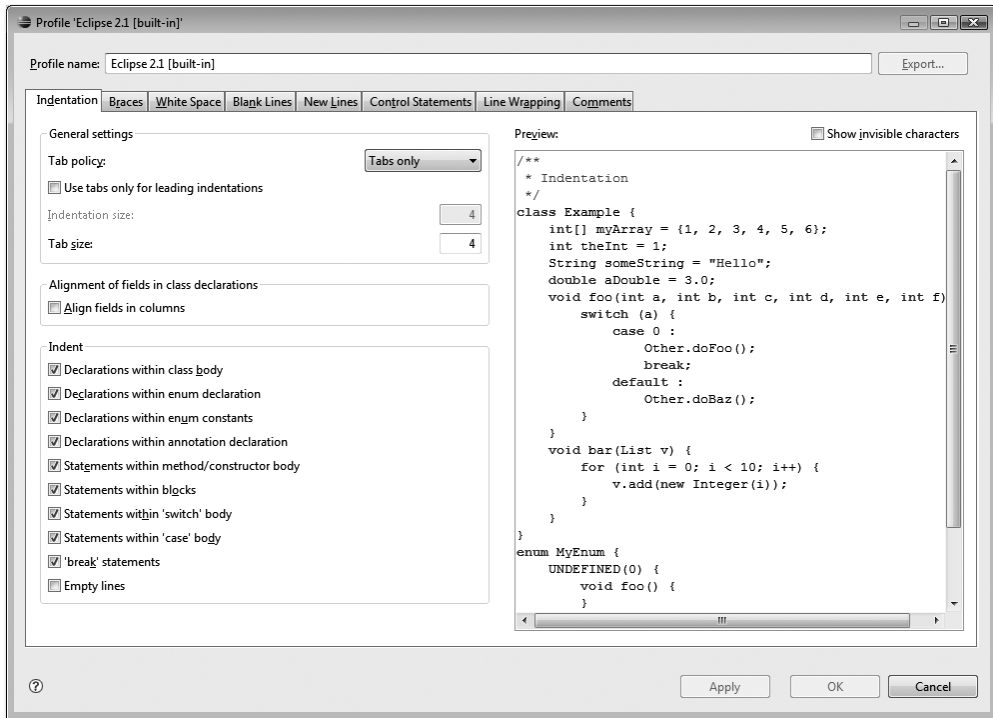


Abbildung 2.25 Dialog zum Bearbeiten eines Profils

Wie so oft gilt auch im Zusammenhang mit Formatierungen das Prinzip »Weniger ist mehr«. Wenn es keine abweichenden firmen- oder teaminternen Konventionen gibt, rate ich Ihnen, die Vorgaben soweit wie möglich beizubehalten.

Und gerade bei der Arbeit im Team ist es wichtig, sich auf ein *gemeinsames* Profil zu verständigen. Sie können es durch Anklicken der Schaltfläche *Export* im Dialog zum Bearbeiten eines Profils exportieren und den Teammitgliedern zur Verfügung stellen. Ihre Kollegen importieren das Profil anschließend, indem sie die Schaltfläche *Import* auf der Seite *Formatter* des Dialoges *Preferences* anklicken.

Um die Formatierungseinstellungen auf Ihren Quelltext anzuwenden, rufen Sie bitte die Menüfunktion `SOURCE • FORMAT` auf oder drücken die Tastenkombination `[Ctrl] + [⇧] + [F]`. Sofern Sie einen Bereich markiert haben, wirkt sich die Formatierung nur auf diesen aus, andernfalls auf der kompletten Quelltext.

2.2.2 Der Java-Editor

Sie haben in Kapitel 1, *Hands on Eclipse*, bereits erste Erfahrungen im Umgang mit dem in Eclipse eingebauten Java-Editor gemacht. Allerdings habe ich dort be-

wusst viele interessante Aspekte ausgeklammert, auf die ich im Folgenden ausführlicher eingehen möchte.

Damit Sie meine Ausführungen am Bildschirm nachvollziehen können, öffnen Sie bitte das Projekt *SystemPropertyViewer* und legen eine leere Klasse *Test* an. Klicken Sie hierzu mit der rechten Maustaste auf die Wurzel des Projekts und wählen dann **NEW • CLASS**. Die im Dialog *New Java Class* voreingestellten Werte können Sie weitgehend beibehalten. Setzen Sie aber bitte ein Häkchen vor *Generate comments* und *public static void main(String [] args)*. Geben Sie als Namen der Klasse *Test* ein und schließen den Dialog mit *Finish*. Die neue Klasse wird im *Package Explorer* angezeigt und in einem Darstellungsbereich geöffnet.

Syntax Highlighting

Wenn Sie einen Blick auf den Quelltext der Klasse *Test* werfen, fällt sofort die unterschiedliche Farbe bestimmter Code-Bestandteile auf. Beispielsweise sind normale Kommentare (`//`) grün, während Javadoc-Kommentare (`/** */`) in verschiedenen Blautönen dargestellt werden. Java-Schlüsselwörter wiederum erscheinen in einem dunklen Rot.

Erwartungsgemäß können Sie diese Farbgebung Ihren Wünschen entsprechend anpassen. Öffnen Sie hierzu bitte den Dialog *Preferences* und navigieren Sie zum Knoten **JAVA • EDITOR • SYNTAX COLORING**. Wie Sie in Abbildung 2.26 sehen, können Sie anhand einer Vorschau prüfen, wie sich Ihre Änderungen auswirken würden.

Wie schon bei der Formatierung des Quelltextes rate ich Ihnen aber auch hier, die Voreinstellungen möglichst beizubehalten. Haben Sie sich an Ihr individuelles Farbschema gewöhnt, wird Ihnen die Darstellung in Standardinstallationen ungewohnt erscheinen.

Eclipse bietet eine Reihe von Funktionen an, die Sie bei der Eingabe von Quelltexten unterstützen. Einige davon möchte ich Ihnen im Folgenden vorstellen.

Unterstützung bei der Texteingabe

Solche Funktionen wurden im Menü **SOURCE** zusammengefasst. Mit *Toggle Comment* können Sie die Zeile, in der sich der Cursor befindet, zu einer Kommentarzeile machen. Rufen Sie die Funktion erneut auf, werden die Kommentarzeichen wieder entfernt. Bitte probieren Sie dies aus, indem Sie den Cursor in die Zeile `// TODO Auto-generated method stub` bewegen und anschließend die Tastenkombination **[Ctrl]+[Z]** drücken. **TOGGLE COMMENT** wirkt übrigens auch auf Bereiche. Jede markierte Zeile wird beim ersten Aufruf mit `//` versehen.

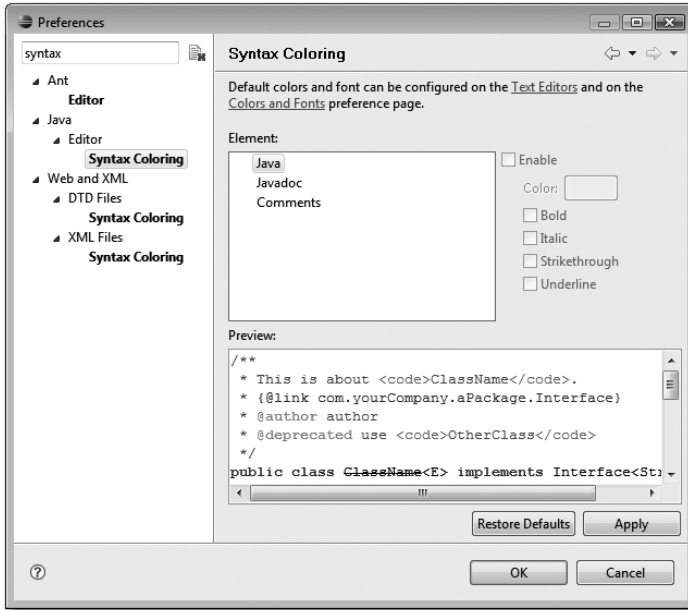


Abbildung 2.26 Die Seite Syntax Coloring des Dialogs Preferences

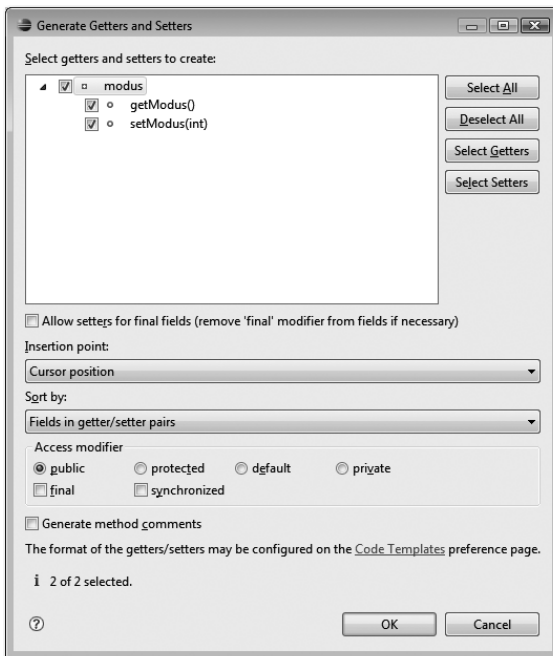


Abbildung 2.27 Dialog zum Anlegen von Gettern und Settern

Ebenfalls äußerst praktisch ist es, bestimmte Elemente wie Konstruktoren, Variablendeklarationen und Methoden mit Kommentarblöcken versehen zu können.

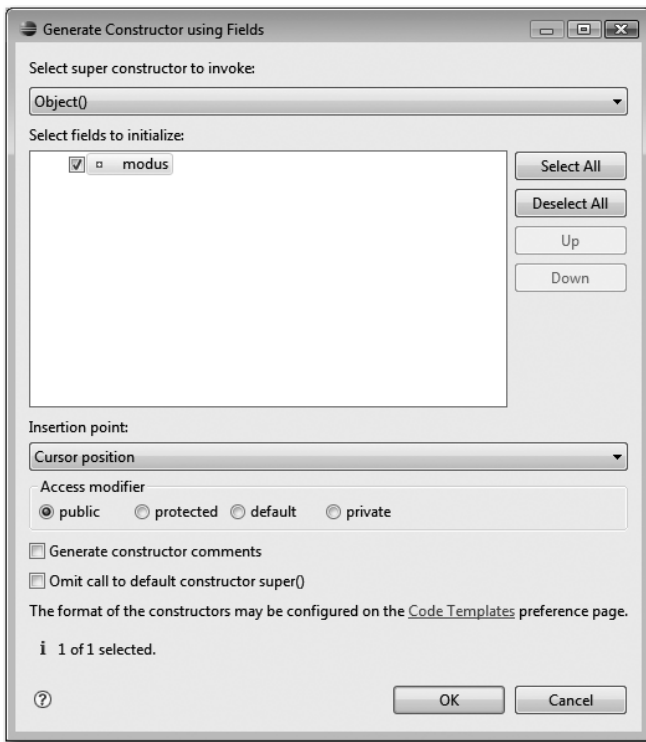


Abbildung 2.28 Dialog zum Erzeugen eines Konstruktors

Hierzu ein Beispiel. Fügen Sie der Klasse *Test* eine Methode `test()` hinzu. Nachdem Sie die öffnende Klammer des Methodenrumpfes getippt haben, drücken Sie bitte die Taste `↵`. Eclipse wird automatisch die schließende Klammer hinzufügen und den Cursor in einer leeren Zeile platzieren. Rufen Sie nun `SOURCE • GENERATE ELEMENT COMMENT` auf oder drücken Sie die Tastenkombination `Alt + ⌘ + J`. Die IDE erzeugt daraufhin oberhalb der Zeile mit der Methodensignatur einen mehrzeiligen Kommentar.

Die Funktionen des Menüs `SOURCE` lassen sich verschiedenen Kategorien zuordnen. Eine, die sich der Behandlung von Kommentaren widmet, haben Sie eben kennengelernt. Die zweite beschäftigt sich mit der Formatierung des Quelltextes. Auch diese kennen Sie bereits. Die dritte Kategorie fasst Funktionen zum automatischen Erzeugen von Methoden und Konstruktoren zusammen. Um sie auszuprobieren, stellen Sie bitte sicher, dass Ihre Klasse *Test* nur noch aus der Klassendeklaration besteht. Alle anderen Bestandteile löschen Sie bitte.

Fügen Sie nun bitte folgende Zeile ein: `private int modus;`. Rufen Sie anschließend bitte `SOURCE • GENERATE GETTERS AND SETTERS` auf. Sie sehen den Dialog *Generate Getters and Setters*, dessen Vorbelegung in etwa Abbildung 2.27 entsprechen sollte. In ihm können Sie einstellen, für welche Variablen entsprechende Methoden erzeugt werden und wo diese im Quelltext platziert werden. Schließen Sie den Dialog bitte durch Anklicken der Schaltfläche `OK`. Eclipse hat der Klasse *Test* die beiden Methoden `getModus()` und `setModus()` hinzugefügt.

Da jede Eigenschaft einer Klasse in deren Konstruktor initialisiert werden sollte, fügen Sie *Test* bitte einen solchen hinzu. Auch hier ist Ihnen die IDE behilflich. Rufen Sie die Funktion `SOURCE • GENERATE CONSTRUCTOR USING FIELDS` auf. Sie sehen den in Abbildung 2.28 gezeigten Dialog *Generate Constructor using Fields*. Da Konstruktoren normalerweise vor Methoden stehen, wählen Sie als *Insertion point* den Wert *First method*. Da *Test* keine explizite Elternklasse braucht, setzen Sie bitte ein Häkchen vor *Omit call to default constructor super()*. Wenn Sie möchten, können Sie einen Kommentar erzeugen lassen, der dem Konstruktor vorangestellt wird. Schließen Sie bitte den Dialog durch Anklicken der Schaltfläche `OK`.

Wie Sie gesehen haben, lassen sich mit Eclipse eine Reihe von häufig wiederkehrenden Aufgaben automatisieren. Hierzu gehört auch das Einfügen von Quelltext-Bestandteilen mittels `SOURCE • SURROUND WITH`. Im folgenden Abschnitt möchte ich Ihnen noch die Funktion *Externalize Strings* vorstellen, die Texte in separate Dateien auslagert.

Texte auslagern

Das Auslagern von Texten in separate Dateien ist vor allem im Hinblick auf die Übersetzung in andere Sprachen sinnvoll, kann aber auch zu einer Reduzierung des benötigten Speicherplatzes führen, indem es der Mehrfachdeklaration von Texten vorbeugt. Außerdem entstehen auf diese Weise übersichtlichere Quelltexte. Um diese Funktion auszuprobieren, fügen Sie der Klasse *Test* bitte die folgende Zeile hinzu:

```
private String meldung = "Herzlich Willkommen";
```

Öffnen Sie nun den in Abbildung 2.29 gezeigten Dialog *Externalize Strings*, indem Sie `SOURCE • EXTERNALIZE STRINGS` aufrufen.

Ausgelagerte Texte werden über sogenannte *Schlüssel* angesprochen. In der Tabelle *Strings to externalize* legen Sie fest, welcher Schlüssel zu welchem Text gehört. Mit den Schaltflächen *Externalize*, *Internalize* und *Ignore* steuern Sie, wie die IDE mit Texten, die sie in Ihrem Quelltext gefunden hat, umgehen soll. Beispielsweise könnten Sie einen bereits ausgelagerten Text wieder in den Quelltext übernehmen oder eine Meldung vom Auslagern ausnehmen.

