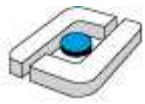
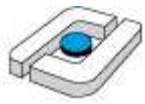
Nutzungshinweise für Eclipse



Thema:	Nutzung der SW-Entwicklungsumgebung Eclipse
Autor:	Prof. Dr. Stephan Kleuker
Version / Datum:	5.4 / 8.11.2010
Empfänger:	Teilnehmer der Lehrveranstaltungen im Bereich SW-Technik

Diese Anleitung fasst einige Tipps und Tricks zum Umgang mit Eclipse in meinen Lehrveranstaltungen zusammen. Dieser Text ist nicht als vollständiges Manual anzusehen. Er wird individuell an die aktuell laufenden Veranstaltungen angepasst, so dass auch Plugins beschrieben werden, die in der momentan in der HS installierten Version nicht vorhanden sind. In der Lehrveranstaltung, insbesondere in den Praktikas wird für konkrete Aufgaben auf einzelne Kapitels dieses Texts verwiesen.

Eclipse ist ein sehr komplexes Werkzeug, das allerdings dann sehr einfach zu bedienen ist, wenn man sich an die typischen Arbeitsabläufe hält, die fast kein Detailwissen über Eclipse benötigen. Eclipse ist deshalb sehr gut für Anfänger geeignet. Jeder Nutzer ist aufgefordert, sich selbst intensiver mit Eclipse zu beschäftigen, da man dann einige eigene Arbeiten noch vereinfachen kann. Eclipse unterstützt diese Experimentierfreudigkeit durch gute Help-Files und die einfache Chance, ursprüngliche Einstellungen wieder herzustellen.



1 Vorbemerkungen

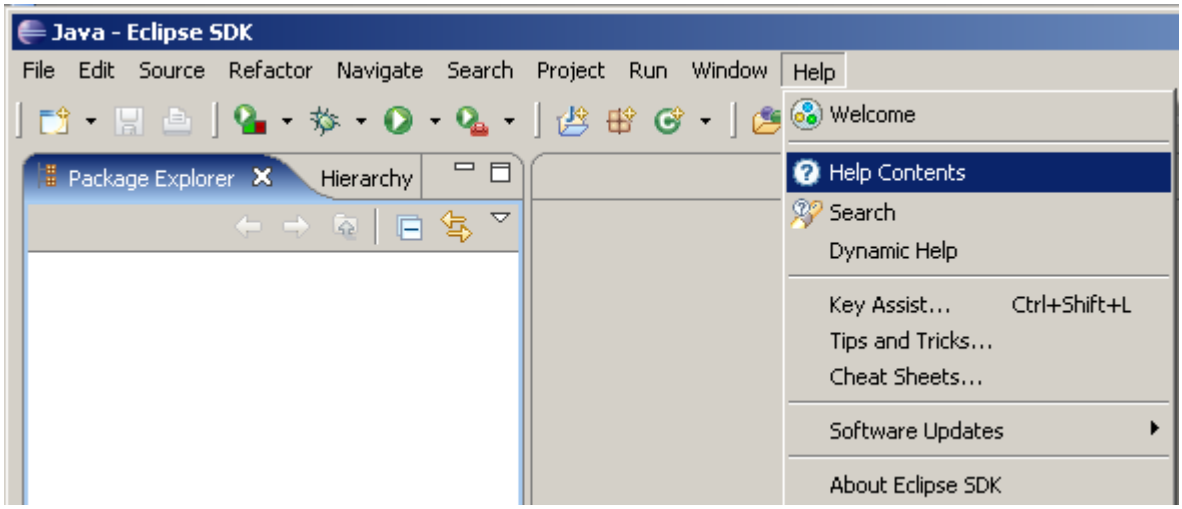
In verschiedenen Lehrveranstaltungen wird Eclipse als durchgehendes Werkzeug genutzt, wobei verschiedene Plugins zur Erweiterung der Funktionalität genutzt werden. Die verwendete Version mit allen installierten Plugins kann aus dem Dozentenordner kopiert werden. Die Installation erfolgt durch einfaches Auspacken, wodurch ein Ordner eclipse entsteht, der das Programm eclipse.exe enthält. Die aktuelle Datei ist typischerweise in einem Unterordner von

http://www.edvsz.fh-osnabrueck.de/kleuker/index.html

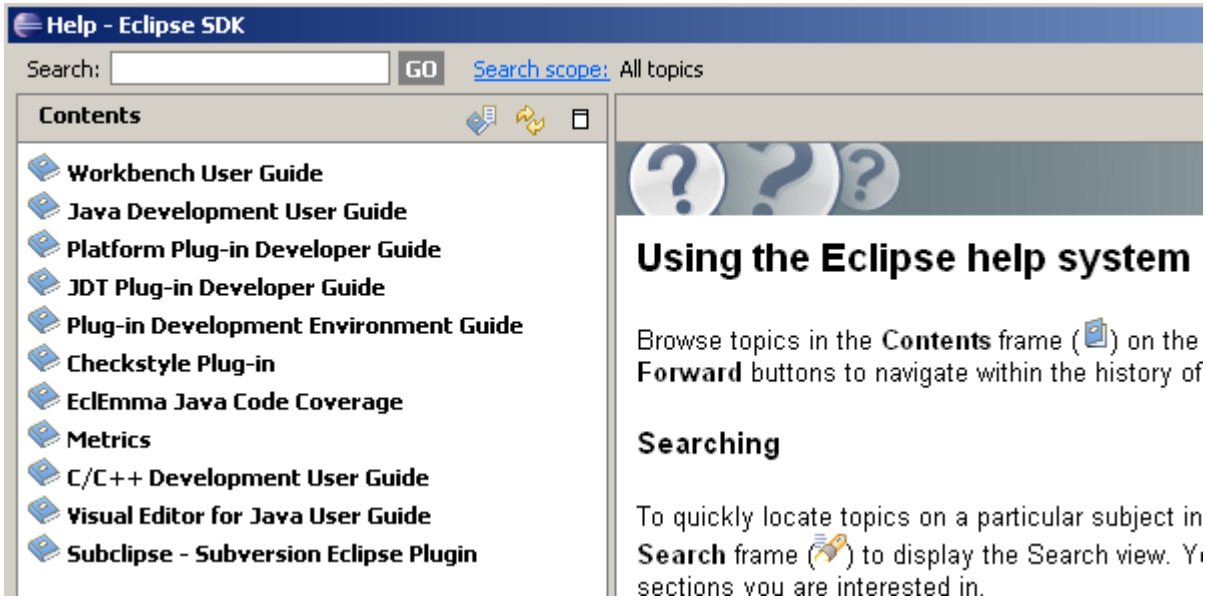
auffindbar, wobei weitere Informationen den Vorlesungen zugeordnet sind.

Damit Eclipse funktioniert, ist vorher Java, genauer ein JDK ab der Version 5 zu installieren. Für C++ wird ein korrekt installierter C/C++-Compiler benötigt, der entweder aus einer Cygwin-Installation oder einer Mingw-Installation stammt. Bei der Installation ist u. a. die korrekte Setzung der Pfadvariablen (PATH) zu beachten.

Für Eclipse sei einleitend auf die integrierten Help-Dateien verwiesen, die unter „Help > Help Contents“ erreichbar sind.



Das Help-System enthält auch Informationen zu den vorhandenen Plugins, insofern diese die Help-Informationen mitliefern.

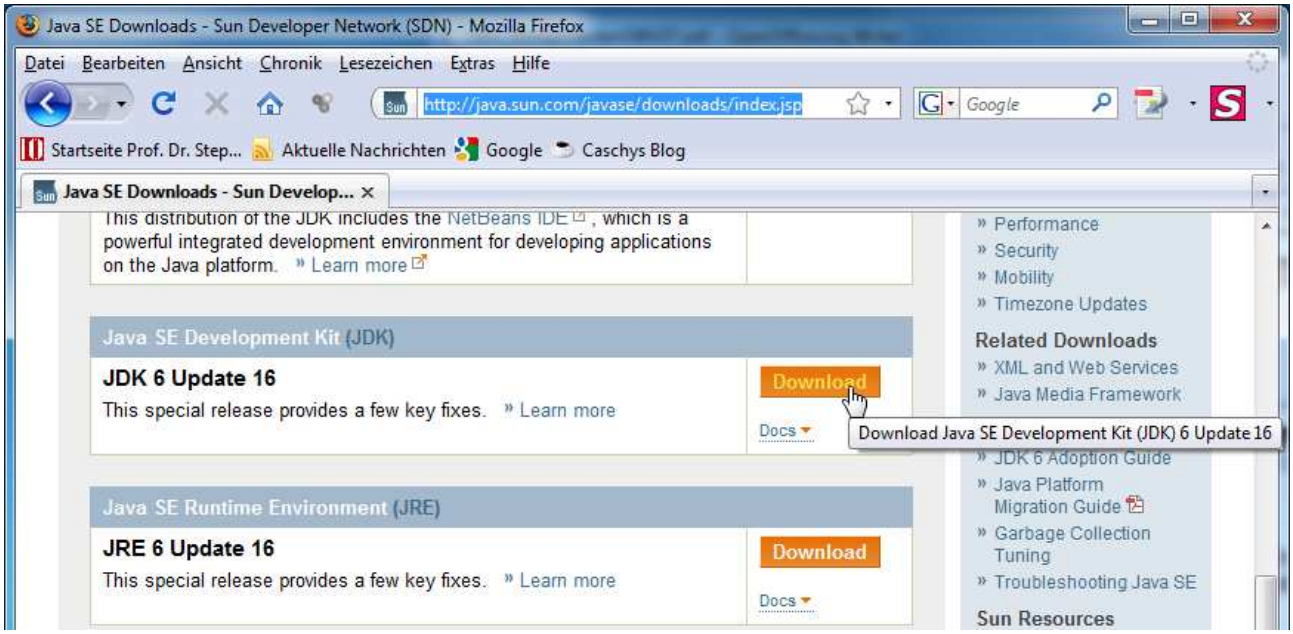


Im Folgenden bezieht sich die textuelle Beschreibung immer auf das nachfolgende Bild. Bei einzelnen Bildern können nach dem Bild noch Kommentare ergänzt werden.