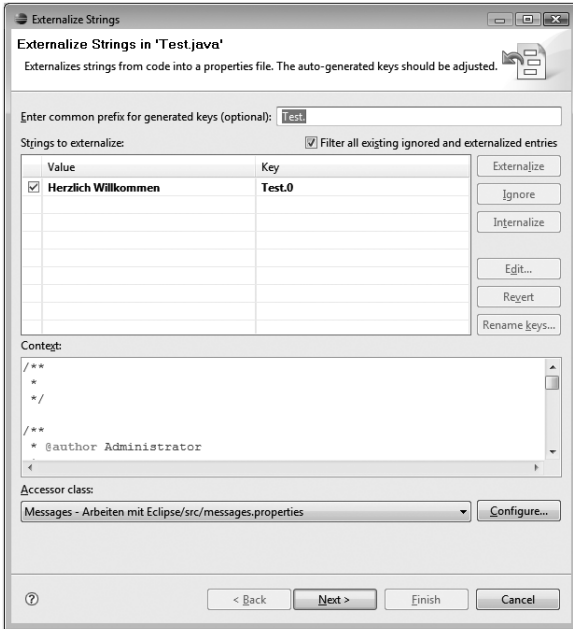


Abbildung 2.29 Dialog zum Auslagern von Texten

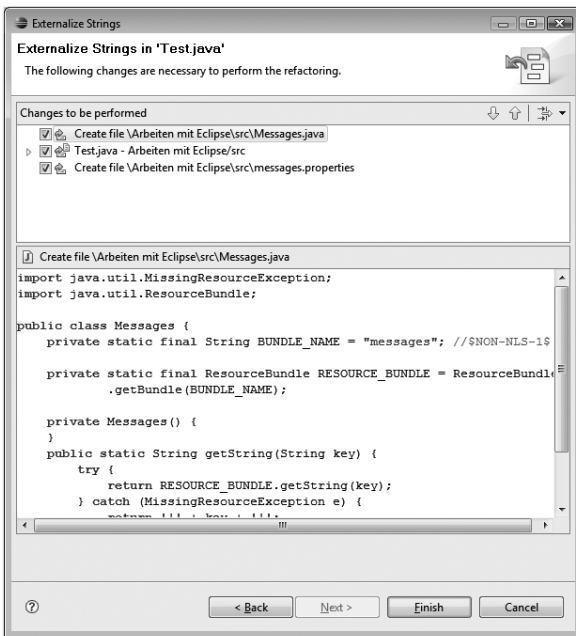


Abbildung 2.30 Vorschau der Änderungen am Quelltext

Wie Sie in Abbildung 2.30 sehen, ist der Text `Herzlich Willkommen` mit einem Häkchen versehen, wird also ausgelagert. Er wird über den Schlüssel `Test.0` angesprochen. Sie können jeden Schlüssel automatisch mit einem Präfix versehen lassen, das standardmäßig dem Namen der Klasse entspricht. Möchten Sie ohne ein Präfix arbeiten, lassen Sie das entsprechende Feld einfach leer. Sie haben auch die Möglichkeit, Schlüsselnamen vollkommen individuell zu vergeben. Klicken Sie nun bitte auf *Next*.

Eclipse wird Sie auf ein Problem hinweisen. Die Datei `messages.properties`, welche die Texte später aufnehmen wird, existiert nämlich noch nicht. Die IDE legt die Datei später jedoch automatisch an. Klicken Sie bitte erneut auf *Next*. Der Dialog *Externalize String* enthält eine Vorschau der Änderungen, die an Ihrem Quelltext vorgenommen werden, wenn Sie auf *Finish* klicken. Abbildung 2.30 zeigt diese Vorschau.

Um zu prüfen, ob die beiden genannten Dateien tatsächlich angelegt wurden, werfen Sie bitte einen Blick in den *Package Explorer*. Die Klasse `Messages` finden Sie im selben Paket wie `Test`. Die Datei `messages.properties` wird als letztes Element des Verzeichnisses `src` angezeigt.

Sie haben bisher zahlreiche Funktionen kennengelernt, die Ihnen dabei helfen, Java-Programme komfortabel einzugeben und zu bearbeiten. Voraussetzung für flüssiges Arbeiten ist aber auch das schnelle Bewegen im Code. Im folgenden Abschnitt beschäftige ich mich deshalb mit der Navigation innerhalb des Quelltextes, aber auch mit dem Umschalten zwischen mehreren Editoren.

2.2.3 Navigation

Lassen Sie mich diesen Abschnitt mit einer Frage beginnen. Was bedeutet eigentlich *Navigation*? Grundsätzlich versteht man darunter das Bewegen im Quelltext mithilfe der Cursortasten bzw. dem direkten Anspringen einer Position durch Anklicken mit der linken Maustaste. Eine weitere Möglichkeit, gezielt zu bestimmten Bereichen eines Programms zu gelangen, habe ich Ihnen mit `NAVIGATE • GO TO LINE` vorgestellt.

Nicht in den Bereich Navigation hingegen fällt das Verschieben des sichtbaren Ausschnitts eines Editors. Denn Sie *sehen* zwar einen anderen Teil Ihres Programms, allerdings passt sich die Cursor-Position nicht an. Sie können dies prüfen, indem Sie einen Blick auf die Statuszeile werfen. Sobald Sie eine beliebige Taste drücken, wird der sichtbare Bereich auch wieder der Cursor-Position angeglichen.

Eine weitere Form der Navigation möchte ich Ihnen nun zeigen. Auch hier geht es um das unmittelbare Anspringen eines Abschnitts im Quelltext. Eclipse bietet Ihnen nämlich zwei Listen, in denen Sie solche Sprungziele definieren können: *Aufgaben* und *Lesezeichen*.

Aufgaben

Im Abschnitt *Ein neues Projekt anlegen* in Kapitel 1, *Hands on Eclipse*, habe ich Ihnen die Sicht *Tasks* bereits kurz vorgestellt. Sie zeigt *Aufgaben*, die entweder automatisch durch den Eclipse-eigenen Java-Compiler oder explizit durch Sie angelegt wurden. Welche Aufgaben das System selbständig erzeugt, steuern Sie über die Seite **JAVA • COMPILER • TASK TAGS** des Dialogs *Preferences*, die in Abbildung 2.31 zu sehen ist. Hierzu weisen Sie einem Schlüsselwort eine Priorität zu. Beispielsweise haben Aufgaben, die das Wort `FIXME` enthalten, eine hohe Priorität, wohingegen `TODO` Aufgaben mit normaler Priorität kennzeichnet.

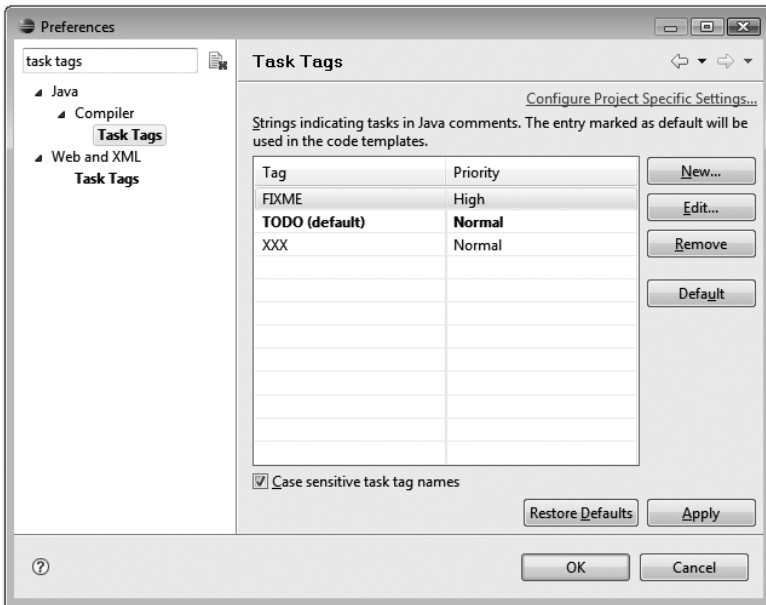


Abbildung 2.31 Dialog zum Anlegen und Bearbeiten von Task Tags

Probieren Sie das Anlegen von Aufgaben durch den Java-Compiler bitte aus, indem Sie in Ihrer Klasse *Test* mehrere Kommentare anlegen, die die Wörter `FIXME` und `TODO` enthalten. Sobald Sie den Quelltext speichern, wird die Sicht *Tasks* entsprechend aktualisiert. Falls sie nicht geöffnet ist, rufen Sie bitte **WINDOW • SHOW VIEW • TASKS** auf. Sie werden bemerken, dass Aufgaben mit hoher Priorität mit einem roten Ausrufezeichen gekennzeichnet werden.

Um zu der Zeile im Quelltext zu navigieren, die eine Aufgabe enthält, können Sie die betreffende Zeile der Sicht doppelklicken. Ein Klick mit der rechten Maustaste hingegen öffnet das Kontextmenü der Aufgabe. Es enthält unter anderem den Menüpunkt **Go To**, der dieselbe Funktion ausführt. Die Sicht *Tasks* ist übrigens unabhängig von einem Editor verwendbar. Sie sehen also auch solche Aufgaben, deren Quelltext im Moment nicht geöffnet ist. Ein Doppelklick öffnet die Datei und positioniert den Cursor an der Stelle, die die Aufgabe enthält. Interessant ist die Funktion **SHOW IN • PACKAGE EXPLORER** des Kontextmenüs, die die betreffende Klasse in der Sicht *Package Explorer* zeigt.

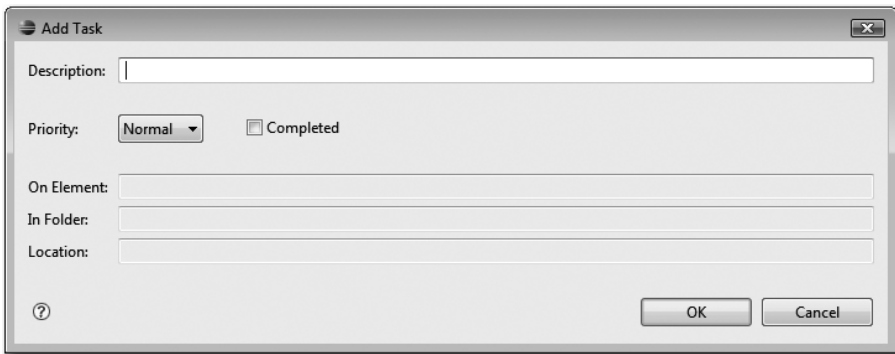


Abbildung 2.32 Anlegen einer neuen Aufgabe

Im Gegensatz zu durch das System erzeugten Aufgaben müssen solche, die Sie selbst anlegen, keinen Bezug zu einem Quelltext haben. Sie können dies testen, indem Sie **ADD TASK** des Kontextmenüs aufrufen. Die Felder *On element*, *In Folder* und *Location* des in Abbildung 2.32 gezeigten Dialogs *Add Task* sind leer und nicht anwählbar. Geben Sie bitte einen beliebigen Aufgabentext ein und schließen den Dialog durch Klick auf die Schaltfläche **OK**. Erwartungsgemäß bleibt ein Doppelklick auf die neue Aufgabe ergebnislos.

Natürlich können Sie auch Aufgaben erstellen, die sich auf eine Zeile Ihres Quelltextes beziehen. Fahren Sie hierzu in den linken (hellgrauen) Randbereich des Editors und drücken die rechte Maustaste. Daraufhin öffnet sich ein Kontextmenü, das unter anderem die Funktion **ADD TASK** enthält. Bitte wählen Sie diesen Eintrag aus. Sie sehen den Ihnen bereits bekannten Dialog *Add Task*. Allerdings sind nun die Felder *On element*, *In Folder* und *Location* mit dem Namen der Datei, deren Verzeichnis sowie der aktuellen Zeilennummer vorbelegt.

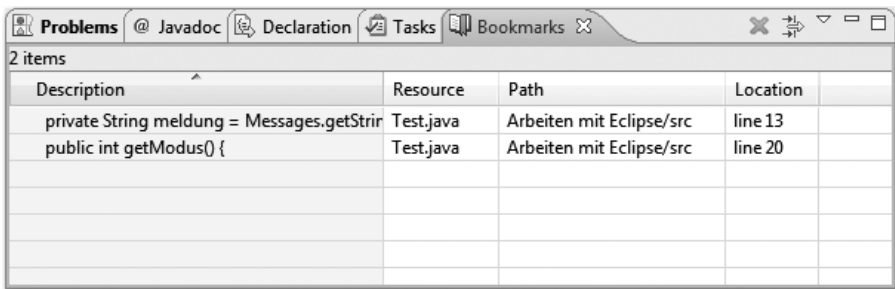
Aufgaben, die Sie selbst angelegt haben, können Sie jederzeit durch Setzen eines Häkchens in der Spalte mit einem Häkchen-Symbol als erledigt markieren. Solche beendeten Aufgaben lassen sich einzeln oder komplett löschen. Öffnen Sie hierzu

das Kontextmenü einer Aufgabe und wählen Sie *Delete* oder *Delete Completed Tasks*. Durch den Eclipse-Compiler erzeugte Aufgaben lassen sich derzeit nur durch Löschen des betreffenden Kommentars entfernen.

Aufgaben sind also eine Art von *Sprungmarken*, mit denen Sie Teile Ihres Quelltextes, an denen Sie noch Korrekturen vornehmen müssen, im schnellen Zugriff haben.

Bookmarks

Auch *Lesezeichen* sind Verweise auf Zeilen im Quelltext und werden wie Aufgaben mit einem Befehl des Kontextmenüs im linken Randbereich des Java-Editors erzeugt (ADD BOOKMARK). Beim Anlegen eines Lesezeichens müssen Sie einen Namen vergeben, der es möglichst knapp beschreiben sollte. Eclipse schlägt hier den Inhalt der betreffenden Zeile (bzw. bei sehr langen Zeilen einen Teil davon) vor. Sie sollten abwägen, ob Sie auf diese Weise das Lesezeichen eindeutig zuordnen können oder gegebenenfalls einen alternativen Namen vergeben.



Description	Resource	Path	Location
private String meldung = Messages.getStrin	Test.java	Arbeiten mit Eclipse/src	line 13
public int getModus() {	Test.java	Arbeiten mit Eclipse/src	line 20

Abbildung 2.33 Die Sicht Bookmarks

Wie auch Aufgaben speichern Lesezeichen eine Zeilennummer, einen Dateinamen sowie das Verzeichnis der Datei. Diese Informationen werden in der Sicht *Bookmarks* in Abbildung 2.33 angezeigt. Um sie zu öffnen, wählen Sie bitte WINDOW • SHOW VIEW • OTHER. Sie sehen nun den Dialog *Show View*. Die Sicht *Bookmarks* finden Sie unterhalb des Knotens *General*. Bitte denken Sie daran, dass Sie Perspektiven speichern können, um häufig benötigte Sichten nicht jedes Mal erneut öffnen zu müssen.

Quelltextzeilen, denen ein Lesezeichen zugewiesen wurde, erkennen Sie an einem blauen Symbol im linken Randbereich des Java-Editors. Um ein Lesezeichen zu entfernen, öffnen Sie dessen Kontextmenü, indem Sie das Symbol mit der rechten Maustaste anklicken und REMOVE BOOKMARK wählen. Alternativ ist dies auch über ein Menü möglich, dass sich nach Rechtsklick auf das Lesezeichen in der Sicht *Bookmarks* öffnet.

Wie aber unterscheiden sich Lesezeichen von Aufgaben? Oder, anders gefragt: Wann nehmen Sie die einen, wann die anderen? Schließlich erlauben beide die Navigation im Quelltext.

Aufgaben helfen Ihnen dabei, Ihre Arbeit zu strukturieren und erinnern Sie an Dinge, die Sie an Ihrem Programm noch ändern müssen. Konsequenterweise erfolgt das Anlegen auch sehr flüssig durch Tippen eines Schlüsselworts innerhalb eines Kommentars. Lesezeichen hingegen sollten Sie einsetzen, wenn Sie oft in unterschiedlichen Bereichen Ihres Quelltextes arbeiten. Denn Lesezeichen sind, anders als die durch Eclipse automatisch erzeugten Aufgaben, nicht an Zeilen mit Kommentar gebunden. Dies trifft zwar auch auf solche Aufgaben zu, die Sie als Entwickler angelegt haben. Allerdings haben Aufgaben durch die ihnen zugewiesene Priorität eindeutig einen temporären Charakter. Irgendwann müssen sie erledigt sein. Lesezeichen hingegen behalten in der Regel ihre Gültigkeit.

Komfortabel navigieren

Wenn Sie mehrere Java-Quelltexte geöffnet haben, können Sie zwischen den verschiedenen Editoren umschalten, indem Sie die betreffende *Registerkarte* anklicken. Allerdings empfinden es viele Entwickler als störend, deshalb von der Tastatur zur Maus wechseln zu müssen. Windows-MDI-Anwendungen kennen für das Umschalten zwischen Programm-Unterfenstern das Tastenkürzel `Ctrl+F6`. Glücklicherweise übernimmt Eclipse diese äußerst praktische Funktion.