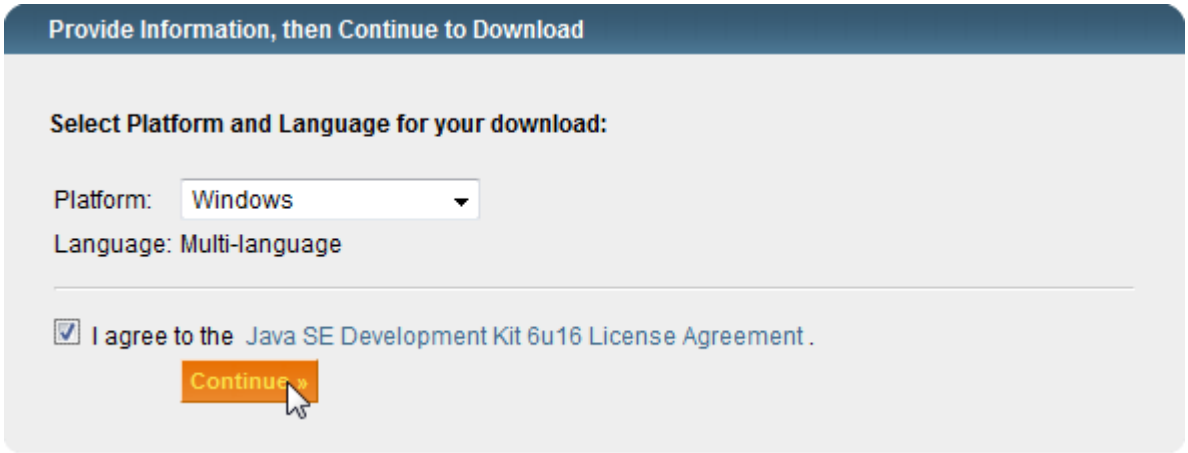
Generell benötigen Anfänger sehr wenige Befehle um sinnvoll mit Eclipse arbeiten zu können. Bei Zeit und Interesse ist jeder aufgefordert, sich z. B. durch das Lesen der Tutorials in den Help-Files intensiver zu informieren. Im Text steht häufiger ein „z. B.“, um anzudeuten, dass es verschiedene Wege gibt, die gleiche Aktion durchzuführen.

2 Installation von Java

Java wird direkt von der Sun-Seite http://java.sun.com/javase/downloads heruntergeladen, dabei benötigt man das Development Kit (JDK) und nicht nur die Runtime-Version.

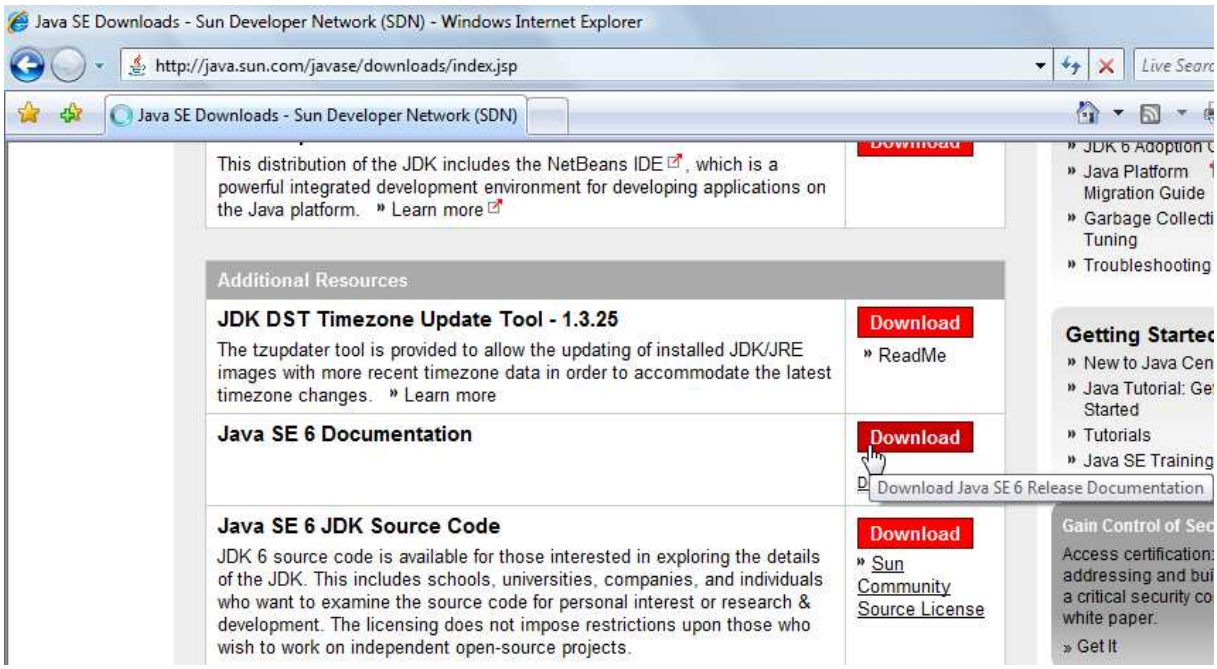


Man wählt das Betriebssystem (hier die Beschreibung für Windows) und setzt den Bestätigungshaken.



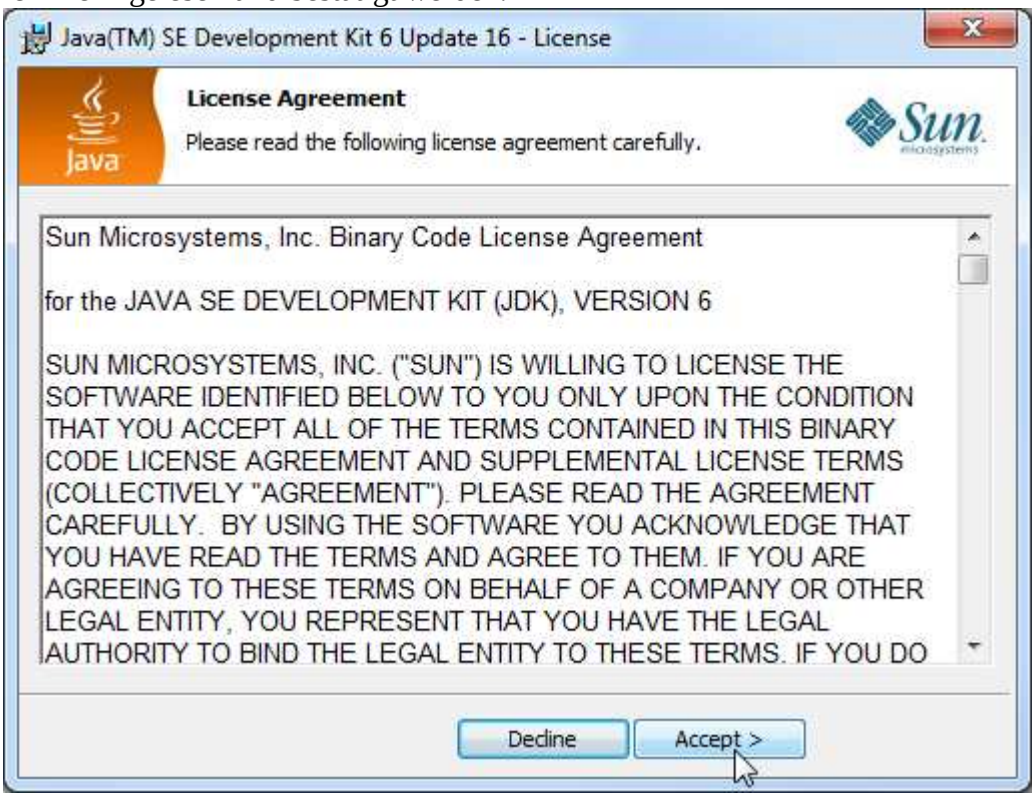
Danach kann Java heruntergeladen werden.

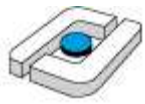
Weiterhin ist es sinnvoll, von Sun die Dokumentation zu Java, JavaDoc, herunterzuladen, was hier nicht weiter betrachtet wird.