Haben Sie also mehrere Editor-Instanzen geöffnet, können Sie zwischen diesen blättern, indem Sie `Ctrl+F6` drücken. Solange Sie die Taste `Ctrl` festhalten, sehen Sie eine Liste vergleichbar mit Abbildung 2.34 die die Namen der geöffneten Dateien zeigt. Drücken von `F6` blättert einen Eintrag weiter, `⇧+F6` in die andere Richtung. Wenn Sie die Taste `Ctrl` loslassen, wird derjenige Editor aktiv, dessen Eintrag mit einem andersfarbigen Hintergrund versehen ist.

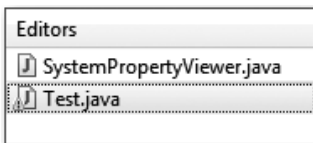


Abbildung 2.34 Liste der geöffneten Editoren

Ein weiteres recht praktisches Tastenkürzel ist übrigens `Ctrl+F4`, das Eclipse ebenfalls von Windows-Anwendungen übernommen hat. Hiermit schließen Sie den derzeit aktiven Editor. Selbstverständlich werden Sie über etwaige nicht gespeicherte Änderungen informiert.

Ich möchte nun einige Befehle ansprechen, die zum Teil über die Menüleiste, zum Teil über die *Toolbar* am oberen Rand der Workbench aufgerufen werden. Welche Funktionen dort verfügbar sind, hängt unter anderem von der aktiven Perspektive ab. Wie Sie bereits wissen, können Sie mittels **WINDOW • CUSTOMIZE PERSPECTIVE** Symbole bzw. Funktionen ein- und ausblenden.

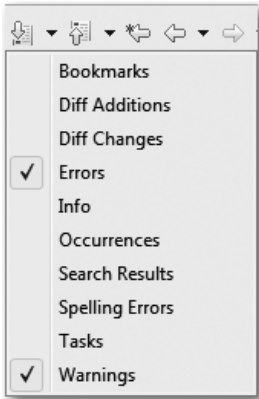


Abbildung 2.35 Liste der zu berücksichtigenden Anmerkungen

Die beiden Symbole **NEXT ANNOTATION** und **PREVIOUS ANNOTATION** helfen Ihnen, zu wichtigen Positionen innerhalb eines Quelltextes zu navigieren. Jeder Klick setzt den Cursor an die jeweils nächste oder vorhergehende geeignete Anmerkung (abhängig davon, in welche Richtung Sie blättern). Welche Anmerkungen Eclipse hierbei berücksichtigen soll, stellen Sie ein, indem Sie auf den kleinen nach unten weisenden Pfeil rechts neben den beiden Symbolen klicken. Sie sehen daraufhin die in **Abbildung 2.35** gezeigte Auswahlliste, in der Sie durch Anklicken beliebig Häkchen setzen und löschen können. Möchten Sie die beiden Symbole **NEXT ANNOTATION** und **PREVIOUS ANNOTATION** beispielsweise dazu verwenden, der Reihe nach alle Lesezeichen Ihrer Quelltexte aufzusuchen, müssen Sie alle Häkchen des Auswahlmenüs mit Ausnahme von **BOOKMARKS** entfernen.

Eine weitere äußerst nützliche Funktion ist **NAVIGATE • LAST EDIT LOCATION**. Sie ist auch über das Tastenkürzel **[Ctrl]-[Q]** sowie das gleichnamige *Toolbar*-Symbol verfügbar. Mit ihr gelangen Sie zu der Stelle im Quelltext, an der Sie zuletzt Änderungen vorgenommen haben.

Ebenfalls über das Menü **NAVIGATE** sowie die *Toolbar* erreichen Sie die beiden Funktionen **BACK** und **FORWARD**. Mit ihnen blättern Sie durch die Ressourcen, die in einem Editor angezeigt werden, wie durch die Seiten in einem Webbrowser. Ein Klick auf den nach unten zeigenden Pfeil öffnet eine Auswahl, in der Sie gezielt zu Dateien springen können.

Im folgenden Abschnitt geht meine Erkundung der IDE weiter. Sie lernen einige wichtige Sichten ausführlicher kennen und verwenden das sogenannte *Scrapbook* sowie die *Local History*.

2.2.4 Komfortabel arbeiten

Sicher haben Sie sich schon einmal gewünscht, eine kurze Befehlssequenz ausprobieren zu können, ohne mühsam eine Klasse anlegen, das Programm übersetzen und starten zu müssen. Schon seit geraumer Zeit gibt es auf www.beanshell.org die von Pat Niemeyer entwickelte *BeanShell*. Diese Scriptsprache mit eingebauter Konsole basiert auf Java-Syntax und erlaubt das unmittelbare Ausführen von Anweisungen ohne den sonst üblichen Eingeben-Übersetzen-Ausführen-Zyklus.

Das Scrapbook

Etwas ganz Ähnliches bietet Eclipse in Form des sogenannten *Scrapbooks*. Mit dessen Hilfe können Sie Java-Anweisungen eingeben und sofort ausführen. Ihnen steht hierbei sogar der Debugger zur Verfügung. Sie können beispielsweise problemlos Variablen inspizieren.

Um das Scrapbook zu verwenden, müssen Sie einem Projekt eine sogenannte *Scrapbook-Seite* hinzufügen. Dies geschieht folgendermaßen: Öffnen Sie das Kontextmenü des Projekts, indem Sie im *Package Explorer* mit der rechten Maustaste auf den Namen des Projekts klicken und **NEW • OTHER** auswählen. Im nun erscheinenden Dialog *New* navigieren Sie zum Knoten **JAVA • JAVA RUN/DEBUG** und klicken *Scrapbook Page* an. Etwas schneller kommen Sie ans Ziel, wenn Sie in der Eingabezeile unterhalb des Wortes *Wizards* den Text *scrap* eintippen. Der Dialog zeigt automatisch den Eintrag *Scrapbook Page*, den Sie nur noch anklicken müssen.

Mit *Next* gelangen Sie auf die zweite Seite des Dialogs, die Sie in Abbildung 2.36 sehen. Bitte geben Sie der Scrapbook-Seite einen Namen, beispielsweise **SCRAPBOOK-TESTSEITE**, und schließen den Dialog, indem Sie die Schaltfläche *Finish* anklicken. Die neu angelegte Datei mit der Endung *.jpage* wird im *Package Explorer* angezeigt und in einem Editor geöffnet.

Bitte geben Sie nun einen beliebigen Java-Ausdruck ein, zum Beispiel:

```
System.out.println("Hallo, Welt");
```

Um ein Quelltext-Fragment auszuführen, müssen Sie es zunächst markieren. Da der Editor bisher nur Ihre eben gemachte Eingabe enthält, geht dies am schnellsten, indem Sie **[Ctrl]-[A]** drücken. Dieses Tastenkürzel entspricht übrigens **EDIT • SELECT ALL**.

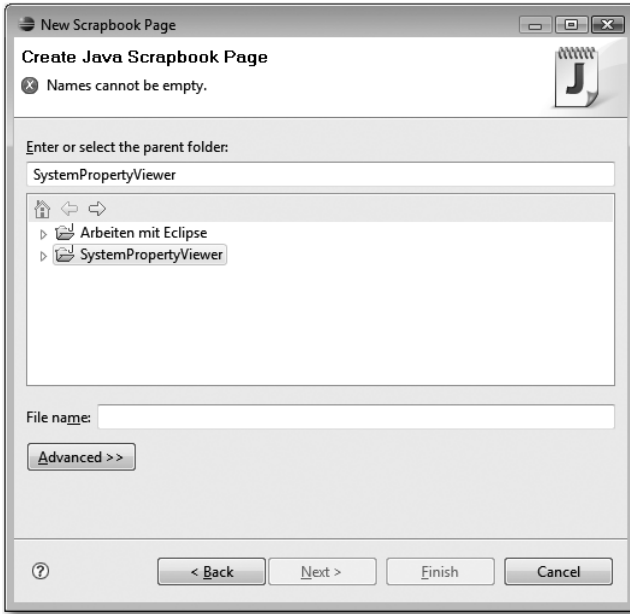


Abbildung 2.36 Anlegen einer Scrapbook-Seite



Abbildung 2.37 Toolbar-Symbole des Scrapbooks



Anschließend klicken Sie bitte auf das Symbol EXECUTE THE SELECTED TEXT, das sich neben einigen weiteren in Abbildung 2.37 gezeigten Symbolen in der Toolbar befindet.

Der Text *Hallo, Welt* wird in der Sicht *Console* ausgegeben. Vereinfacht ausgedrückt sammelt diese alle Ausgaben in die Streams *out* und *err*. Bitte ändern Sie den eben eingegebenen Ausdruck so um, dass die Meldung in den *err*-Stream geschrieben wird und führen Sie die Anweisung erneut aus. Sie werden bemerken, dass die Meldung nun in roter Schrift erscheint. Falls die *Console* nicht sichtbar ist, können Sie diese übrigens mit **WINDOW • SHOW VIEW • CONSOLE** öffnen.

Löschen Sie bitte den Inhalt des Editors und tippen Sie dann `Math.PI`. Nachdem Sie den Text markiert haben, klicken Sie erneut auf EXECUTE THE SELECTED TEXT. Sie werden feststellen, dass der gewünschte Wert nicht in der *Console* ausgegeben wird. Versuchen Sie es nun bitte mit dem Symbol DISPLAY RESULT OF EVALUATING SELECTED TEXT.




Abbildung 2.38 Auswerten eines Ausdrucks

Wie Sie in Abbildung 2.38 sehen, wird der Ausdruck ausgewertet und das Ergebnis dieser Auswertung im Editor eingefügt. Dieser Text ist markiert, kann also durch Drücken der Taste  gelöscht werden. Dies ist praktisch, weil der gesamte Ausdruck nun nicht mehr ohne Fehler ausgewertet werden kann. Bitte probieren Sie es aus, indem Sie zunächst  drücken und anschließend DISPLAY RESULT OF EVALUATING SELECTED TEXT anklicken. Eclipse wird folgende Fehlermeldung einfügen:

```
Syntax error on token "(", ( expected after this token
```

Falls Sie die Ergebnisse der Auswertungen nicht löschen möchten, können Sie am Ende Ihres Ausdrucks einen einzeiligen Kommentar beginnen. Auch hierzu ein Beispiel. Löschen Sie den Inhalt des Editors und tippen Sie anschließend:

```
System.getProperty("user.name") //
```

Bitte drücken Sie nun  und klicken Sie erneut auf das Symbol DISPLAY RESULT OF EVALUATING SELECTED TEXT. Das Ergebnis der Auswertung wird unmittelbar hinter die beiden Kommentarzeichen geschrieben.

Sowohl DISPLAY RESULT OF EVALUATING SELECTED TEXT als auch EXECUTE THE SELECTED TEXT werten Ausdrücke aus. Bei Letzterem wird das Ergebnis der Auswertung verworfen, wohingegen es bei DISPLAY RESULT OF EVALUATING SELECTED TEXT in den Editor übernommen wird. EXECUTE THE SELECTED TEXT bietet sich an, wenn die auszuführende Befehlssequenz entweder keine Ausgaben tätigt oder ohnehin in einen der beiden Ausgabeströme schreibt. Dann sehen Sie das Ergebnis in der Sicht *Console*.

Eine spannende Frage ist, in welcher Laufzeitumgebung das *Scrapbook* eigentlich ausgeführt wird. Wenn Sie im *Package Explorer* mit der rechten Maustaste auf Ihre Scrapbook-Seite klicken und im daraufhin erscheinenden Kontextmenü PROPERTIES auswählen, sehen Sie den in Abbildung 2.39 gezeigten Dialog *Properties for Scrapbook*. Auf dessen Seite *Scrapbook Runtime* können Sie nicht nur das Arbeitsverzeichnis einstellen und Parameter an die virtuelle Maschine übergeben, sondern auch auswählen, welche Java-Version Sie für das Scrapbook verwenden möchten. Sie wird beim Auswerten des ersten Ausdrucks gestartet. Um diese virtuelle Maschine wieder zu beenden, klicken Sie bitte auf das Toolbar-Symbol STOP THE EVALUATION.

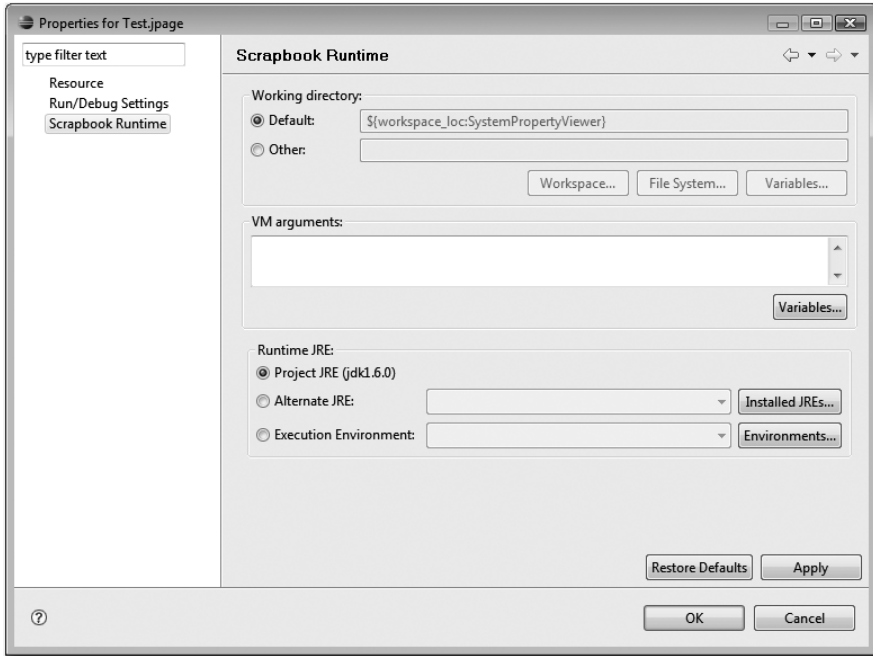


Abbildung 2.39 Eigenschaften einer Scrapbook-Seite

Mit dem Symbol `SETS THE IMPORT DECLARATIONS FOR RUNNING CODE` schließlich legen Sie fest, welche Klassen und Pakete in Ihre Scrapbook-Seite importiert und damit ohne ausdrückliche Nennung eines Paketnamens gefunden werden.

Das Scrapbook bietet eine äußerst komfortable Möglichkeit, schnell und unkompliziert Quelltextfragmente einzugeben und zu testen.

Im nächsten Abschnitt beschäftige ich mich mit der Frage, wie Sie versehentliche Änderungen an Ihren Programmen rückgängig machen können. Neben dem gängigen Undo-Redo-Paradigma bietet Eclipse nämlich einen weiteren praktischen Helfer, die sogenannte *Local History*. Hierbei handelt es sich, wie Sie gleich sehen werden, um eine Art Versionsverwaltung auf Dateiebene.

Undo, Redo und Local History

Beginnen möchte ich diesen Abschnitt allerdings mit den beiden Funktionen `EDIT • UNDO` und `EDIT • REDO`. Sie implementieren die aus vielen Programmen bekannten Funktionen *Zurücknehmen einer Änderung* und *Wiederherstellen der Änderung nach einem Undo*. Um das Verhalten von Eclipse zu testen, öffnen Sie bitte einen beliebigen Editor. Sie können hierzu die Scrapbook-Seite aus dem vorherigen Abschnitt oder beispielsweise die Klasse *Test* verwenden.

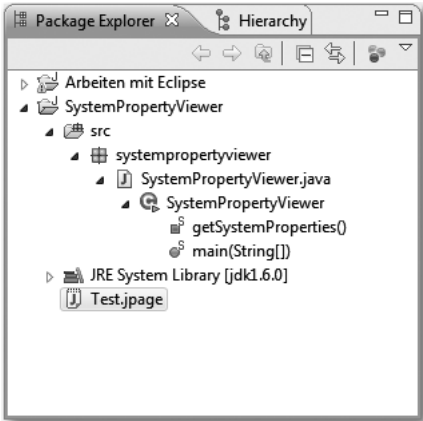


Abbildung 2.40 Die main()-Methode im Package Explorer

Bewegen Sie den Cursor bitte in eine beliebige Zeile und tippen Sie `//` ein Test. Drücken Sie nun die Tastenkombination `Ctrl+Z` oder wählen Sie `EDIT • UNDO`. Erwartungsgemäß ist der von Ihnen getippte Text verschwunden. Um ihn wiederherzustellen, rufen Sie bitte `EDIT • REDO` auf oder drücken `Ctrl+Y`. Er erscheint an seiner ursprünglichen Position, bleibt aber markiert. Ein versehentlicher Tastendruck würde ihn also überschreiben.