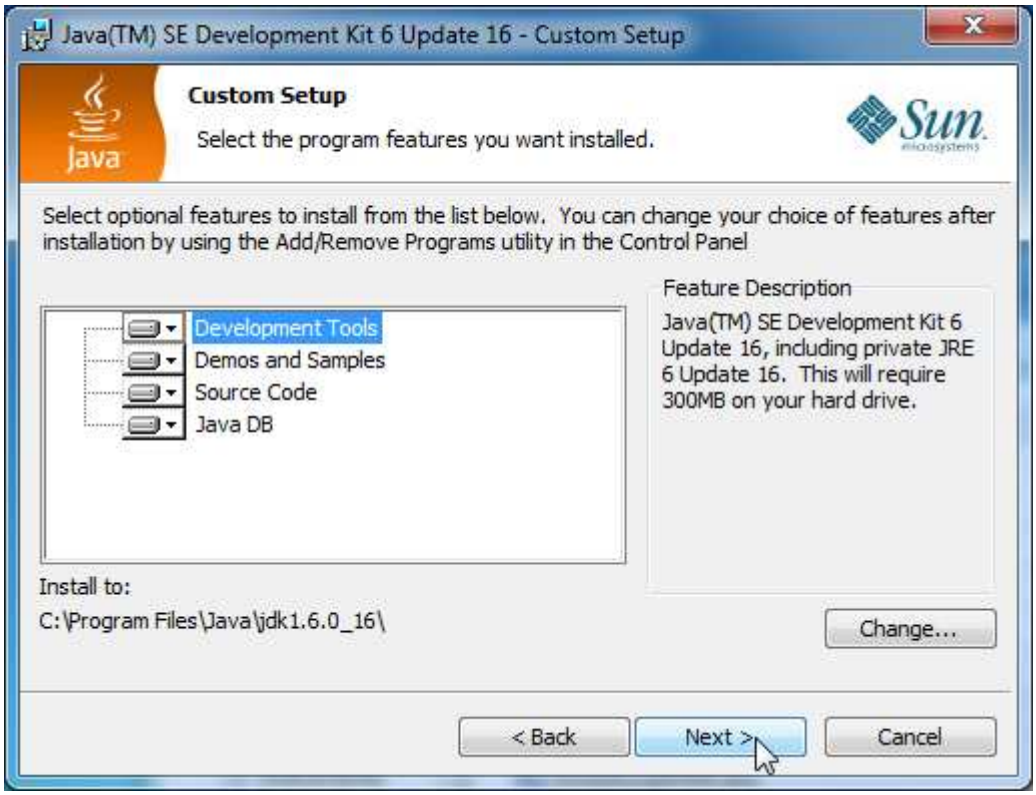
Die Installation von Java wird durch einen Doppelklick auf die heruntergeladene Datei gestartet.

Nach einer eventuell notwendigen Bestätigung, dass die Installation durchgeführt werden soll, muss die Lizenz gelesen und bestätigt werden.

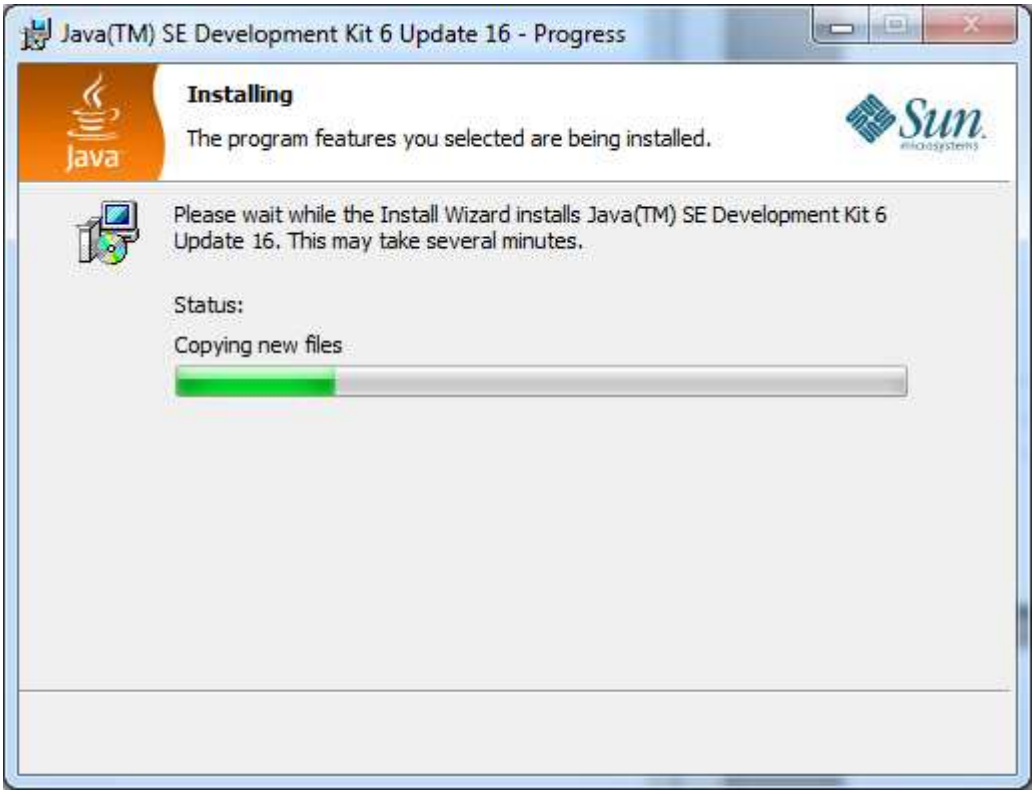




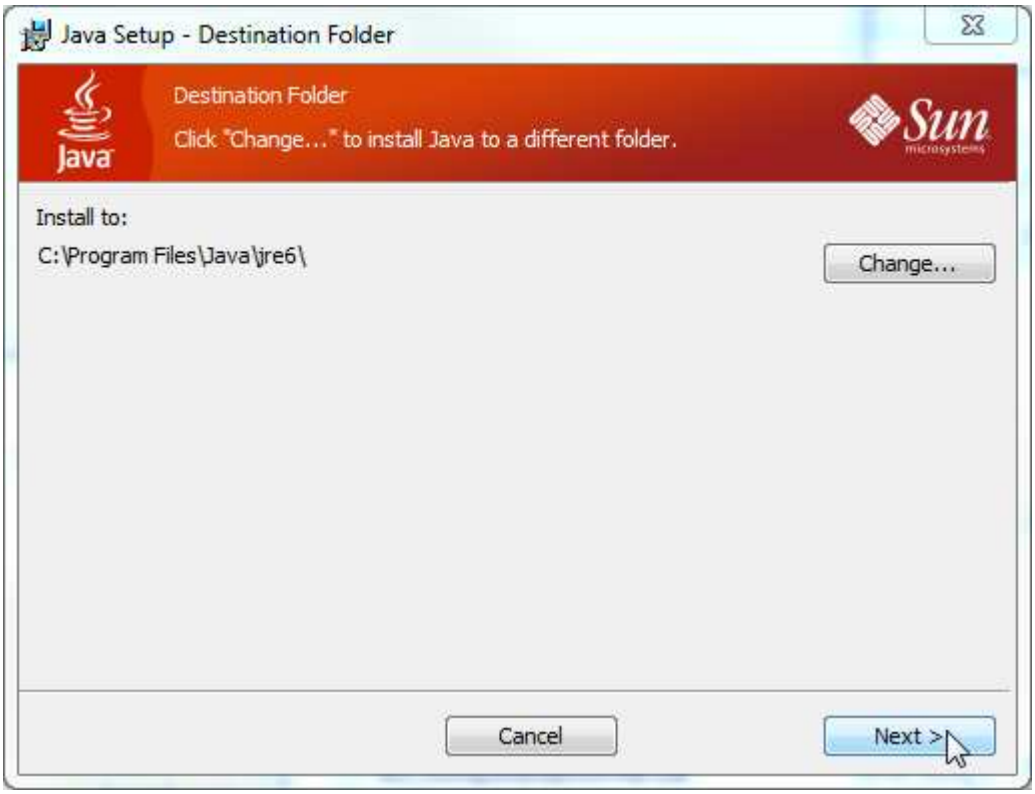
Die Default-Einstellungen können übernommen werden, man sollte sich den Installationspfad merken.



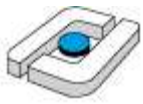
Die Installation benötigt etwas Zeit.