Die beiden Funktionen sind aber keineswegs auf reine Textoperationen beschränkt. Auch dies möchte ich Ihnen an einem Beispiel demonstrieren: Öffnen Sie bitte das Projekt *SystemPropertyViewer* und klappen Sie alle Einträge auf, bis der *Package Explorer* in etwa Abbildung 2.40 entspricht. Klicken Sie nun mit der rechten Maustaste bitte auf die `main()`-Methode und wählen Sie dann `REFACTOR • RENAME`. Sie sehen daraufhin den in Abbildung 2.41 gezeigten Dialog *Rename Method*.

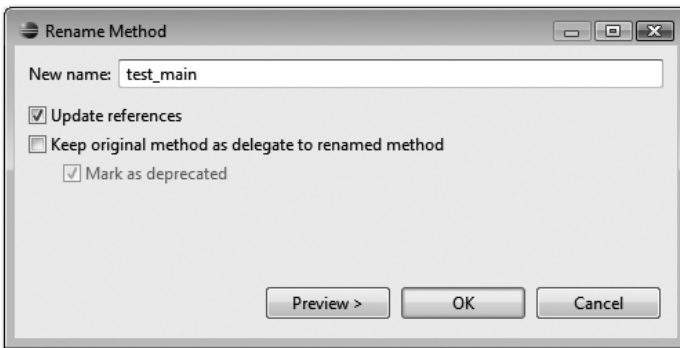


Abbildung 2.41 Dialog zum Umbenennen von Methoden

Geben Sie der Methode `main()` nun einen beliebigen neuen Namen und klicken Sie auf **OK**. Nachdem Eclipse alle Änderungen an Ihrem Quelltext abgeschlossen hat, sehen Sie im *Package Explorer* die umbenannte Methode. Welche weiteren Möglichkeiten Ihnen die IDE bietet, Ihre Quelltexte mittels *Refactoring* umzustellen, zeige ich Ihnen in Abschnitt 2.3, *Suchen, Ersetzen und Refactoring*. Wichtig an dieser Stelle ist, dass Sie diese doch recht umfassenden Änderungen mit minimalem Aufwand zurücknehmen können.

Werfen Sie bitte einen Blick in das Menü **EDIT**. Ganz oben finden Sie **UNDO RE-NAME METHOD**. Eclipse teilt Ihnen also mit, welche Änderung als Nächste widerrufen wird. Tun Sie dies bitte. Ein erneuter Blick in das Menü **EDIT** zeigt, dass Sie mit **REDO** die Methode `main()` erneut umbenennen könnten, wobei natürlich keine Eingaben in einem Dialog nötig sind. Wie viele **UNDO**-Schritte möglich sind, legen Sie im Dialog *Preferences* auf der Seite *Text Editors* fest. Navigieren Sie hierzu bitte zum Knoten *General • Editors • Text Editors* und tragen Sie den gewünschten Wert bei *Undo history size* ein.

UNDO und **REDO** wirken unmittelbar im Text-Editor. Änderungen an gespeicherten Dateien lassen sich auf diese Weise nicht rückgängig machen. Hierzu ein Beispiel: Bitte öffnen Sie die Klasse *Test*, löschen deren kompletten Inhalt und speichern Sie diese Änderung. Danach schließen Sie bitte den Editor.

Was passiert, wenn Sie eine solch gravierende Änderung zurücknehmen müssen? Ein Blick in das Menü **EDIT** zeigt schnell, dass keine entsprechende Operation zur Verfügung steht. Eine Möglichkeit ist sicher, ein Backup einzuspielen. Dies setzt allerdings die regelmäßige Erstellung solcher Sicherungen und deren Verfügbarkeit im Schadensfall voraus.

Eclipse bietet für solche Situationen die sogenannte *Local History*. Jedes Mal, wenn Sie (signifikante) Änderungen an einer Datei vornehmen und speichern, erzeugt die IDE eine Kopie. Sie können zu jedem Zeitpunkt die aktuelle Version durch eine frühere Version aus der *Local History* ersetzen. Auf diese Weise lassen sich sogar gelöschte Dateien wiederherstellen. Wie Sie später noch sehen werden, können Sie auch Versionsstände miteinander vergleichen.

Klicken Sie im *Package Explorer* bitte mit der rechten Maustaste auf die Klasse *Test* und wählen Sie *Restore from local History*. Wenn Eclipse das wiederherzustellende Element selbständig ermitteln kann, erhalten Sie die ursprüngliche Fassung ohne weitere Rückfrage. Andernfalls erscheint der Dialog *Restore Java Element from Local History*, den Sie in Abbildung 2.42 sehen. Er enthält eine Liste der Versionsstände und zeigt an, wann die Änderungen stattgefunden haben. Anhand einer Vorschau können Sie sehr schön erkennen, welche Version dem gewünschten Stand entspricht. Klicken Sie bitte auf *Restore*, um die markierte Fassung als neue Version der Datei zu übernehmen.

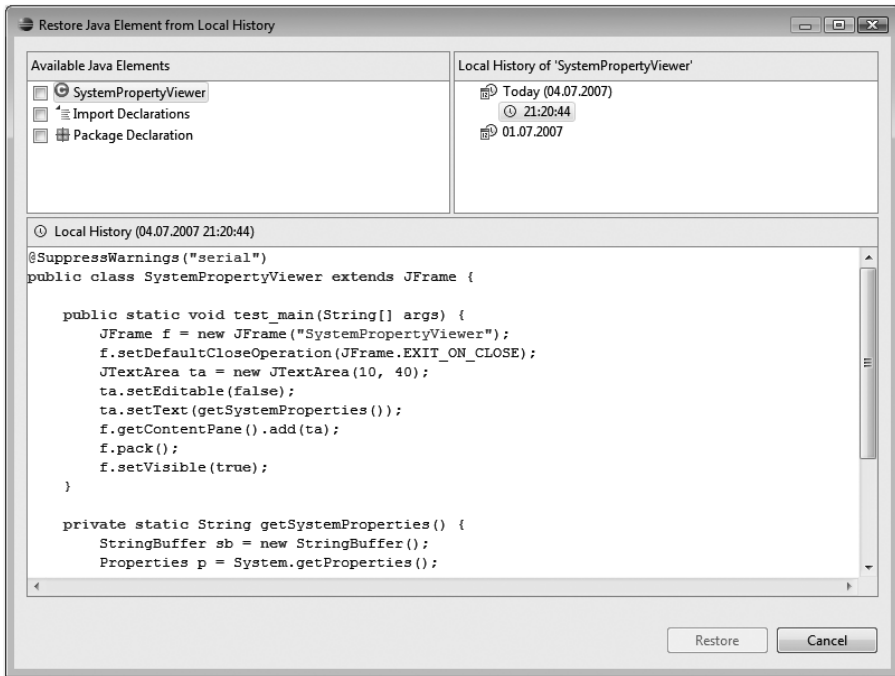


Abbildung 2.42 Wiederherstellen aus der Local History

Wie Sie gleich noch sehen werden, gibt es für das Rückgängigmachen von weniger »destruktiven« Änderungen eine eigene Funktion. Das eben vorgestellte RESTORE FROM LOCAL HISTORY setzen Sie ein, wenn Sie komplette Dateien wiederherstellen müssen. Haben Sie beispielsweise eine Datei im *Package Explorer* gelöscht, können Sie sie auf diese Weise retten.

Um dies auszuprobieren, klicken Sie bitte mit der rechten Maustaste auf die Klasse *Test* und wählen DELETE. Sofern Sie die Datei in einem Editor geöffnet haben, wird dieser geschlossen. Auch der *Package Explorer* zeigt *Test* nicht mehr an. Klicken Sie nun mit der rechten Maustaste auf sein Wurzelement und wählen Sie erneut RESTORE FROM LOCAL HISTORY. Im nun erscheinenden Dialog *Restore from Local History* können Sie diejenige Datei auswählen, die Sie wiederherstellen möchten.

Um Ihnen weitere Funktionen der *Local History* zu zeigen, werde ich der Klasse *Test* einige Elemente hinzufügen. Positionieren Sie den Cursor hierzu bitte auf dem Namen *Test* der Konstruktor-Deklaration und rufen Sie dann REFACTOR • INTRODUCE FACTORY auf. Im Dialog *Introduce Factory*, den Sie in Abbildung 2.43 sehen, fragt Eclipse den Namen der zu generierenden Methode sowie der Factory-

Klasse ab. Außerdem können Sie festlegen, ob der bestehende Konstruktor als `private` deklariert werden soll. Schließen Sie den Dialog bitte mit einem Klick auf **OK**.

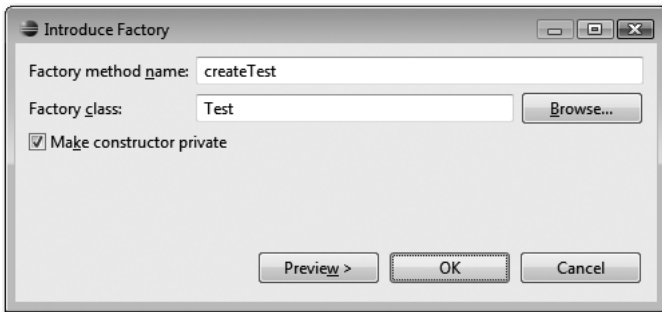


Abbildung 2.43 Dialog zum Erzeugen von Factory-Methoden

Auch das Erzeugen von Factory-Methoden ist ein Aspekt beim Refactoring von Java-Quelltexten. Informationen zu diesem Thema finden Sie in Abschnitt 2.3, *Suchen, Ersetzen und Refactoring*. Im Hinblick auf meine Erklärungen zur *Local History* ist es wichtig, dass die Klasse `Test` in zwei Punkten modifiziert wurde: Zum einen gibt es die neue Methode `createTest()`, zum anderen ist der Konstruktor von außen nicht mehr zugänglich.

Was den aktuellen Stand eines Quelltextes von seinen Vorgängerversionen unterscheidet, können Sie sehr schön mit **COMPARE WITH • LOCAL HISTORY** herausfinden. Sie erreichen diese Funktion über das Kontextmenü, das Sie durch Klicken mit der rechten Maustaste auf die Klasse `Test` im *Package Explorer* öffnen. Tun Sie dies bitte.

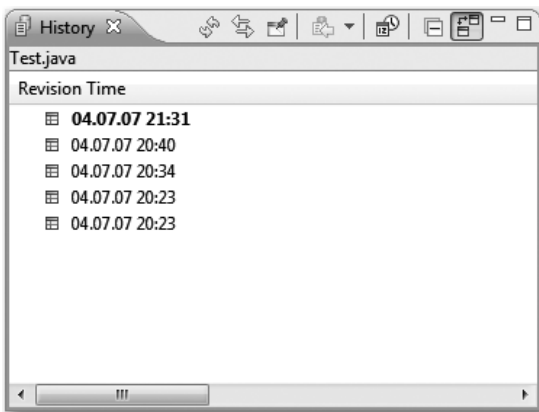


Abbildung 2.44 Die Sicht History

Eclipse wird daraufhin die Sicht *History* öffnen, die Sie in Abbildung 2.44 sehen. Den größten Raum nimmt eine Liste mit Versionsständen ein, die Sie in ähnlicher Form bereits vom Dialog *Restore from Local History* kennen. Ein Doppelklick auf einen der Einträge öffnet die entsprechende Version in einem Editor. Was hierbei genau passiert, ist vom Status des Symbols *Compare Mode* abhängig. Ist dieser Vergleichsmodus nicht aktiv, wird jede Version in einer eigenen schreibgeschützten Editor-Instanz angezeigt. Möchten Sie Versionsstände hingegen miteinander vergleichen, klicken Sie das Symbol *Compare Mode* bitte einmal an. Doppelklicks auf Versionsstände zeigen dann eine spezielle Vergleichsansicht, die Sie in Abbildung 2.45 sehen.

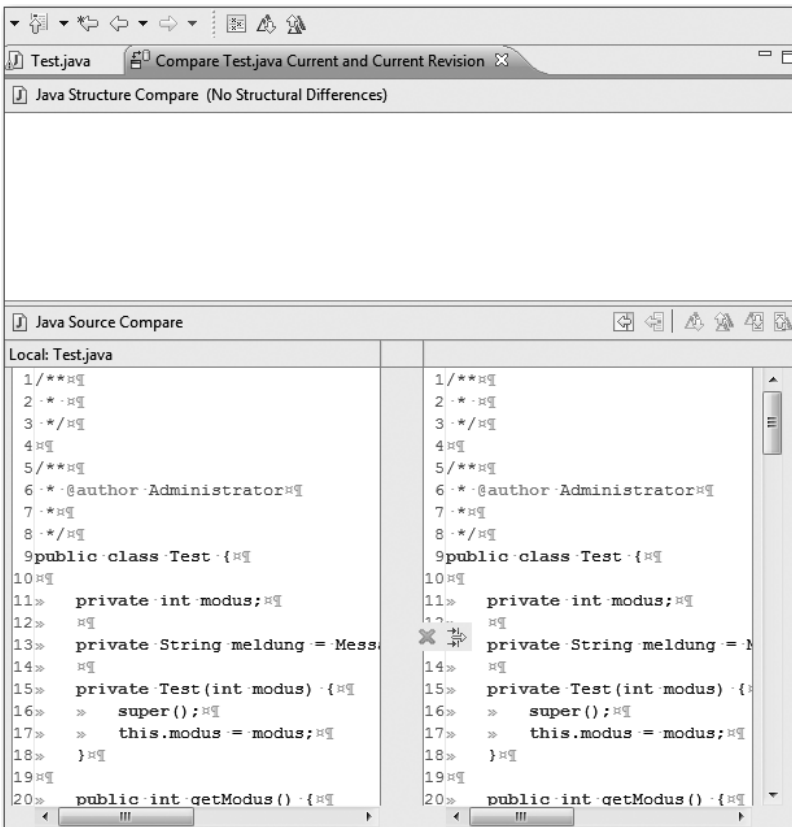


Abbildung 2.45 Vergleich zweier Versionsstände

Diese Vergleichsansicht besteht aus zwei Bereichen. *Java Structure Compare* zeigt Ihnen beispielsweise mit einem Plus-Symbol an, dass die Methode `createTest()` neu hinzugefügt wurde. Sie erhalten so einen Überblick über Änderungen an der Struktur eines Quelltextes. *Java Source Compare* hingegen stellt alle Änderungen

detailliert gegenüber. Sie sehen hier nicht nur die neue Methode `createTest()`, sondern auch die Änderung der Sichtbarkeit des Konstruktors. Mit den Symbolen des Vergleichseditors können Sie gezielt zu Änderungen navigieren sowie Bereiche einer früheren Version in die aktuelle Fassung übernehmen. Dies ist besonders praktisch, wenn Sie eine Methode gelöscht haben und später feststellen, dass Sie diese doch noch benötigen.

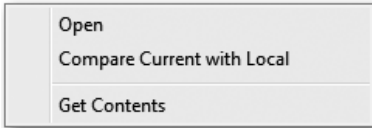


Abbildung 2.46 Kontextmenü eines Versionsstandes

Die Sicht *History* öffnet für jeden Versionsstand nach Rechtsklick auf dessen Eintrag ein Kontextmenü, das in Abbildung 2.46 zu sehen ist. Dessen Menüpunkt `OPEN` stellt die ausgewählte Version in einem eigenen, schreibgeschützten Editor dar, auch wenn der Vergleichsmodus aktiv ist. `COMPARE CURRENT WITH LOCAL` öffnet den eben beschriebenen Vergleichseditor. Dies funktioniert auch dann, wenn Sie den Vergleichsmodus gar nicht aktiviert haben. `GET CONTENTS` schließlich übernimmt die ausgewählte Version vollständig als neue Fassung.

Mithilfe der Vergleichsfunktion können Sie also sehr komfortabel Änderungen zwischen verschiedenen Versionsständen nachvollziehen und gegebenenfalls ältere Fassungen wiederherstellen. Wie viele Änderungen in der Local History vorgehalten werden und wie lange ältere Versionsstände verfügbar sind, stellen Sie auf der Seite *Local History* des Dialogs *Preferences* ein. Navigieren Sie hierzu bitte zum Knoten `GENERAL • WORKSPACE • LOCAL HISTORY`.

Die Sichten Navigator und Outline

In diesem Abschnitt möchte ich Ihnen mit *Navigator* und *Outline* zwei weitere wichtige Sichten vorstellen, die Sie bei Ihrer Arbeit mit Eclipse unterstützen.

Die Sicht *Navigator* stellt Ressourcen des Arbeitsbereichs hierarchisch dar. Ihre Projekte erscheinen als Äste eines Baums, was auf den ersten Blick stark an den Ihnen bereits bekannten *Package Explorer* erinnert. Wenn Sie die Elemente, die die beiden Sichten anzeigen, miteinander vergleichen, stellen Sie schnell fest, dass der *Navigator* auch solche Ressourcen anzeigt, die im *Package Explorer* verborgen bleiben. Beispiele hierfür sind die in Abbildung 2.47 gezeigten Dateien `.classpath` und `.project` sowie das Verzeichnis `bin`. Der *Navigator* zeigt Ihnen die physikalische Struktur von Projekten, wohingegen der *Package Explorer* diese aus Java-Sicht darstellt.

Wie im *Package Explorer* können Sie auch im *Navigator* Dateien mittels Doppelklick öffnen. Um ein Kontextmenü aufzurufen, das die zu der Ressource passenden Funktionen wie *Kopieren*, *Verschieben* und *Vergleichen* anbietet, klicken Sie eine Ressource mit der rechten Maustaste an.

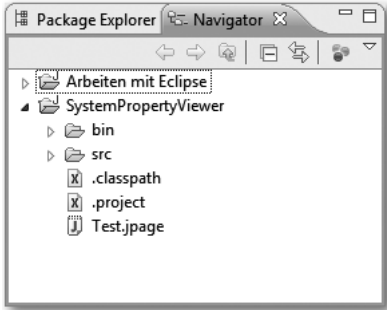


Abbildung 2.47 Die Sicht Navigator

Das Symbol *Link with Editor* verbindet den *Navigator* mit dem aktiven Editor. Wenn Sie zwischen verschiedenen Instanzen eines Editors wechseln, passt sich die Sicht automatisch an und markiert die in diesem Editor angezeigte bzw. bearbeitete Ressource. Hierzu ein Beispiel: Öffnen Sie bitte die beiden Dateien *Test.java* und *.classpath*, nachdem Sie *Link with Editor* aktiviert haben. Wechseln Sie nun zwischen den beiden Editor-Instanzen hin und her. Beachten Sie dabei bitte die Anzeige im *Navigator*. Sie werden feststellen, dass die im aktiven Editor angezeigte Datei im *Navigator* markiert ist.

Die Möglichkeit, den Inhalt einer Sicht an einen Editor zu binden, steht für einige Sichten zur Verfügung. So finden Sie auch im *Package Explorer* das Symbol *Link with Editor*. Auch die Sicht *Outline*, die ich Ihnen nun vorstellen möchte, kann ihren Inhalt an die aktive Editor-Instanz anpassen. Das Symbol ist allerdings nicht unmittelbar zu sehen, sondern über das Klappmenü der Sicht zugänglich.

Generell stellt *Outline* strukturierte Dateien in Form einer Übersicht oder Gliederung dar. Was genau die Sicht anzeigt, ist also von der Art der Datei abhängig. Beispielsweise sehen Sie von Java-Quelltexten deren Konstruktoren, Methoden und Variablen.

Prüfen Sie bitte zunächst, ob Sie die Funktion *Link with Editor* in der Sicht *Outline* aktiviert haben. Falls nicht, tun Sie dies bitte. Öffnen Sie nun die Klasse *Test*. Ihre Sicht sollte in etwa Abbildung 2.48 entsprechen. Um im Editor zu einem bestimmten Element zu gelangen, können Sie es in *Outline* anklicken. Der umgekehrte Weg ist ebenfalls möglich. Das bedeutet, wenn Sie im Quelltext navigieren, wird das aktuelle Element in *Outline* markiert.

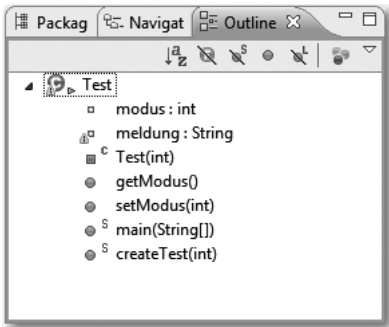


Abbildung 2.48 Die Sicht Outline

Mit der Symbolleiste der Sicht können Sie einstellen, welche Elemente Sie anzeigen möchten. Beispielsweise lassen sich statische Variablen und Methoden verbergen. Die Reihenfolge der Elemente in *Outline* entspricht deren Auftreten im Quelltext, sofern Sie die Sortierfunktion nicht eingeschaltet haben.

Übrigens können Sie durch Verschieben von Elementen in der Sicht deren Position im Quelltext ändern. Haben Sie beispielsweise eine Methode an der falschen Stelle eingefügt, lässt sich dies auf einfache Weise beheben. Klicken Sie hierzu auf den Methodennamen und halten Sie die Maustaste gedrückt, während Sie das Element an seine neue Position schieben.

Working Sets

In diesem Abschnitt stelle ich Ihnen die sogenannten *Working Sets* vor. Hierbei handelt es sich um eine Art Filtermechanismus, mit dem Sie unter anderem steuern können, welche Elemente in den Navigations-Sichten der Workbench angezeigt werden.

Ganz allgemein fassen Working Sets Elemente zu Gruppen zusammen. Für bestimmte Sichten können Sie dann festlegen, dass diese nur noch diejenigen Elemente anzeigen, die zu einem bestimmten Working Set gehören. Dies ist nützlich, wenn Sie die Zahl der angezeigten Elemente eingrenzen möchten. Working Sets können aber auch dazu verwendet werden, Aktionen auf mehrere Elemente gleichzeitig anzuwenden.

Öffnen Sie bitte die Sicht *Package Explorer* und klappen dessen Menü auf, indem Sie den kleinen nach unten weisenden Pfeil anklicken. Nachdem Sie *SELECT WORKING SET* aufgerufen haben, öffnet sich der Dialog *Select Working Set*, den Sie in Abbildung 2.49 sehen. Klicken Sie bitte auf *New*, um ein neues Working Set anzulegen.

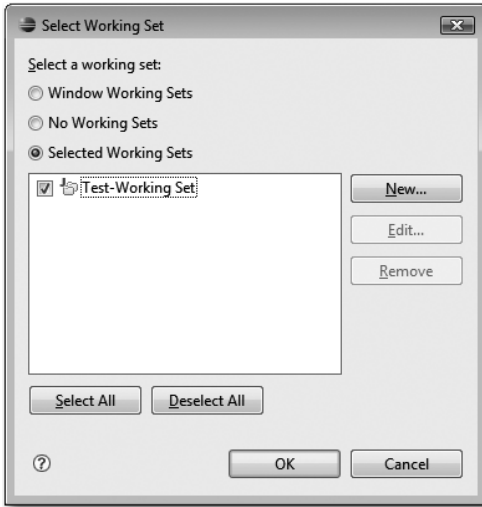


Abbildung 2.49 Dialog zum Auswählen von Working Sets

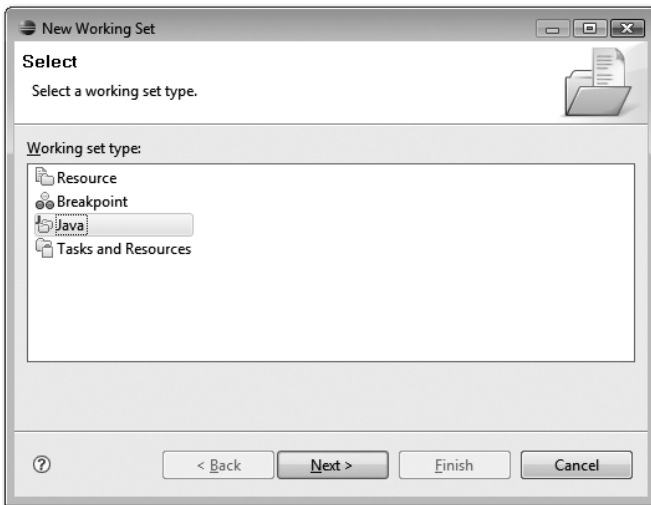


Abbildung 2.50 Dialog zum Anlegen von Working Sets

Im daraufhin erscheinenden Dialog *New Working Set*, den Sie in Abbildung 2.50 sehen, wählen Sie zunächst den Typ des anzulegenden Working Sets aus. Markieren Sie bitte den Eintrag *Java* und klicken Sie anschließend auf *Next*. Auf der zweiten Seite des Assistenten, die Sie in Abbildung 2.51 sehen, geben Sie dem Working Set einen Namen (beispielsweise *Test-Working Set*) und legen fest, welche Elemente Sie ihm zuordnen möchten. Deselektieren Sie hierzu bitte alle Ele-

mente außer den drei Dateien *.classpath*, *.project* und Ihre Scrapbook-Seite. Um alle Einträge zu sehen, müssen Sie die Knoten gegebenenfalls aufklappen. Bitte schließen Sie den Dialog mit *Finish*.

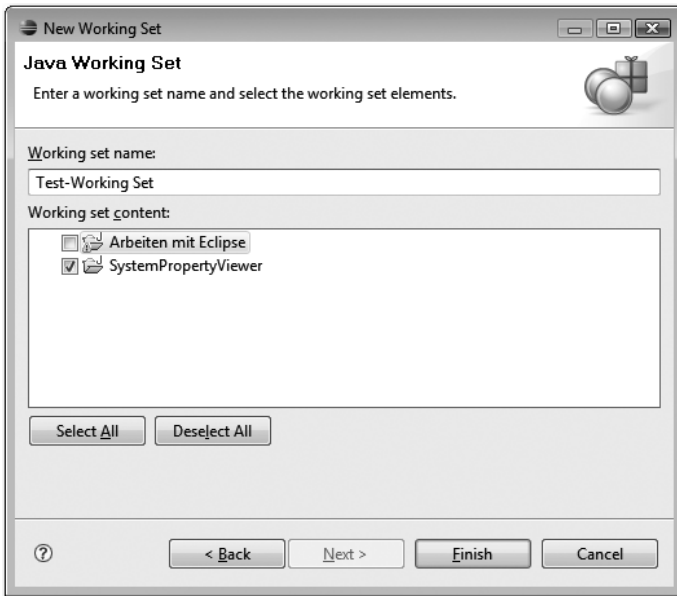


Abbildung 2.51 Zweite Seite des Assistenten zum Anlegen von Working Sets

Eine weitere Möglichkeit, Dateien zu Working Sets hinzuzufügen und eine bestehende Zuordnung zu lösen, bietet das Menü EDIT in Gestalt der beiden Funktionen ADD TO WORKING SET und REMOVE FROM WORKING SET. In beiden Fällen wählen Sie aus einem Untermenü das betreffende Working Set aus.

Der Dialog *Select Working Set* zeigt Ihr neu angelegtes Working Set sowie das bereits bestehende *Other Projects* an. Bitte deselektieren Sie dieses. Wenn Sie den Dialog mit OK beenden, werden im *Package Explorer* nur noch Elemente angezeigt, die zu dem neu angelegten Working Set *Test-Working Set* gehören.

Sie werden feststellen, dass sich die Anzeige des *Package Explorers* geändert hat. Statt des Projektnamens finden Sie nunmehr das Element *Test-Working Set*. Welche Wurzelemente der *Package Explorer* anzeigt, können Sie in dessen Klappmenü einstellen. Wählen Sie hierzu bitte TOP LEVEL ELEMENTS • PROJECTS oder TOP LEVEL ELEMENTS • WORKING SETS.

Ihnen ist wahrscheinlich aufgefallen, dass unter *Test-Working Set* nur Ihre Scrapbook-Seite zu finden ist, nicht aber die beiden ebenfalls zum Working Set gehörenden Dateien *.classpath* und *.project*. Der Grund hierfür ist, dass der *Package*

Explorer stets eine logische Sicht auf Java-Projekte bietet und bestimmte Dateien immer ausblendet.

Um die Wirkung von Working Sets auf andere Sichten auszuprobieren, öffnen Sie nun bitte den *Navigator* und wählen *Select Working Set* in dessen Klappmenü. Es öffnet sich der Dialog *Select Working Set* mit seinen drei zusätzlichen Auswahlmöglichkeiten *Window Working Sets*, *No Working Sets* und *Selected Working Sets*, die Sie bereits aus Abbildung 2.51 kennen.

Klicken Sie bitte auf *Selected Working Sets* und wählen Sie *Test-Working Set*. Beenden Sie anschließend den Dialog, indem Sie auf *OK* klicken. Sie werden feststellen, dass der *Navigator* nur noch die drei Dateien *.project*, *.classpath* sowie Ihre Scrapbook-Seite anzeigt. Um die Filterung durch Working Sets aufzuheben, wählen Sie bitte die Klappmenü-Funktion *DESELECT WORKING SET*.

Working Sets helfen Ihnen also dabei, nicht benötigte Elemente auszublenden. Anstatt Projekte im *Package Explorer* zu schließen, legen Sie zwei Working Sets an. Eines enthält die auszublendenden Elemente, das zweite diejenigen, mit denen Sie aktuell arbeiten. Sie können Working Sets übrigens auch dazu verwenden, den Build-Vorgang Ihrer Projekte zu optimieren. Mit *PROJECT • BUILD WORKING SET* wählen Sie das Working Set aus, dessen Elemente Sie neu erzeugen möchten. Informationen zum Thema Projektverwaltung finden Sie im folgenden Kapitel, *Arbeitsbereiche und Projekte*.

2.3 Suchen, Ersetzen und Refactoring

Es gibt unzählige Situationen, in denen Sie nach Schlüsselwörtern oder Bezeichnern in Ihren Quelltexten oder projektbegleitenden Dateien suchen. Beispielsweise kann es nötig sein, alle Referenzen auf einen bestimmten Datentyp zu finden. Oder Sie möchten gerne wissen, wie oft eine Methode aufgerufen wird. Sehr häufig kommt es auch vor, dass Sie Elemente einfach umbenennen müssen. All dies hat irgendwie mit *Suchen* und *Ersetzen* zu tun. Eclipse bietet für verschiedene Problemstellungen in diesem Bereich spezialisierte Hilfsmittel. Welche dies sind und wie Sie sie einsetzen, zeige ich Ihnen im Folgenden.

2.3.1 In und nach Dateien suchen

Grundlage für die folgenden Erklärungen ist ein kleines Programm, das ich *UI-Properties* genannt habe. Es erzeugt eine Textdatei, die Schlüssel aus der Swing-Klasse `UIDefaults` enthält. Sie benötigen diese Schlüssel, wenn Sie bestimmte Eigenschaften von Swing-Komponenten abfragen möchten.

Das Programm `UIProperties`

Legen Sie bitte ein neues, leeres Java-Projekt mit dem Namen `UIProperties` an. Wählen Sie hierzu `FILE • NEW • PROJECT` und klicken Sie im anschließend erscheinenden Dialog *New Project* auf `JAVA • JAVA PROJECT`. Nachdem Eclipse das Projekt angelegt hat, fügen Sie ihm die Klasse `UIProperties` hinzu. Klicken Sie im *Package Explorer* mit der rechten Maustaste auf die Projekt-Wurzel und wählen Sie `NEW • CLASS`. Sie sehen den Dialog *New Java Class*, in dem Sie als Paketnamen bitte `uiproperties` eingeben. Nachdem Sie den Dialog durch Klicken auf *Finish* geschlossen haben, ergänzen Sie bitte die Klasse `UIProperties` entsprechend dem folgenden Listing:

```
package uiproperties;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Enumeration;

import javax.swing.UIManagerDefaults;
import javax.swing.UIManager;

public class UIProperties {

    public static void main(String[] args) {
        UIManagerDefaults uiDefaults = UIManager.getDefault();
        Enumeration<Object> keys = uiDefaults.keys();
        String datei = "ausgabe.txt";
        FileWriter fr = null;
        try {
            fr = new FileWriter(datei);
            while (keys.hasMoreElements()) {
String key =
keys.nextElement().toString();
                if ((args.length == 0) ||
(key.indexOf(args[0]) >= 0)) {
                    fr.write(key.toString() + "\n");
                }
            }
        } catch (IOException e) {
            System.err.println(e);
        } finally {
            // Ausgabestrom schließen
            if (fr != null) {
                try {
                    fr.close();
                }
            }
        }
    }
}
```


Auf der Registerkarte *File Search* tragen Sie unter *File name patterns* den Namen der gesuchten Datei ein, also *ausgabe.txt*. Das Feld *Containing text* lassen Sie bitte leer. Wo Sie suchen möchten, stellen Sie unter *Scope* ein. Solange Sie nur wenige Projekte verwalten, ist *Workspace* eine gute Wahl. Bei umfangreichen Projekten mit vielen Quelltexten kann es sich allerdings lohnen, ein eigenes *Working Set* anzulegen und die Suche darauf zu beschränken.