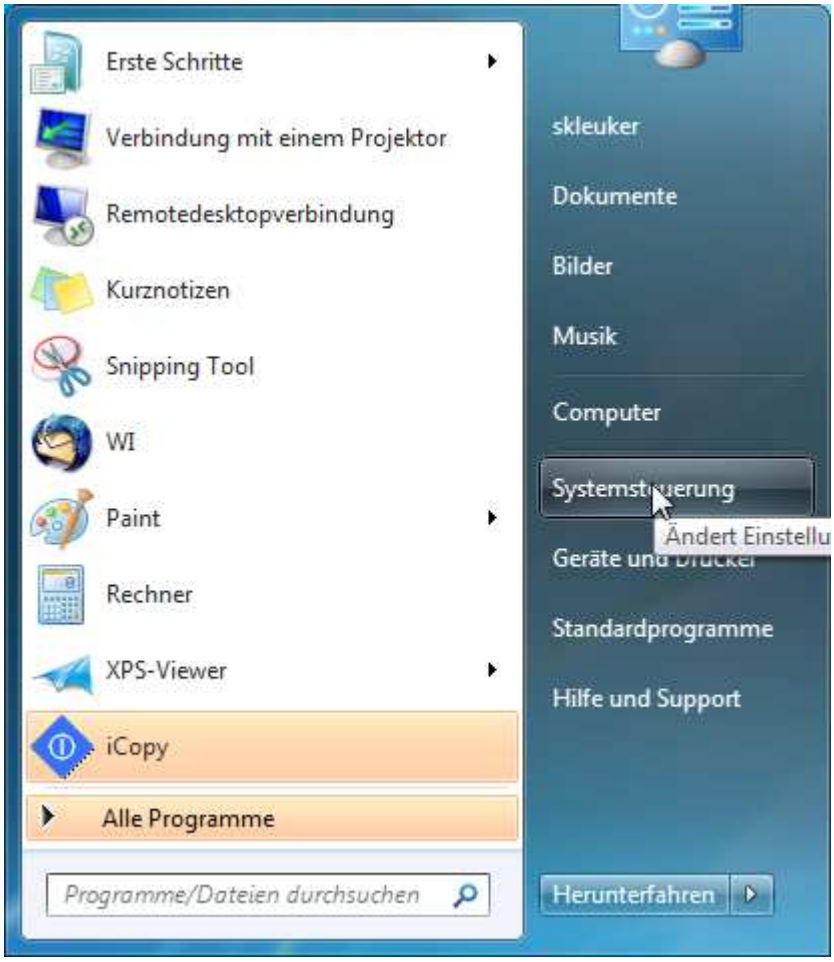
Man wird aufgefordert, den Installationsort für die JRE anzugeben. Der Vorschlag wird hier übernommen.



Danach wird die Java-Installation abgeschlossen.

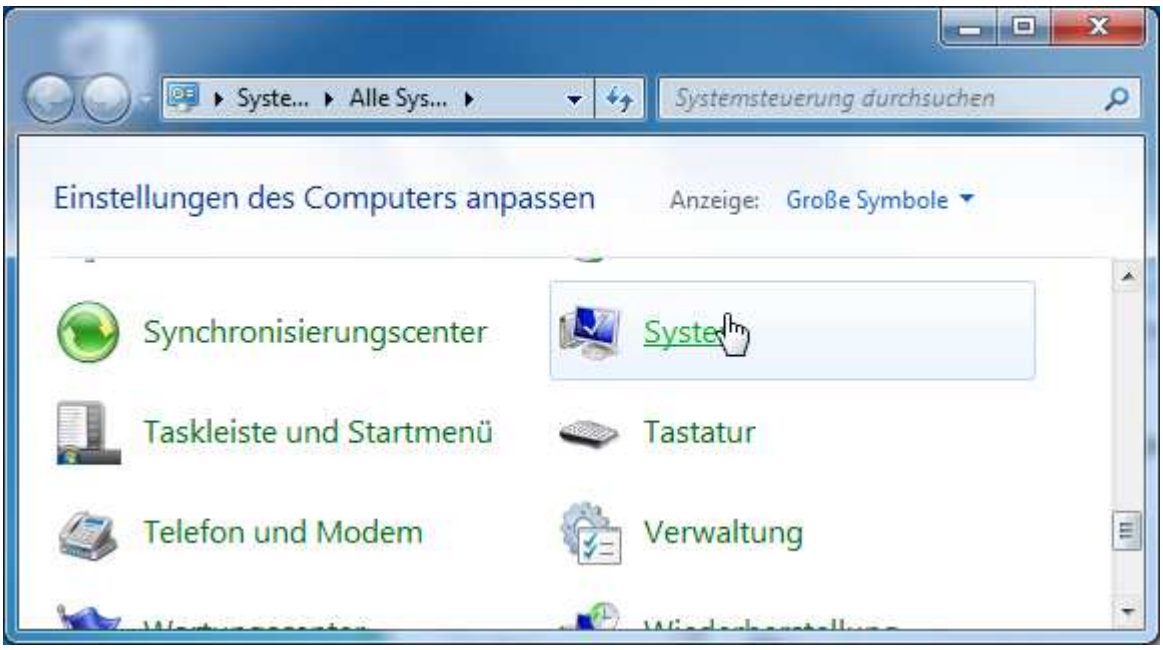


Zum Abschluss sollte noch die Systemvariable JAVA_HOME auf das Installationsverzeichnis gesetzt werden. Ein Weg dazu sieht wie folgt aus. Man wählt die



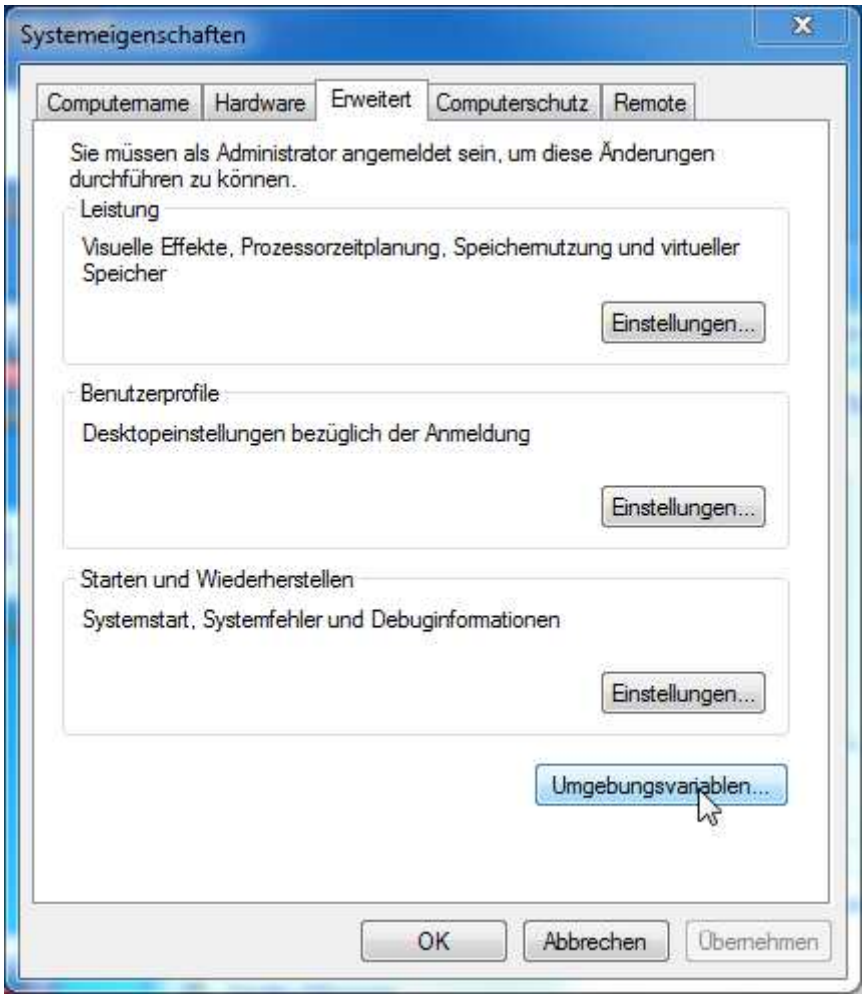
Systemsteuerung im Start-Menü.

Danach macht man einen Doppelklick auf „System“.

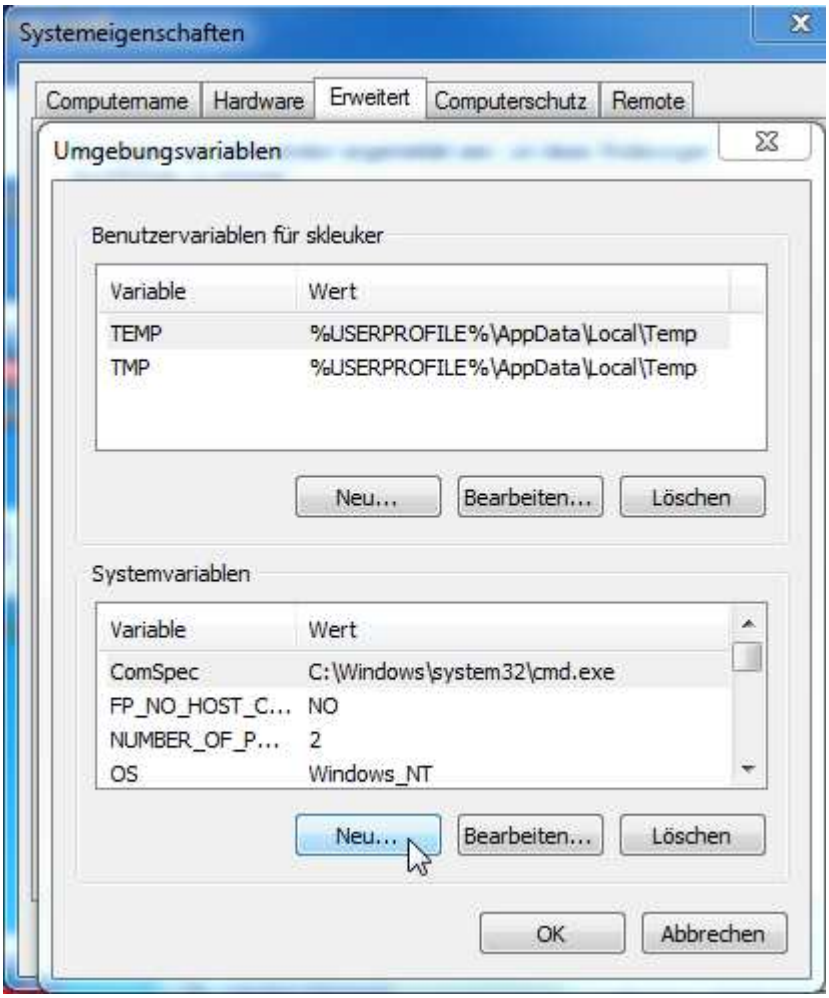


Hier wählt man den Punkt „Erweiterte Systemeinstellungen“ mit einem einfachen Klick aus.