Sie starten die Suche, indem Sie auf die Schaltfläche *Search* klicken. Die Liste mit Fundstellen wird in einer eigenen Sicht *Search* dargestellt, die Sie in Abbildung 2.53 sehen. Falls sie sich nicht automatisch geöffnet hat, können Sie die Sicht über **WINDOW • SHOW VIEW • SEARCH** aktivieren.

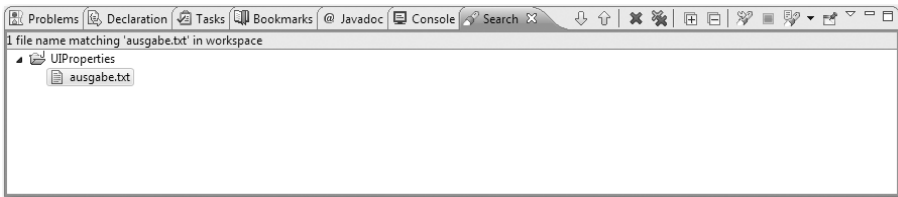


Abbildung 2.53 Die Sicht Search

Wie Sie bereits wissen, ist der *Arbeitsbereich* eine Art Container, in dem Projekte und Metainformationen abgelegt werden. Er wird von Eclipse verwaltet. Wenn die IDE selbst Änderungen an ihm vornimmt, beispielsweise weil Sie als Anwender eine Datei hinzufügen oder löschen, kann sie ihre Verwaltungsinformationen entsprechend aktualisieren.

Anders verhält es sich, wenn Dritte in den Arbeitsbereich schreiben, so wie es *UIProperties* tut. In diesem Fall bemerkt Eclipse unter Umständen nicht, dass der Arbeitsbereich eine neue Datei namens *ausgabe.txt* enthält. Ihre Suche liefert dann keine Treffer. Falls dies passiert, können Sie Eclipse auf die Sprünge helfen, indem Sie im *Navigator* an einer beliebigen freien Stelle die rechte Maustaste drücken und anschließend **REFRESH** auswählen. Führen Sie dann die Suche erneut aus.

Dies ist ohne erneutes Aufrufen des Dialogs *Search* möglich. Klicken Sie stattdessen in der Sicht *Search* auf das Symbol **RUN THE CURRENT SEARCH AGAIN** oder drücken Sie die Taste **F5**. Eine weitere interessante Funktion bietet **SHOW PREVIOUS SEARCHES**. Ein Klick auf das Symbol öffnet den Dialog *Previous Searches*, den Sie in Abbildung 2.54 sehen. Mit ihm verwalten Sie Suchanfragen und führen diese gegebenenfalls erneut aus. Wählen Sie hierzu die gewünschte Suche aus und klicken Sie anschließend auf *Open*. Nicht mehr benötigte Anfragen lassen sich nach dem Markieren mit der Schaltfläche *Remove* entfernen.

Der kleine nach unten weisende Pfeil rechts neben dem Symbol **SHOW PREVIOUS SEARCHES** öffnet ein Klappenmenü, das Sie in [Abbildung 2.55](#) sehen. Mit ihm können Sie Ihre früheren Suchanfragen ohne Umweg über den Dialog *Previous Searches* erneut starten.

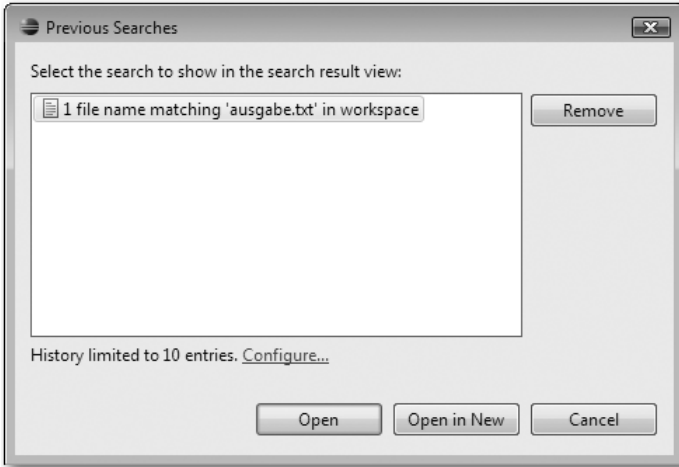


Abbildung 2.54 Der Dialog *Previous Searches*

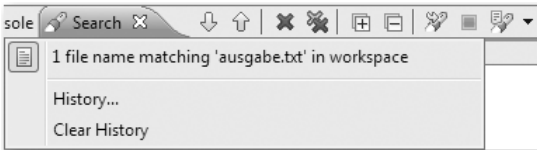


Abbildung 2.55 Klappenmenü des Symbols **Show Previous Searches**

Mit einem Doppelklick auf ihren Namen in der Sicht *Search* können Sie die Datei *ausgabe.txt* in einem Editor bearbeiten. Ein Druck auf die rechte Maustaste hingegen öffnet ein Kontextmenü, in dem Sie die Datei beispielsweise in den Sichten *Package Explorer* und *Navigator* anzeigen können.

Die Suche nach *ausgabe.txt* lieferte erwartungsgemäß nur einen Treffer. In der Regel werden Sie bei Suchanfragen aber mehrere Fundstellen erwarten. Bitte probieren Sie dies aus, indem Sie eine neue Suche starten, die alle Java-Quelltexte im Arbeitsbereich findet. Im Dialog *Search* tragen Sie hierzu im Feld *File name pattern* den Wert **.java* ein. Die Sicht *Search* stellt alle gefundenen Dateien nach Projekten geordnet dar. Mit den beiden Symbolen **SHOW NEXT MATCH** und **SHOW PREVIOUS MATCH** können Sie sie in einer Editor-Instanz der Reihe nach ansehen bzw. bearbeiten.

In Dateien suchen

Ihnen ist im Dialog *Search* sicher das Feld *Containing Text* aufgefallen. Wenn Sie hier etwas eintragen, gilt dies als zusätzliches Suchkriterium. Es werden also nur diejenigen Dateien gefunden, die den eingetragenen Text enthalten und dem angegebenen Namensmuster entsprechen. Bitte probieren Sie es aus, indem Sie als Namensmuster `*` wählen und Eclipse nach dem Text *UI* in allen Projekten suchen lassen. Achten Sie bitte darauf, dass Unterschiede in der Groß- und Kleinschreibung ignoriert werden. Das Häkchen vor *case sensitive* darf also nicht gesetzt sein.

Eclipse wird eine ganze Reihe von Treffern präsentieren, die sich nicht nur auf Java-Quelltexte beziehen. Auch die Datei *.project* enthält den von Ihnen eingegeben Suchtext. Um die Fundstellen zu überprüfen, öffnen Sie sie bitte. Eclipse hat *UI* also auch als Bestandteil von Wörtern erkannt, beispielsweise in `<buildSpec>` und in `</buildCommand>`.

Anstelle von festen Zeichenfolgen können Sie übrigens auch nach regulären Ausdrücken suchen, indem Sie den entsprechenden Schalter setzen. Beispiele zum Umgang mit ihnen finden Sie im folgenden Abschnitt, der sich mit dem Suchen und Ersetzen in Texten beschäftigt.

2.3.2 Suchen und Ersetzen im Quelltext

Wenn eine Editorinstanz aktiv ist, stehen Ihnen im Menü EDIT zahlreiche Funktionen zum Suchen und Ersetzen von Text zur Verfügung. Sie steuern den Suchvorgang mithilfe des Dialogs *Find/Replace*, den Sie in Abbildung 2.56 sehen. Um ihn aufzurufen, wählen Sie bitte EDIT • FIND/REPLACE oder drücken die Tastenkombination `[Ctrl]+[F]`. Der Dialog wird sowohl für die reine Suche als auch für das Ersetzen von Zeichenketten verwendet. Im ersten Fall können Sie das Feld *Replace With* leer lassen. Dies gilt auch, wenn Sie Zeichenfolgen löschen, also mit *nichts* ersetzen möchten.

Sie können den Such- und Ersetzvorgang über zahlreiche Optionen steuern. Beispielsweise lässt sich Eclipse anweisen, Unterschiede in der Groß-/Kleinschreibung zu beachten oder die Suche am Beginn der Datei fortzusetzen, wenn das Ende erreicht wurde. Außerdem können Sie entweder in der gesamten Datei oder nur im markierten Bereich suchen sowie die Suchrichtung bestimmen.

Bitte probieren Sie die Suchfunktion aus, indem Sie die Datei *.project* eines Projekts, beispielsweise *UIProperties*, öffnen und anschließend FIND/REPLACE aufrufen. Suchen Sie bitte nach *build*, wobei Sie alle Optionen außer *Wrap Search* deaktiviert lassen. Jeder Klick auf die Schaltfläche *Find* springt im Editor zur jeweils nächsten Fundstelle. Wie Sie in Abbildung 2.57 sehen, werden Treffer farb-

lich hervorgehoben. Die Zeile, die die aktuelle Fundstelle enthält, ist zusätzlich mit einer anderen Hintergrundfarbe versehen.

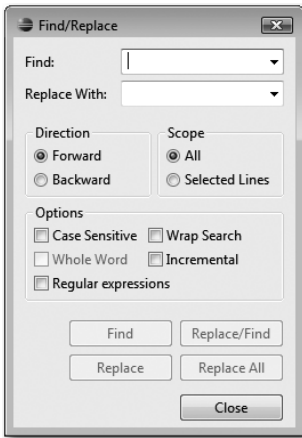


Abbildung 2.56 Dialog zum Suchen und Ersetzen von Zeichenfolgen

Bitte schließen Sie den Dialog *Find/Replace*, indem Sie auf *Close* klicken. Möchten Sie noch einmal nach *build* suchen, wäre es umständlich, ihn erneut öffnen zu müssen. Stattdessen können Sie mit *FIND NEXT* bzw. *FIND PREVIOUS* im Menü *EDIT* oder durch Drücken der korrespondierenden Tastaturkürzel unmittelbar zur nächsten bzw. vorherigen Fundstelle springen. Hatten Sie *Wrap Search* aktiviert, wird die Suche automatisch am Beginn bzw. Ende der Datei fortgesetzt. Ein akustisches Signal macht Sie darauf aufmerksam.

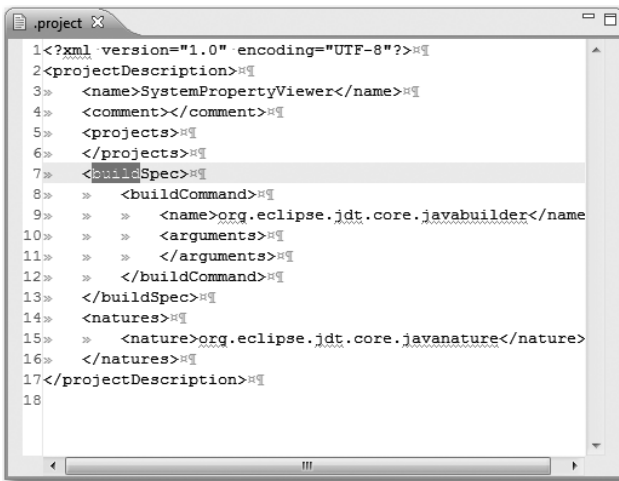


Abbildung 2.57 Markierung von Fundstellen im Texteditor

Eine weitere äußerst praktische Funktion ist die inkrementelle Suche, die Sie im Dialog *Find/Replace* oder über das Menü EDIT aktivieren können.

Inkrementelles Suchen

Um sie zu testen, öffnen Sie bitte die Datei *ausgabe.txt* und wählen anschließend EDIT • INCREMENTAL FIND NEXT oder drücken die Tastenkombination **[Ctrl]+[J]**. Sie sehen in der *Statuszeile* einen Hinweis, dass Sie das inkrementelle Suchen aktiviert haben. Angenommen, Sie möchten wissen, welche Schlüssel den Namensbestandteil *table* enthalten. Tippen Sie hierzu die Buchstaben **[t]** und **[a]**. Die Zeile, die die erste Fundstelle enthält, wird mit einer besonderen Hintergrundfarbe gekennzeichnet. Außerdem erscheinen die bereits getippten Buchstaben markiert. Allerdings ist der erste gefundene Schlüssel *TabbedPane.textIconGap*. Tippen Sie deshalb bitte zwei weitere Buchstaben ein, **[b]** und **[l]**.

Nun sollte die Zeile *Table.dropCellBackground* als Fundstelle gekennzeichnet werden. Eclipse hat also den ersten Schlüssel, der die gesuchte Zeichenfolge enthält, gefunden. Jetzt können Sie mit den beiden Tasten **[↓]** und **[↑]** zur nächsten bzw. zur vorherigen Fundstelle springen.

Suchen mit regulären Ausdrücken

Eclipse bietet in den meisten Suchdialogen die Möglichkeit an, nach *regulären Ausdrücken* anstelle von festen Zeichenfolgen zu suchen. Dies ist beispielsweise nützlich, wenn Sie nur den Anfang und das Ende eines Wortes kennen. Öffnen Sie bitte die Datei *ausgabe.txt* und rufen Sie den Dialog *Find/Replace* auf. Aktivieren Sie anschließend die Suche nach regulären Ausdrücken. Sobald Sie den Cursor in das Suchfeld bewegen, zeigt Ihnen das Symbol *Content Assist available* in Gestalt einer kleinen Glühbirne die Verfügbarkeit des sogenannten *Content Assists* an. Um ihn aufzurufen, drücken Sie bitte die Tastenkombination **[Ctrl]+[Leertaste]**.

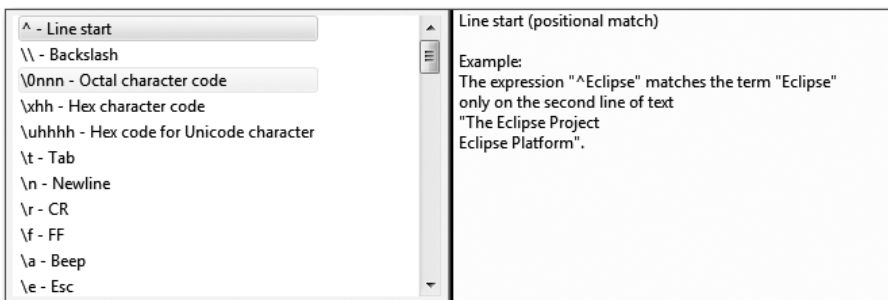


Abbildung 2.58 Der Content Assist

Wie Sie in Abbildung 2.58 sehen, gibt Ihnen *Content Assist* einen Überblick, wie Sie reguläre Ausdrücke zusammensetzen.

Beispielsweise ist der Punkt (.) ein Platzhalter für ein beliebiges Zeichen. Der *Asterisk* (*) gibt an, dass das ihm vorangestellte Zeichen beliebig oft vorkommen kann. Die beiden Ausdrücke lassen sich kombinieren, um beliebige Wortbestandteile zu finden. Möchten Sie also alle Zeilen von *ausgabe.txt* finden, die mit *split* beginnen und auf *color* enden, so können Sie den einfachen regulären Ausdruck *split.*color* verwenden.

2.3.3 Refactoring

Der Begriff *Refactoring* wird in Zusammenhang mit der Verbesserung der Struktur eines Programms verwendet, wobei sich das bestehende Verhalten der Anwendung nicht ändert. Eine solche »Umgestaltung« des Quelltextes kann maschinell oder von Hand erfolgen.

Ziel des Refactorings ist, die Lesbarkeit, Wartbarkeit und Verständlichkeit der Software zu verbessern. Dies wiederum führt idealerweise zu einer Reduzierung der Aufwände für die Fehleranalyse, aber auch für die Erweiterung des Funktionsumfangs. Populär wurde das Refactoring als Bestandteil des Programmiermodells *Extreme Programming*, wengleich seine Ursprünge bis in das Jahr 1990 zurückreichen.

In diesem Abschnitt möchte ich Ihnen anhand einiger Beispiele zeigen, wie Sie Eclipse beim Umstrukturieren Ihrer Quelltexte unterstützt. Zwei weitere Einsatzmöglichkeiten finden Sie in Abschnitt 2.2.4, *Komfortabel arbeiten*. Wenn Sie tiefer in die Ideen und Konzepte des Refactorings eintauchen möchten, empfehle ich Ihnen, sich entsprechende weiterführende Literatur zu besorgen. Die Zusammenfassung am Ende dieses Kapitels hält eine kleine Lektüreliste bereit.

Umbenennen und Verschieben

Praktisch alle Befehle, die sich auf das Refactoring beziehen, sind über das Menü *REFACTOR* erreichbar. Auch das Kontextmenü einer Datei, das Sie beispielsweise im *Package Explorer* durch Anklicken mit der rechten Maustaste öffnen können, enthält ein solches Untermenü.

Duplizieren Sie bitte die Datei *UIProperties.java*, indem Sie Sie deren Kontextmenü öffnen und *COPY* wählen. Klicken Sie nun die Projektwurzel *UIProperties* mit der rechten Maustaste an und wählen Sie dann *PASTE*. Sie haben auf diese Weise eine Kopie der Datei erzeugt, die allerdings nicht in einem Quelltextverzeichnis abgelegt wurde, sondern auf derselben Ebene liegt wie *ausgabe.txt*.

Refactoring sieht vor, bei Bedarf Klassen in andere Pakete zu verschieben. Sie können sich dies zunutze machen, um die Datei in ein Quelltextverzeichnis zu bewegen. Klicken Sie die eben erzeugte Kopie *UIProperties.java* im *Package Explorer* an und wählen Sie in der Menüleiste **REFACTOR • MOVE**. Sie sehen daraufhin den in Abbildung 2.59 gezeigten Dialog zum Verschieben einer Klasse. Mit ihm legen Sie deren Zielpaket fest. Wenn Sie die Wurzel *UIProperties* aufklappen, finden Sie das Verzeichnis *src* mit den beiden Paketen *uiproperties* und (*default package*). Klicken Sie Letzteres bitte an und schließen Sie den Dialog mit **OK**.

Der *Package Explorer* zeigt die Kopie der Klasse *UIProperties* nun ebenfalls unterhalb des Knotens *src* (im Standardpaket) an. Wenn Sie diese mit einem Doppelklick öffnen, wird Eclipse allerdings einen Fehler melden. Der Grund hierfür liegt in der Anweisung `package`. Der Quelltext liegt nämlich in einem anderen Verzeichnis, als er (aufgrund dieser Anweisung) eigentlich müsste. Eclipse hat ihn nicht angepasst, weil der Ursprung der Verschiebeaktion kein Quelltextverzeichnis war. Um das Problem zu beheben, könnten Sie die betreffende Zeile einfach löschen. Stattdessen rufen Sie bitte wie eben beschrieben erneut den Dialog *Move* auf.

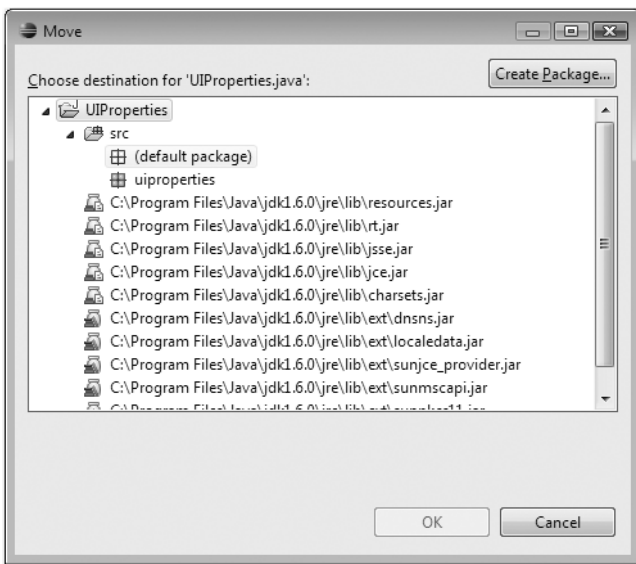


Abbildung 2.59 Dialog zum Verschieben einer Klasse

Klicken Sie auf die Schaltfläche *Create Package*, woraufhin sich der in Abbildung 2.60 gezeigte Dialog zum Anlegen neuer Java-Pakete öffnet. Geben Sie in der Zeile *Name* bitte einen Paketnamen in der üblichen Java-Notation an, beispielsweise *de.thomaskuenneth.uiproperties*.

Sie können in diesem Dialog komplett neue Hierarchien aufbauen oder aber für bestehende Pakete Subpakete anlegen. Nachdem Sie ihn mit *Finish* geschlossen haben, legt Eclipse die Pakete an und zeigt sie im *Package Explorer* an. Öffnen Sie nun die Klasse `de.thomaskuenneth.uiproperties.UIProperties` und sehen Sie sich die `package`-Anweisung an. Die IDE hat diesmal das richtige Paket eingetragen. Falls Sie übrigens versuchen sollten, die Klasse *UIProperties* vom Paket `de.thomaskuenneth.uiproperties` nach `uiproperties` zu verschieben, weist Sie Eclipse daraufhin, dass dort schon eine Klasse mit gleichem Namen existiert. Auf diese Weise wird ein ungewolltes Überschreiben verhindert.

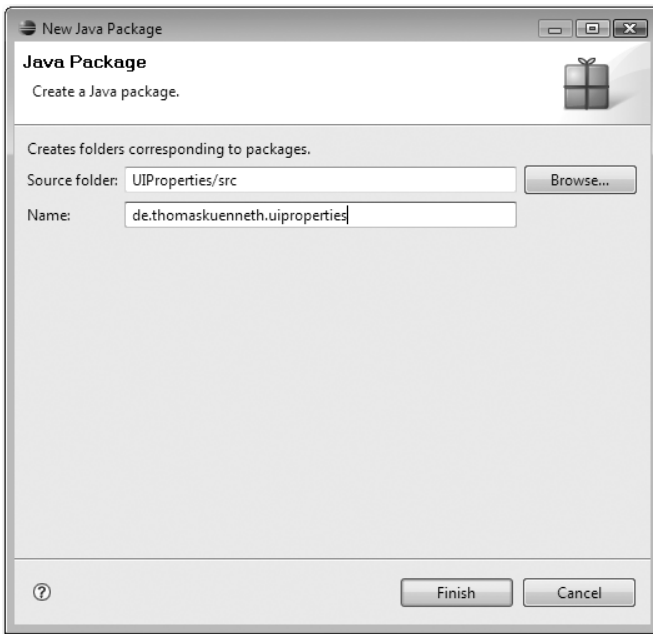


Abbildung 2.60 Dialog zum Anlegen eines neuen Pakets

Ein weiterer Aspekt des Refactorings ist das Umbenennen von Klassen oder Bezeichnen. Klicken Sie im *Package Explorer* bitte eine der beiden Klassen *UIProperties* mit der rechten Maustaste an und wählen Sie **REFACTOR • RENAME**. Sie sehen daraufhin den in [Abbildung 2.61](#) gezeigten Dialog *Rename Compilation Unit*, in dem Sie in *New name* einen neuen Namen eintragen können.

Zahlreiche Schalter beeinflussen, welche Teile des Quelltextes gegebenenfalls angepasst werden. Wenn Sie beispielsweise in Javadoc-Kommentaren Bezug auf eine umzubenennende Klasse nehmen, sollten Sie ein Häkchen vor *Update textual occurrences in comments and strings* setzen. Klicken Sie bitte auf *Finish*, um den Vorgang abzuschließen.

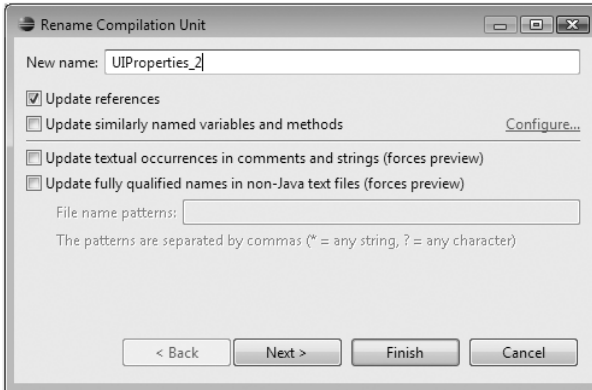


Abbildung 2.61 Dialog zum Umbenennen einer Klasse

Eclipse macht Sie nun auf ein mögliches Problem aufmerksam. Wie solche Hinweise aussehen können, sehen Sie in Abbildung 2.62. Die Klasse *UIProperties* enthält nämlich eine `main()`-Methode, könnte also durch Scripts oder andere Klassen gestartet werden. Durch das Ändern des Klassennamens würden diese Aufrufer die Klasse nicht mehr finden. Da dies in diesem Beispiel nicht der Fall ist, können Sie den Hinweis ignorieren und den Dialog durch erneutes Klicken auf *Finish* schließen.

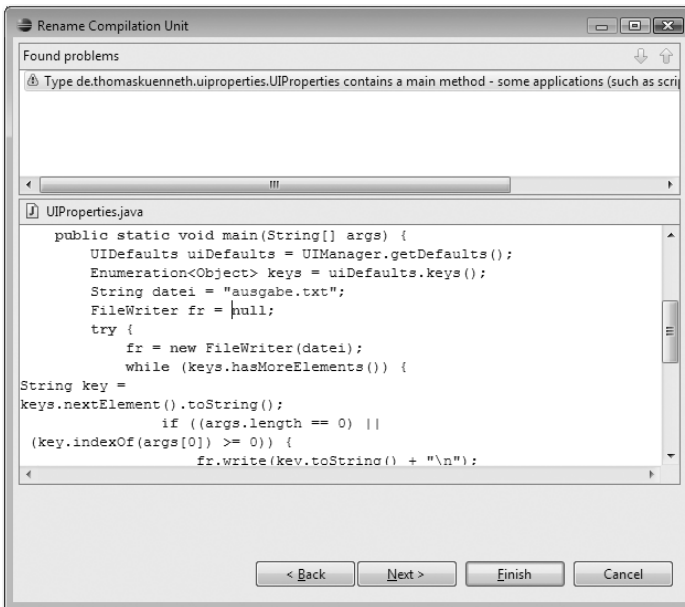


Abbildung 2.62 Hinweis auf ein mögliches Problem nach dem Refactoring

Eclipse erlaubt nicht nur das Umbenennen von Klassen, sondern auch von Variablen. Um dies auszuprobieren, öffnen Sie bitte die Klasse *UIProperties* und markieren, wie in Abbildung 2.63 zu sehen ist, die Variable `args`, den einzigen Parameter der Methode `main()`. Anschließend können Sie durch Klick mit der rechten Maustaste das Kontextmenü des Java-Editors öffnen oder in der Menüleiste **REFACTOR • RENAME** aufrufen. Der schnellste Weg ist wahrscheinlich, die entsprechende Tastenkombination **Alt + ⌘ + R** zu drücken.

Anschließend müssen Sie nur den neuen Variablennamen eintippen und die Taste **↵** drücken. Haben Sie bemerkt, dass der Tooltip einen kleinen blauen Pfeil enthält? Wenn Sie diesen anklicken, öffnet sich ein Menü, das weitere Funktionen zur Verfügung stellt. Beispielsweise können sie mit *Open Rename Dialog* den aus früheren Eclipse-Versionen bekannten Dialog öffnen.

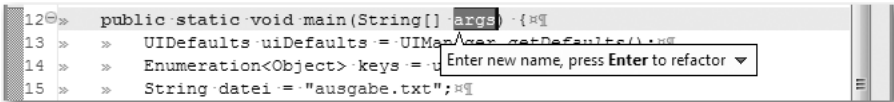


Abbildung 2.63 Umbenennen einer Variablen unmittelbar im Java-Editor

Mit den beiden Funktionen zum Umbenennen und Verschieben lässt sich auf sehr komfortable Weise die Struktur einer Anwendung verändern. Im folgenden Abschnitt stelle ich Ihnen weitere Werkzeuge vor, mit denen Sie Ihre Programme modifizieren können.

Weitere Werkzeuge für das Refactoring

Im Abschnitt *Texte auslagern* habe ich Ihnen eine Möglichkeit vorgestellt, Zeichenketten in externe Dateien zu verschieben. Mithilfe der Funktion **EXTRACT CONSTANT** können Sie Konstanten, die Sie in Ihrem Quelltext verwenden, automatisch einem Bezeichner zuweisen.

Werfen Sie bitte einen Blick auf die Klasse *UIProperties*. Sie enthält die Zeile `String datei = "ausgabe.txt";`. Die Zeichenkette könnte stattdessen auch als Konstante definiert werden. Markieren Sie bitte `"ausgabe.txt"` und wählen Sie anschließend **REFACTOR • EXTRACT CONSTANT**.

Sie sehen daraufhin den in Abbildung 2.64 gezeigten Dialog *Extract Constant*, in dem Sie der zu erstellenden Konstante einen Namen geben sowie ihre Sichtbarkeit festlegen müssen. Klicken Sie bitte auf **OK**, um den Vorgang abzuschließen. Eclipse fügt nun eine entsprechende Zeile in den Quelltext ein.

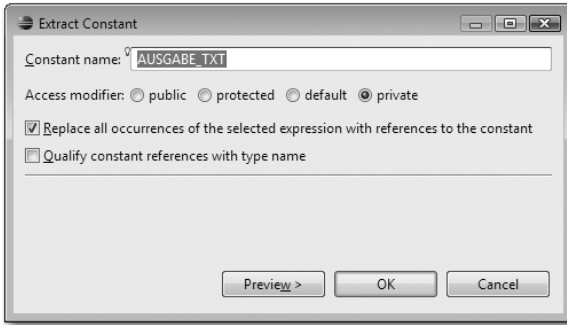


Abbildung 2.64 Dialog zum Extrahieren einer Konstanten

Eine weitere, äußerst praktische Funktion ist, aus Bereichen des Quelltextes eine eigene Methode zu erzeugen. Dies kann aus verschiedenen Gründen sinnvoll sein. Zum einen können Sie auf diese Weise lange, und damit vermutlich schlecht lesbare Methoden entzerren. Ein anderes Einsatzgebiet ergibt sich, wenn Sie in Ihrem Code ähnliche oder identische Fragmente entdecken. Diese sind klassische Kandidaten für das Refactoring.

Bitte markieren Sie die vollständige `while()`-Schleife der Methode `main()` und klicken Sie anschließend auf **REFACTOR • EXTRACT METHOD**. Sie sehen daraufhin den in Abbildung 2.65 gezeigten Dialog zum Extrahieren von Methoden.

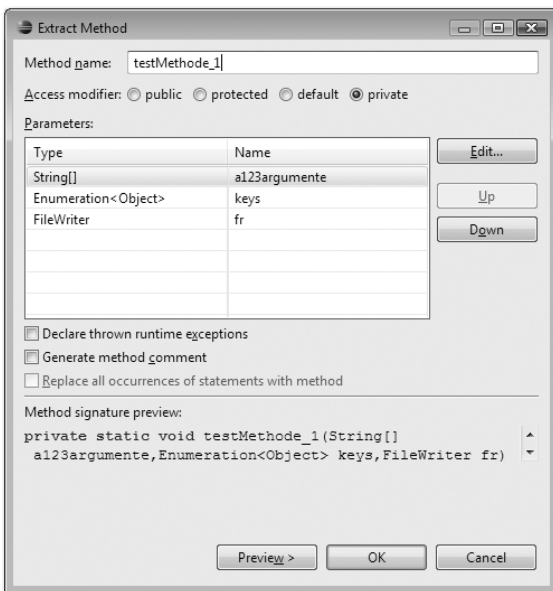


Abbildung 2.65 Dialog zum Extrahieren von Methoden

In diesem Dialog legen Sie die Parameter sowie den Namen der neu anzulegenden Methode fest. Klicken Sie auf *Preview*, um den Quelltext zu sehen, den die IDE generiert, wenn Sie den Dialog mit *OK* schließen. Wie Sie in Abbildung 2.66 sehen, listet Eclipse unter *Changes to be performed* alle Änderungen auf. Sie können einzelne Änderungen durch Entfernen des entsprechenden Häkchens deaktivieren.