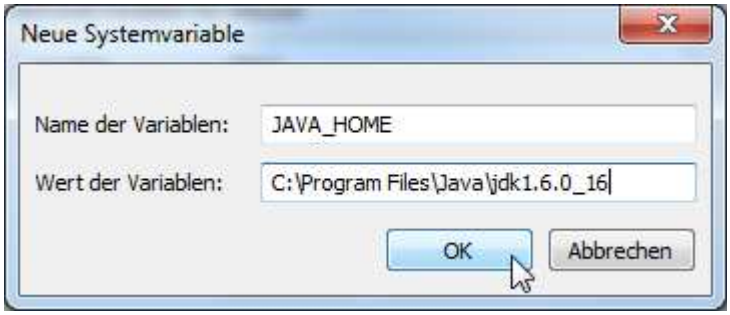
Man nutzt den Reiter „Erweitert“ und klickt unten auf „Umgebungsvariablen“.



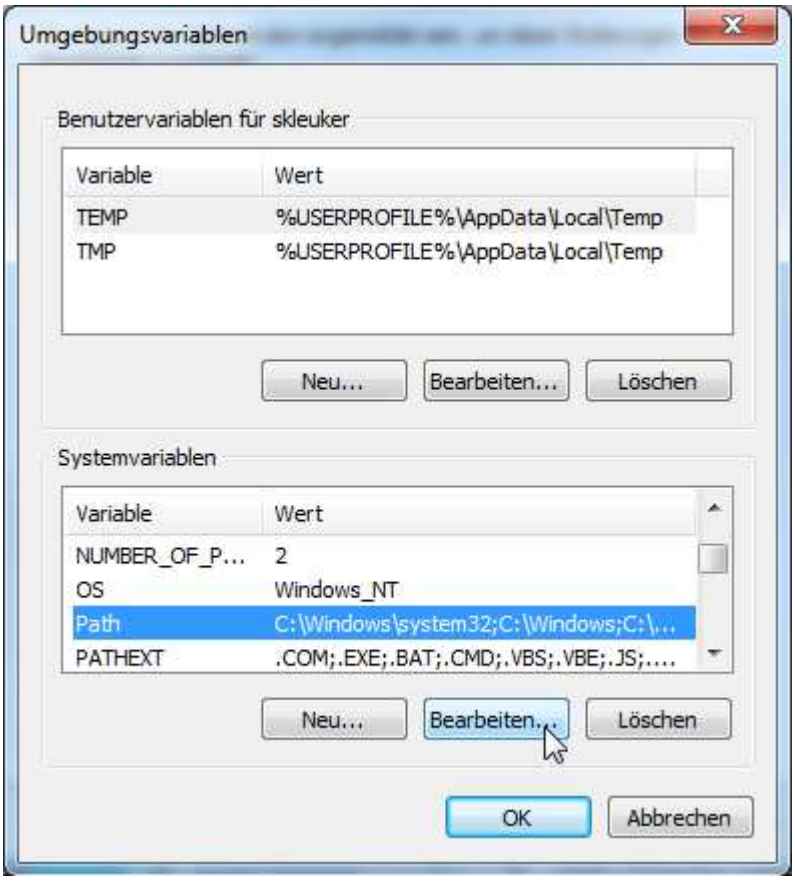
Administratoren des Systems nutzen den Knopf „Neu...“ unter Systemvariablen, andere Nutzer den Knopf unter Benutzervariablen.



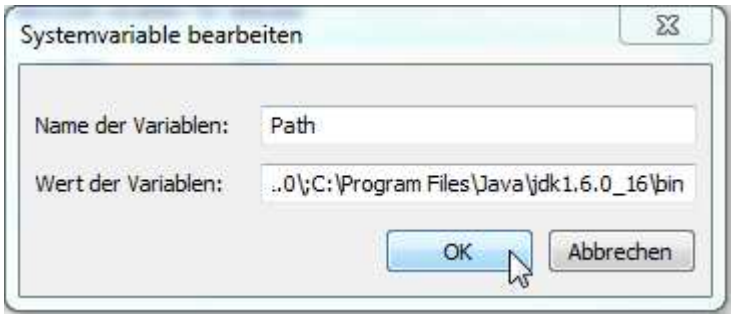
Hier trägt man den Variablennamen sowie den Pfad ein und bestätigt die neue Variable mit „OK“.

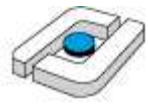


Abschließend muss Java in die PATH-Variable eingetragen werden. Dazu wird die Path-Variable ausgewählt und „Bearbeiten...“ geklickt.



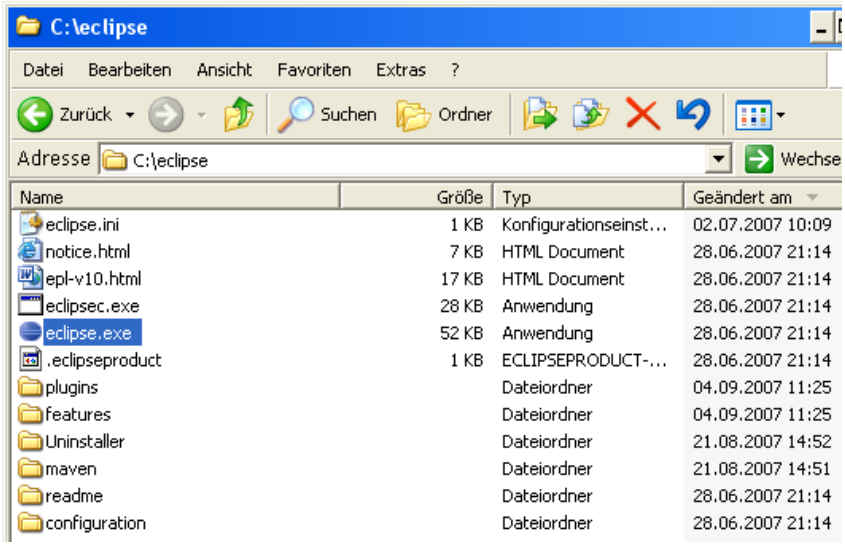
Der Pfad muss irgendwo um das Java-Verzeichnis verlängert um \bin ergänzt werden. Danach werden alle Änderungen mehrfach über „OK“ bestätigt. Man beachte, dass einzelne Einträge in der Pfad-Variablen mit einem Semikolon getrennt sind.





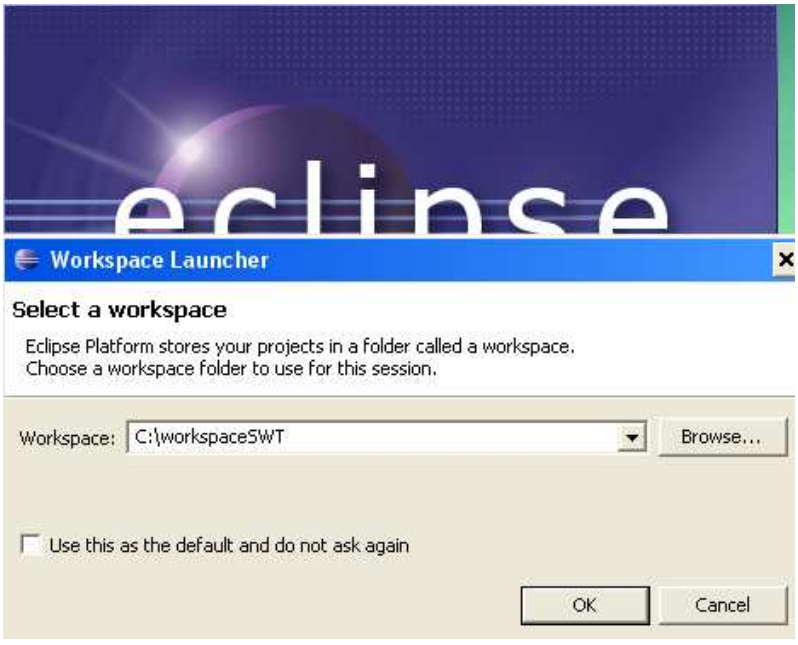
3 Einstieg in Eclipse

Der Programmstart erfolgt durch einen Doppelklick auf eclipse.exe im Eclipse-Ordner. Natürlich kann man sich dafür auch eine Verknüpfung auf der Oberfläche anlegen.

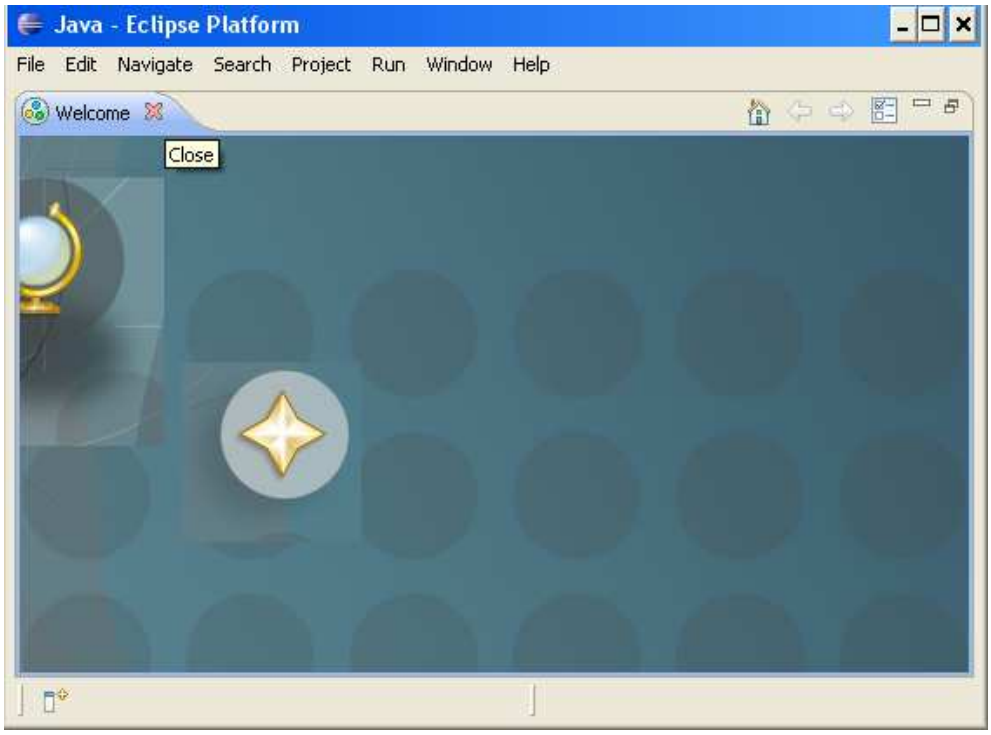


Wie für Java-Programme typisch, dauert der Programmstart etwas (zur Erinnerung: der Byte-Code muss zunächst in den Maschinencode der Maschine übersetzt werden, wird der Code dann erneut benötigt, liegt er bereits übersetzt vor, weshalb Java-Programme dann genau so schnell wie andere Programme sind).

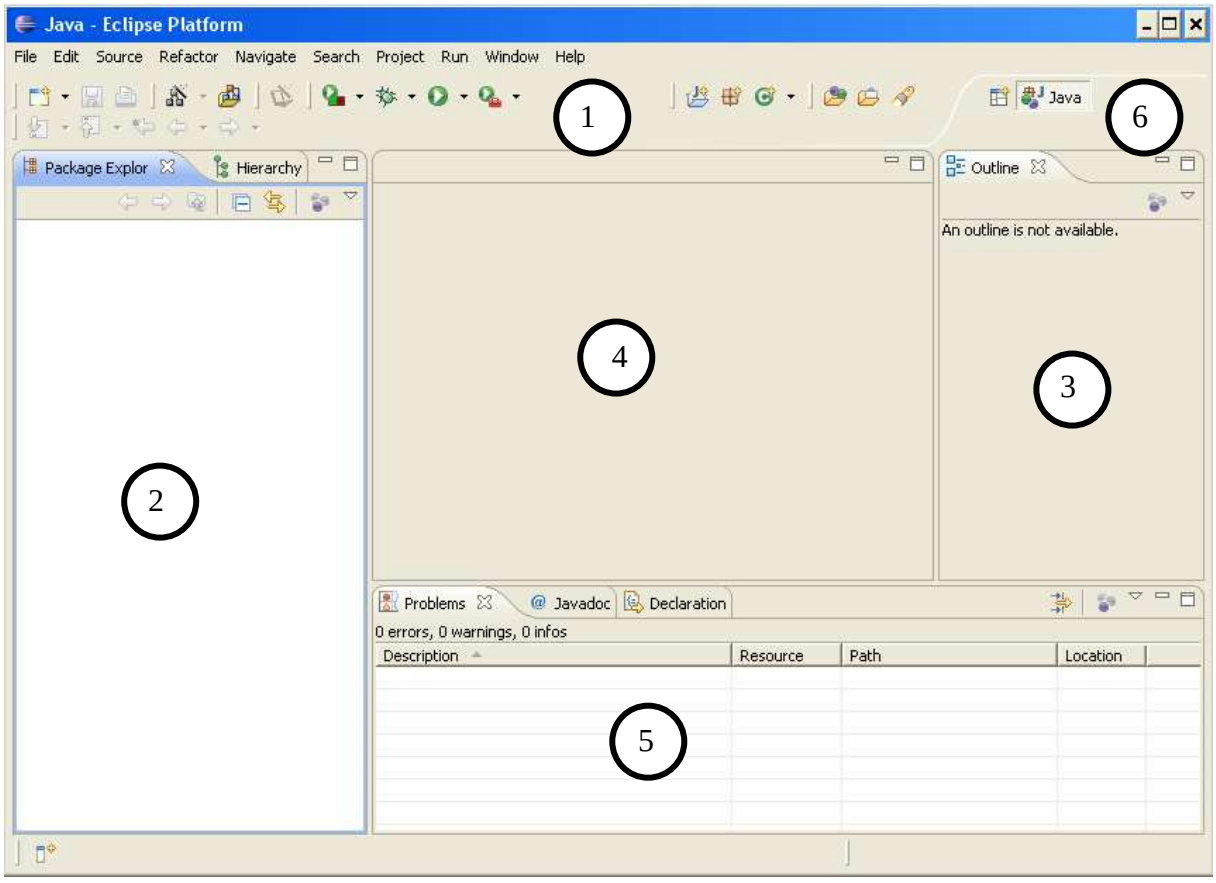
Der Nutzer wird dann aufgefordert, einen Workspace zu wählen. Dies ist der Ordner, in dem Eclipse alle Dateien und seine Verwaltungsinformationen speichert. Dieser Ordner sollte nie direkt bearbeitet werden, da Eclipse sonst instabil werden könnte. Es ist sinnvoll, zumindest für jede Vorlesung, in der Eclipse genutzt wird, einen eigenen Workspace anzulegen. Der Workspace kann sich auch auf einem USB-Stick befinden, sollte aber nie mit unterschiedlichen Betriebssystemen (UNIX/Windows) bearbeitet werden.

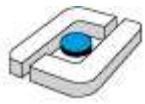


Danach erscheint der Welcome-Bildschirm, der zu einigen Informationen führt, die man aber später auch über die Hilfe absteuern kann und der deshalb jetzt geschlossen wird.



Danach öffnet sich die eigentliche Arbeitsfläche, die abhängig von der Eclipse-Version leicht anders aussehen kann.





Im vorherigen Fenster wurden rechts die Detailfenster „Task List“ und „Font and Colors“ und Links „UML 2.1 Modeler“, so sie vorhanden waren, bereits geschlossen. Die markierten Fensterbereiche haben folgende Bedeutung:

1: In der Werkzeugleiste sind alle vorhandenen Werkzeuge sichtbar. Die gesamte Funktionalität ist immer auch über das obige Menü erreichbar. Häufig kann man durch den Rechtsklick auf ein bestimmtes Element im Bereich 2 zu einem passenden Menü kommen.

2: Hier werden die von Eclipse verwalteten Projekte, also z. B. Java- und C++-Entwicklungen, mit ihren Dateien angezeigt. Die Anzeige erfolgt hierarchisch browserartig.

3: Hier findet eine Zusammenfassung der Informationen über den aktuell ausgewählten Projektbaustein statt. Dies kann z. B. eine Liste aller Variablen und Methoden einer Klasse sein.

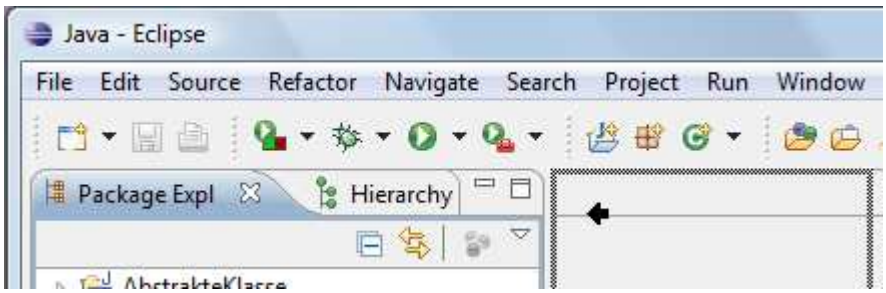
4: Hier findet die eigentliche Arbeit statt, es wird ein zur Aufgabe passender Editor, typischerweise mit Syntax-Highlighting, geöffnet.

5: Verteilt auf die verschiedenen Reiter werden hier typischerweise Ausgaben zu aktuellen Vorgängen angezeigt. Dies können z. B. Meldungen des Kompilierungsprozesses, Listen mit Syntaxfehlern oder Eingabeaufforderungen von Nutzungsdialogen sein.