Anhand der Vergleichsansicht, die den größten Teil des Dialogs einnimmt, können Sie sehr genau nachvollziehen, wo Quelltext eingefügt oder gelöscht wird. Klicken sie bitte auf *OK*, um die neue Methode zu erzeugen.

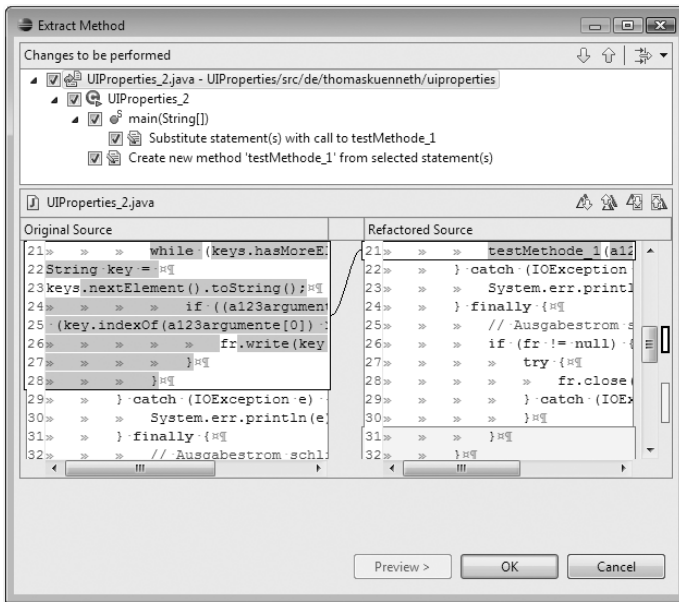


Abbildung 2.66 Vorschau des generierten Quelltextes

Das Menü **REFACTOR** bietet zahlreiche weitere Funktionen, mit denen Sie die Struktur Ihrer Programme verändern können. Bitte probieren Sie diese anhand kleiner Beispielanwendungen aus.

2.4 Zusammenfassung

Sie haben in diesem Kapitel sehr viel über komfortables Arbeiten mit Eclipse erfahren. Ihnen sind nun wichtige Konzepte wie Sichten und Perspektiven vertraut. Im nächsten Kapitel möchte ich Ihnen das Arbeiten mit Projekten sowie den Umgang mit dem Arbeitsbereich näherbringen.

Wenn Sie sich ausführlicher mit dem spannenden Thema Refactoring auseinandersetzen möchten, empfehle ich Ihnen eines der Werke von Martin Fowler, einem ausgewiesenen Experten auf diesem Gebiet. Sofern Sie gute Englischkenntnisse haben, rate ich Ihnen zur englischen Originalfassung *Refactoring: Improving the Design of Existing Code*. Zu diesem Buch gibt es auch eine deutsche Übersetzung: *Refactoring. Oder wie Sie das Design vorhandener Software verbessern*. Ebenfalls sehr empfehlenswert ist das englische *Refactoring Workbook* von William C. Wake. Die vollständigen bibliographischen Angaben finden Sie im Literaturverzeichnis im Anhang.

Index

.jar-Archive 299, 339

A

ActionBar Advisor 198
ActionListener 312, 313, 322
actionPerformed 313, 322
Add JRE 50
Advisor 198
AJAX 360
AJAX Toolkit Framework 377
Aktualisierungen 27
Alias 24
Annotation 85
Ant 157, 160
 Sicht 160
Ant Builder 139
Ant-Script 137
Ant-Scripts 161
Applet 343
Arbeitsbereich 22, 23, 24, 117, 118, 128,
 131, 269, 375
 Anlegen 118
 Clean 140
 Clean Builds 140
 Dateisystem 120
 Logische Struktur 120
 Löschen 119
 Verzeichnisauswahl 118
 Wechseln 118
Arbeitsbereichsordner 23, 24, 63, 118,
 363
Arbeitsbereichsverzeichnis 23, 38
Arguments 207
ArithmeticException 225
Aufgaben 33, 81
Auschecken von Projekten 269
Automatische Builds 140

B

bash 20, 217
BeanShell 86
Bedingte Breakpoints 231
Begleit-DVD 18, 21

Benutzeroberfläche 300
Bibliotheken 150
 Mehrteilige 150
Bookmarks 40, 44, 83
BorderLayout 315, 321, 329
Branding 197, 199, 200
Breakpoints 64, 214, 219, 220, 226, 229,
 234
Build automatically 140, 141
Build Path 141, 142, 144
Builder 138, 139
Buildfile 157

C

Callisto 302
Cheat Sheets 47
Check out 271
Class Load-Breakpoints 227
Client-Server-System 215
Close Project 37
Collapse All 41
Commit 268, 269, 275, 282, 285, 289,
 295
Compare 274, 296
Compiler 215
Concurrent Versions System 254
Confirm Project Delete 37
Console 59, 87, 88, 134, 209, 212, 283,
 351, 359, 382
Container 320
Content Assist 349
Contents 40, 42
Copy Settings 118
Cursor-Position 71
CVS 254, 272
 Linux 255
 Mac OSX 256
 Windows 254
CVS-Client 263
CVSNT 254, 256
CVS-Repository 128
CVS-Server 263
Cypal Studio for GWT 368

D

Daemon 255, 263
 Darstellungsbereich 55, 57
 Debug 54, 208, 210, 214, 218, 219, 236, 243
 Debuggen
 Visuelles 206
 Delete 37
 Delete All Bookmarks 44
 Delete Selected Bookmark 44
 Desktop 22, 23
 Detached 58
 Disconnect 297
 Display 211, 212
 Distribution 21
 Dokumentation 19
 DOM 360
 Drop To Frame 243
 Dynamic Help 46
 Dynamic Web Project 354, 369, 380

E

Eclipse
 Aktualisierung 26
 Aufruf-Parameter 27
 Benutzervorgaben 25
 Hilfe 40
 Installation 21
 Installation (Linux) 22
 Installation (Mac OSX) 23
 Installation (Windows) 21
 Installationsverzeichnis 18
 Maximize 41
 Start 24
 Tipps and Tricks 45
 Willkommensbildschirm 25, 40, 54
 Zielformat 21
 Eclipse Classic 21
 Eclipse Debuggers 215
 Eclipse Foundation 21
 Eclipse Homepage 21, 22, 23
 Eclipse Platform 21, 174, 179, 195, 373
 Eclipse Platform Core 177
 Eclipse Rich Client Platform 54, 195, 197
 Eclipse SDK 21, 22, 23, 177, 179
 Edit 91
 Edit Local Site 377

Editoren 61, 337
 Eingabeaufforderung 19, 217, 259
 EmptyBorder 328
 Entry Points 367
 Enumeration 35
 Ersetzen 100
 Europa 302
 Europa Discovery Site 344, 353
 Exception Breakpoints 224
 EXIT_ON_CLOSE 31
 Export Wizard 185
 Expressions 212, 213
 Extension Points 178
 Externalize Strings 78
 Extreme Programming 108

F

Facetten 345
 Failure Trace 248
 Fakultät 206
 Fast Views 59, 60
 Favoriten 156
 Feature Manifest Editor 186, 187
 Feature Selection 189
 Features 179, 186
 Fehlersuche 205
 Find and Install 27, 169
 Finder 261
 Firewall 255, 257
 Force Return 235, 240
 Formatter 74

G

geschützte Bereiche 301
 getSystemProperties() 35
 Google Web Toolkit 361
 GUI Forms 304, 318
 GUI Properties 306, 310, 312, 319, 320, 322
 GUI-Designer 300, 301
 GUI-Editoren 300
 GWT Browser 361

H

HEAD 272
 Help 40

Hierarchy 56, 57
 History 95, 288
 Hit Counts 220, 229
 Hosted Mode 361, 364, 367
 HTML-Editor 350

I

Import 38, 286
 Indentation 72
 Index 40
 inetd 255, 256
 Information 187
 Inkrementelle Builds 140
 Inspect 212, 235
 Install 172
 Install/Update 27
 Installation History (Plug-ins) 175
 Installationsort 171
 Internet Explorer 353
 Introduce Factory 92

J

J2EE Preview 354, 357, 380
 J2EE Standard Tools 353
 Java 48, 54
 Installation (Mac) 20
 Java Build Path 133, 146, 151, 245
 Java Build path 145
 Java Builder 138, 140
 Java Development Kit 18
 Java Development Toolkit 215
 Java Development Tools 21, 179
 Java EE 356
 Java Project 29
 JAVA_BIN 19, 20
 Javadoc location 133
 Javadoc location path 133
 Javadoc-Dokumentation
 Erzeugen 134
 Java-Dokumentation 27
 Java-Laufzeitumgebung 18
 Java-Laufzeitumgebungen 49
 Java-Perspektive 29
 Java-Web-Anwendungen 344
 JComboBox 326
 JDialog 318, 328
 JDK 19

JDT 179
 JFace 178
 JForm Designer 302
 JFrame 29, 31
 Jigloo 299, 303
 JMenuItem 313
 JPanel 323, 330, 331
 JScrollPane 331
 JTable 316
 JUnit 68, 244, 249
 JUnit Test 247
 JUnit Test Case 245
 JUnit Test Suite 249

K

Key Assist 40
 Klappmenü 58
 Klassen hinzufügen 29
 Konflikte 274, 275, 290, 293

L

Launch Configurations 38, 153, 163
 LayoutManager 300, 315, 320, 328
 Lesezeichen 44, 83
 Libraries 146, 245
 Line Breakpoint 219
 Link with Editor 96
 Linked Resources 124
 Linux-Distribution 19
 Local History 86, 91, 93, 285
 Local Site 277
 Lokale Repositories 254

M

Mac OS X 20, 23
 main() 29
 Manuelle Builds 140
 Mark as Merged 296
 Mark Completed 34
 MDI 315
 merge 290
 Method Breakpoint 219
 Modell 334
 Module 266, 280, 367
 Multiple Document Interface 53, 315

N

native launcher 199
 Navigation 80
 Navigator 95, 96, 103, 104, 120, 122,
 137, 194, 371
 New Archived Site 377
 New Java Class 29
 New JUnit 3 Test 245
 New Search List 43
 New Task 34

O

Object Linking and Embedding 63
 OLE 63
 Open Debug Dialog 207, 218
 Open Run Dialog 372
 Ordner suchen 38
 Organize Favorites 156
 Organize Imports 36
 Orientation 60
 OSGi Framework 375
 Outline 57, 95, 96, 97, 159, 222, 311,
 312, 313, 316, 320, 323, 332, 349
 Overview 185, 187, 200

P

Package Explorer 29, 37, 39, 54, 55, 57,
 63, 64, 80, 82, 86, 88, 90, 91, 95, 97, 99,
 104, 108, 128, 131, 133, 137, 158, 173,
 184, 187, 188, 194, 198, 245, 246, 250,
 269, 274, 275, 280, 285, 286, 288, 291,
 292, 297, 304, 332, 339, 356, 364, 380
 pane 55
 Parameter 27
 part 55
 Patchlevel 20
 PATH 19, 20
 PDE 179
 Perspektiven 48, 53, 64
 command group 69
 Pfadvariable 125
 Platform Runtime 178
 Plug-in Development 192, 375
 Plug-in Development Environment 21,
 179, 181

Plug-in Manifest Editor 184, 194, 195,
 196
 Plug-ins 168, 178, 194, 301, 375
 Deaktivieren und Entfernen 175
 Eigene 181
 Manuelle Installation 168
 Update Manager 169
 Verwaltung 174
 Preferences 25, 27, 38, 49, 52, 71, 75,
 122, 123, 140, 141, 151, 153, 215, 236,
 238, 239, 305, 369, 375
 Print topics 41
 Problems 47, 59, 64, 350
 Product Configuration 174, 191
 Product Configuration Editor 191, 192,
 199, 200, 201
 Products Extension Points 180
 Produkt 177, 179
 Produktaktualisierung 28
 Profile 72
 Programmaktualisierungen 26
 Projects 134, 138, 140, 145
 Projekt
 anlegen 22, 24
 auschecken 286
 einchecken 280
 importieren 38
 Projektverwaltung 36, 37
 Properties 35, 357
 Public JRE 18

R

RAP Widget Toolkit 373
 RCP-Anwendungen 195
 Redo 90
 Refactor 108, 132
 Refactoring 100, 108, 301
 Referenzen 144
 Refresh 41, 122
 Registerkarte 61
 Remote Debugging 215, 217, 219
 Remote Launch Configuration 216
 Remote Site 169, 277, 303
 Rename 90
 Rendering Kits 374
 Repositories 254, 259, 262, 269, 273,
 280, 286
 Repository hinzufügen 279

Repository Path 263
 Ressourcen 120
 Restore 41
 Revert 176
 Rich Ajax Platform 373
 Rich Client Application 195
 Round-Trip-Editing 301
 Run 102, 154, 156, 163, 208, 210, 211, 234
 Run on Server 359, 380
 Run to Line 234
 Runtime Type 378

S

Save and Launch 32
 Schlüssel 78
 Schnellzugriffsleiste 54, 65
 Scrapbook 86, 211
 Search 46, 102, 104, 105
 Search Results 40, 42, 377
 Search scope 42
 Select Search Scope 42
 Servers 350
 Servlet 343, 356
 Servlet-Container 343
 setDefaultCloseOperation() 31
 Shell 20
 Show Current Topic 41
 Show result categories 42
 Show result descriptions 42
 Sichten 29, 33, 37, 46, 53
 Site Manifest Editor 188
 Softwareaktualisierung 20
 Software-Updates 27
 Splash Screens 180, 196
 Sprungmarken 83
 Startmenü 22
 Startup and Shutdown 123
 Static Web Project 345
 Statuszeile 55
 Step Filter 236
 Step Into 209, 233, 234, 236, 239, 242
 Step Into Selection 234
 Step Over 209, 219, 233, 234, 236, 239, 242
 Step Return 234
 Subclipse 277

Subversion 254, 258
 Linux 260
 Mac OSX 261
 Windows 259
 Suchen 100
 Inkrementell 107
 Reguläre Ausdrücke 107
 Suchen und Ersetzen 105
 Suchlisten 42
 sudo 20
 SVN Repository 279, 280, 293
 SVN Repository Exploring 279
 svnadmin 259
 svnservice 260, 262
 Swing 28, 300
 SWT 178
 Synchronize 295
 System-Properties 28
 SystemPropertyViewer 29
 Systemsteuerung 254

T

Tab Group 58
 Tab size 72
 tabbed notebook 57
 Target Platform 375
 Target Runtime 345, 354, 370, 380
 TaskPane 150
 Tasks 33, 64, 67, 81, 158, 246
 Team 69
 Team Synchronizing 276, 294, 295
 Terminal 261
 Test 205
 Testfall 244
 Themengebieten 42
 TODO 33
 Toggle Breakpoint 218
 Toolbar 85
 Tooltip 32, 36

U

Ubuntu 255
 UIProperties 101
 Umgebungsvariable 19
 Undo 90
 Unit-Tests 243
 Update 170, 296

Index

Update Manager 169, 172, 174, 188, 277,
303, 344, 353, 377
Update Site 170, 172, 188, 190
Update Site Map 190
Update Site Project 188
Updates 344
Use Step Filters 236
User Libraries 151, 153

V

Validate 350
Variables 64, 213, 219, 240, 241
verknüpfte Ressourcen 120
Verknüpfung 22, 23
Versionsverwaltung 21, 254
Vervollständigungen 31
vi 256, 257
Video-Training 388
Visual Editor 302

W

Watch Expressions 213
Watch Points 221
Web Mode 361, 364

Web Standard Tools 344
Web Start 18
Web Tools Platform 353
WindowBuilder Pro 302
windowClosing 311, 319
WindowListener 311, 319
Workbench 46, 54, 120, 177, 178, 198
Workbench Advisor 198
Workbench User Guide 27
Workbench Window Advisor 198
Working Set 97, 103, 214
Workspace 24, 117, 140, 178
Workspace Launcher 22, 23, 24, 25, 63,
119, 120

X

Xcode 256
Xcode Developer Tools 21, 256
XMLHttpRequest 360

Z

Zeilennummer einblenden 71
Zimbra 380
Zimbra Ajax Toolkit 378