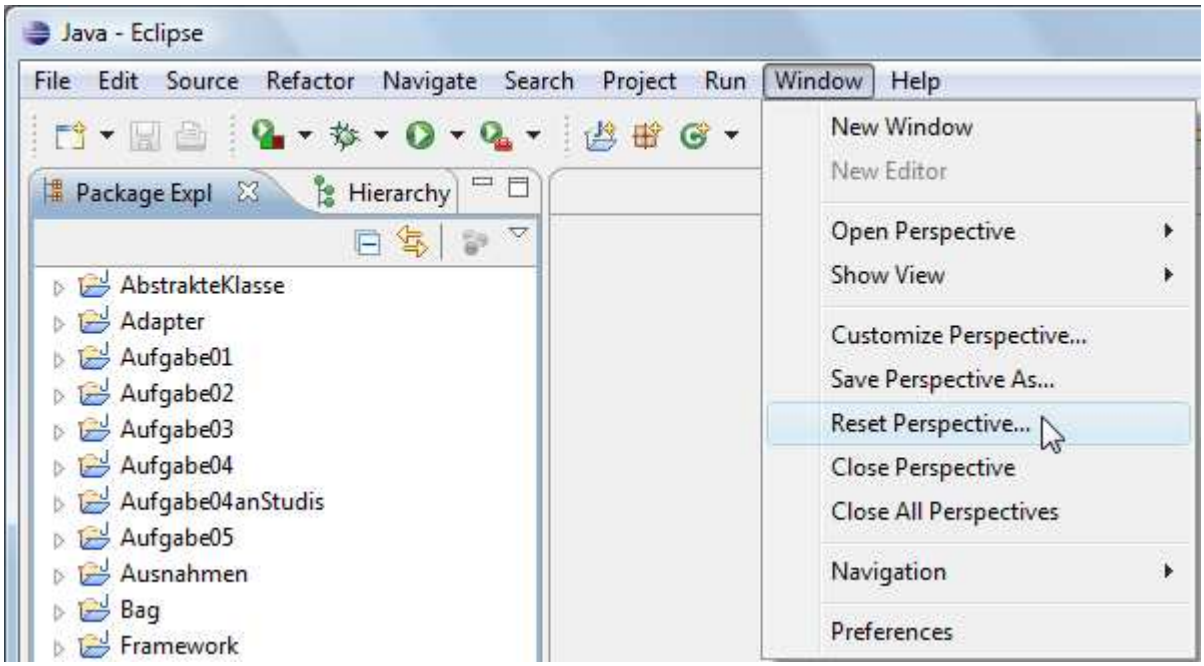
6: Hier wird die aktuell von Eclipse genutzte Sicht angezeigt. Jede Sicht beinhaltet spezielle Fenster und kann eine etwas andere Darstellung des Projektinhalts bieten.

Man beachte, dass die Fenster frei verschiebbar und an verschiedenen Stellen andockbar sind. Zum Verschieben wird auf die Mitte eines Reiters mit der linken Maustaste gedrückt und dies dann festgehalten.

Das Fenster kann dann beliebig mit gedrückter Maustaste verschoben werden. Pfeile zeigen an, wo das Fenster dann angezeigt werden würde.



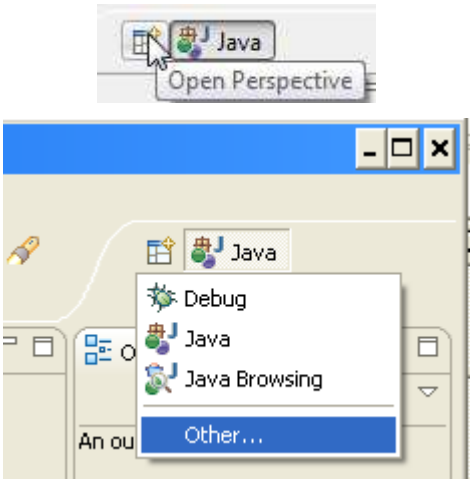
Man sollte auf ein zu starkes konfigurieren verzichten, da die Oberfläche dann für andere Personen und eventuell für einen selbst unleserlich wird. Falls man sich „verbastelt“ haben sollte kann mit dem Punkt „Window -> Reset Perspective ...“ den Ursprungszustand wieder herstellen.


Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Expl
Hierarchy
AbstrakteKlasse
Adapter
Aufgabe01
Aufgabe02
Aufgabe03
Aufgabe04
Aufgabe04anStudis
Aufgabe05
Ausnahmen
Bag
Framework
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Navigation
Preferences

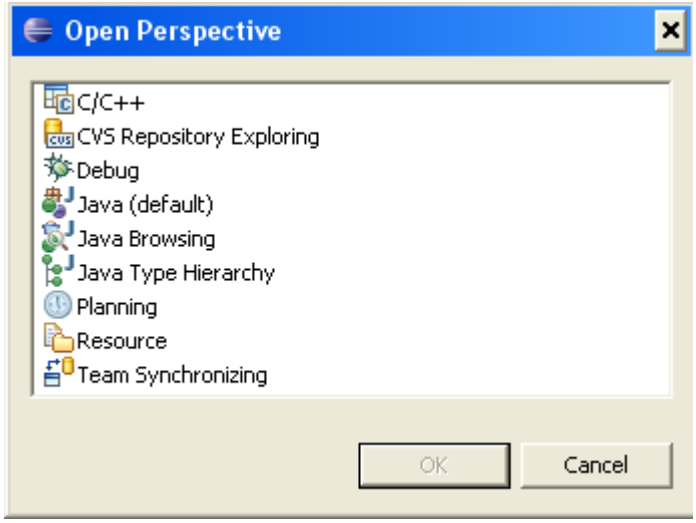

Reset Perspective
Do you want to reset the current Java perspective to its defaults?
OK
Cancel

4 Einstellungen und Spielereien in Eclipse

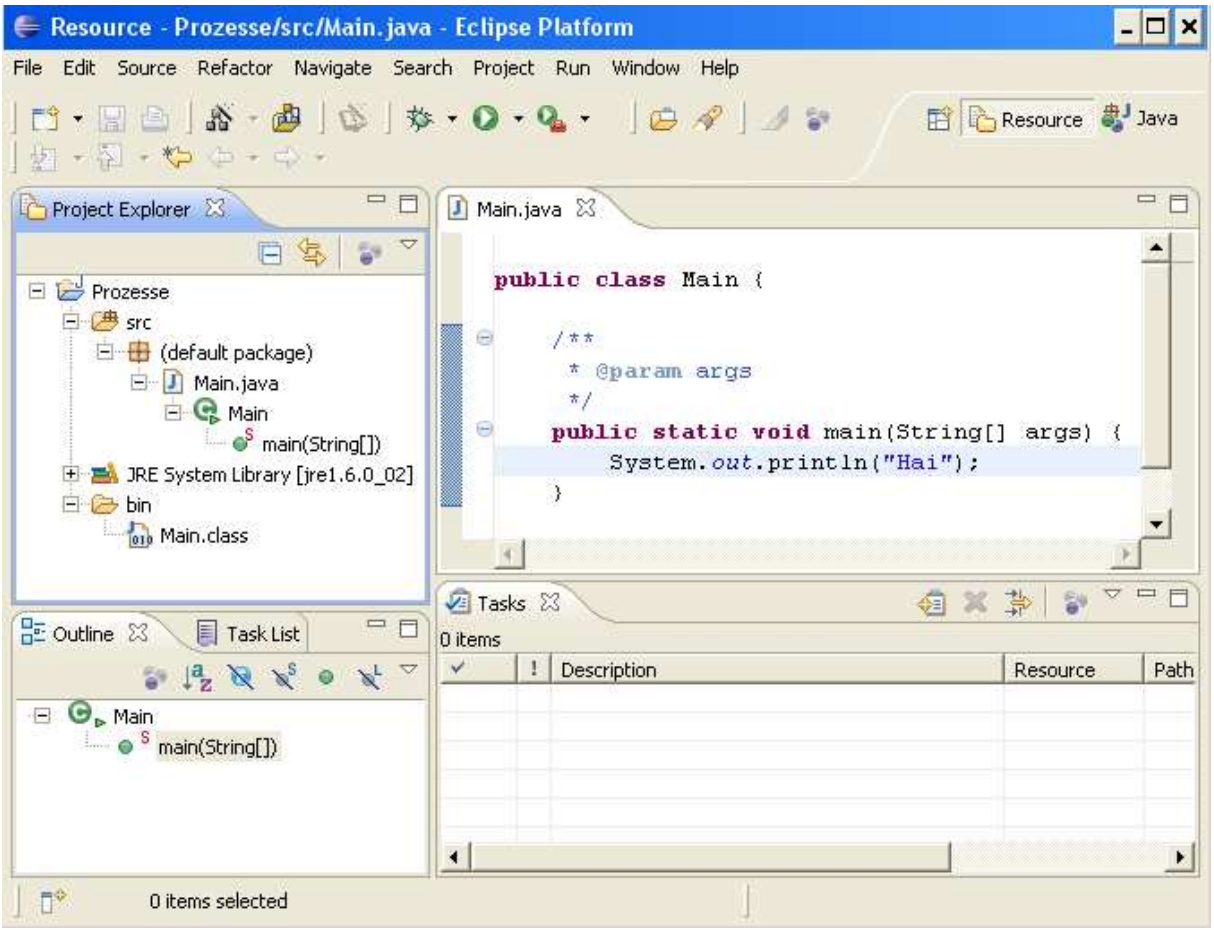
Im vorherigen Abschnitt wurden die verschiedenen Sichten angesprochen, diese können rechts oben eingestellt werden. Dazu wird das Symbol links neben dem Sichtnamen Java angeklickt.



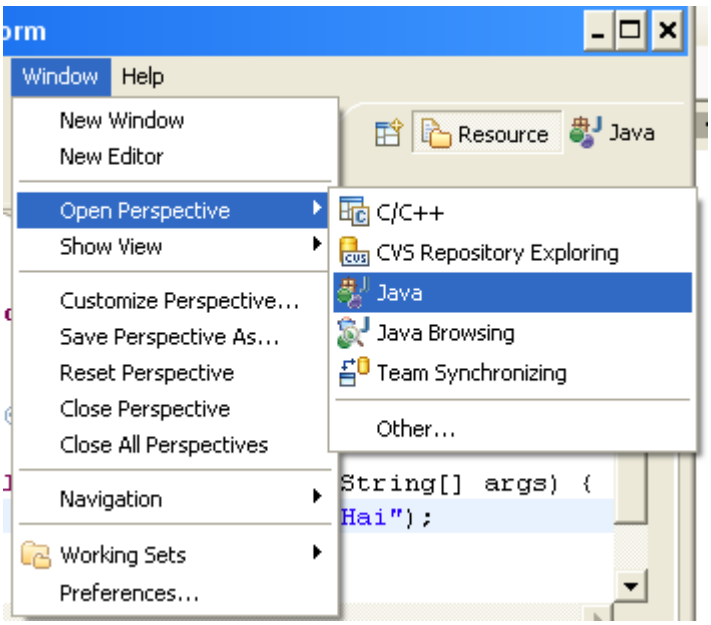
Unter „Other“ erhält man eine vollständige Übersicht über die vorhandenen Sichten. Diese Übersicht hängt von den vorhandenen Plugins ab und kann wie folgt aussehen.



Interessant ist u. a. die Ressource-Sicht, da man damit einen Datei-Browser erhält, der alle Dateien so anzeigt, wie sie auf der Platte angeordnet sind. Dies kann für ein Java-Projekt z. B. wie folgt aussehen, man erkennt das sonst nicht sichtbare bin-Verzeichnis.

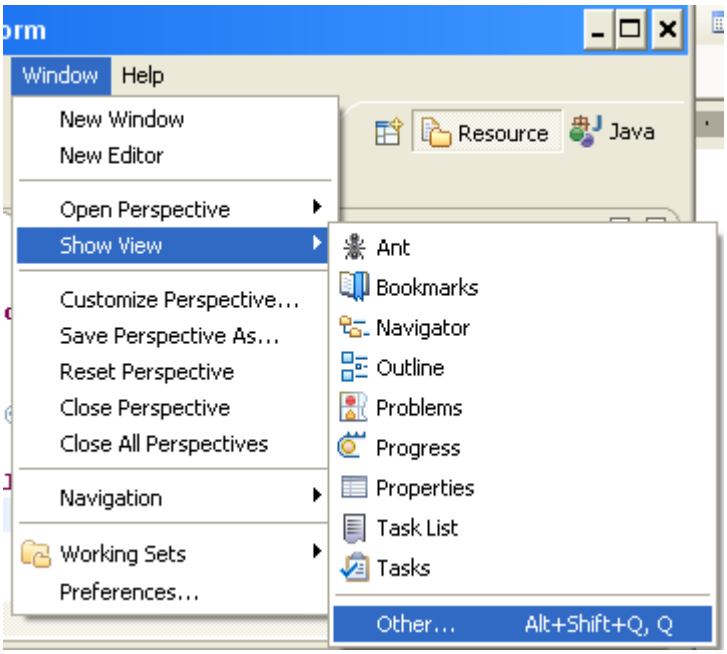


Die Sicht kann auch unter „Window->Open Perspective“ eingestellt werden.

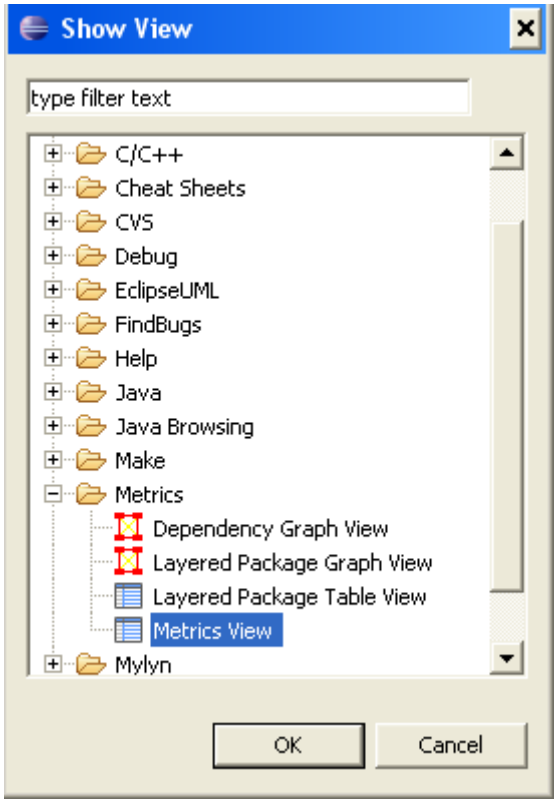


Das obige Menü bietet ebenfalls den Punkt „Reset Perspective“, mit man alle in einer Sicht vorgenommenen Änderungen, z. B. aus Versehen geschlossene Fenster, wieder rückgängig machen kann. Weiterhin besteht die Möglichkeit, individuell eingestellte Sichten zu speichern, so dass auch diese immer wieder hergestellt werden können.

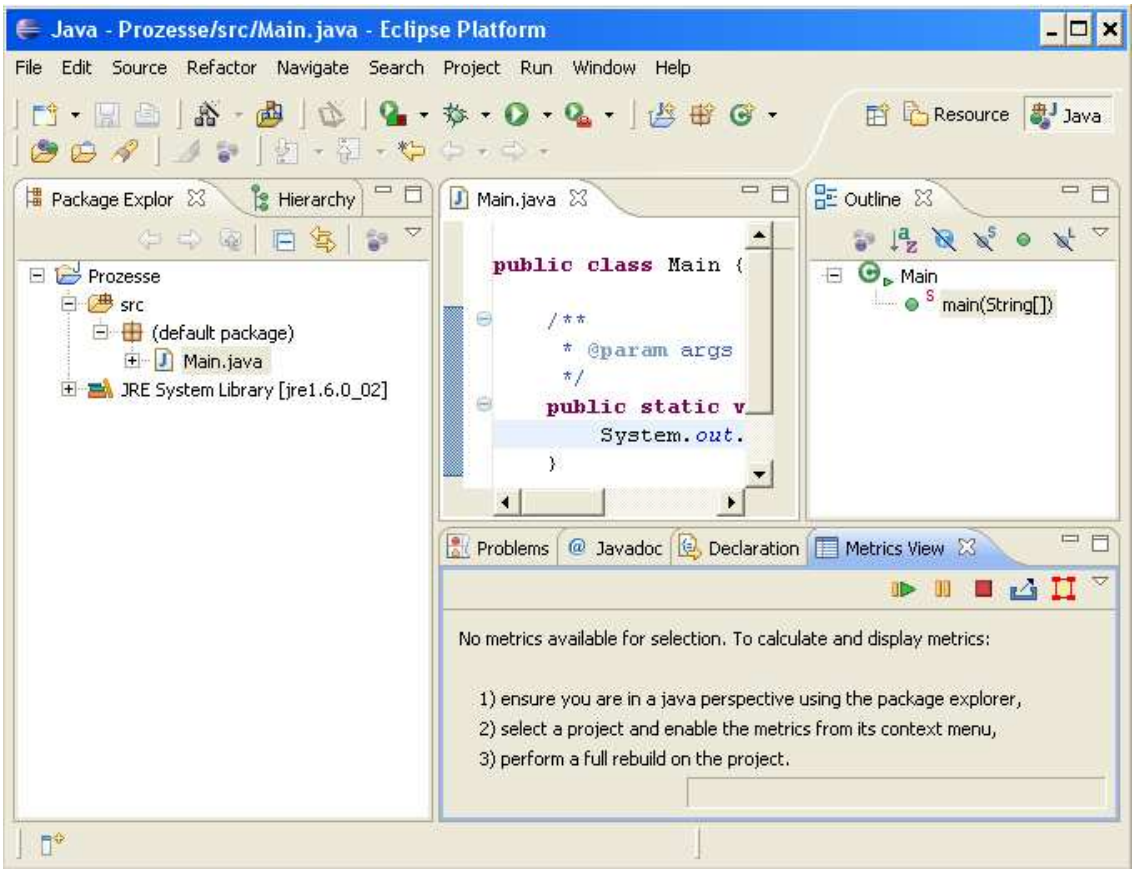
Häufig bieten Plugins zusätzliche Fenster. Diese können unter „Show View“ in der vorherigen Sicht ausgewählt werden.



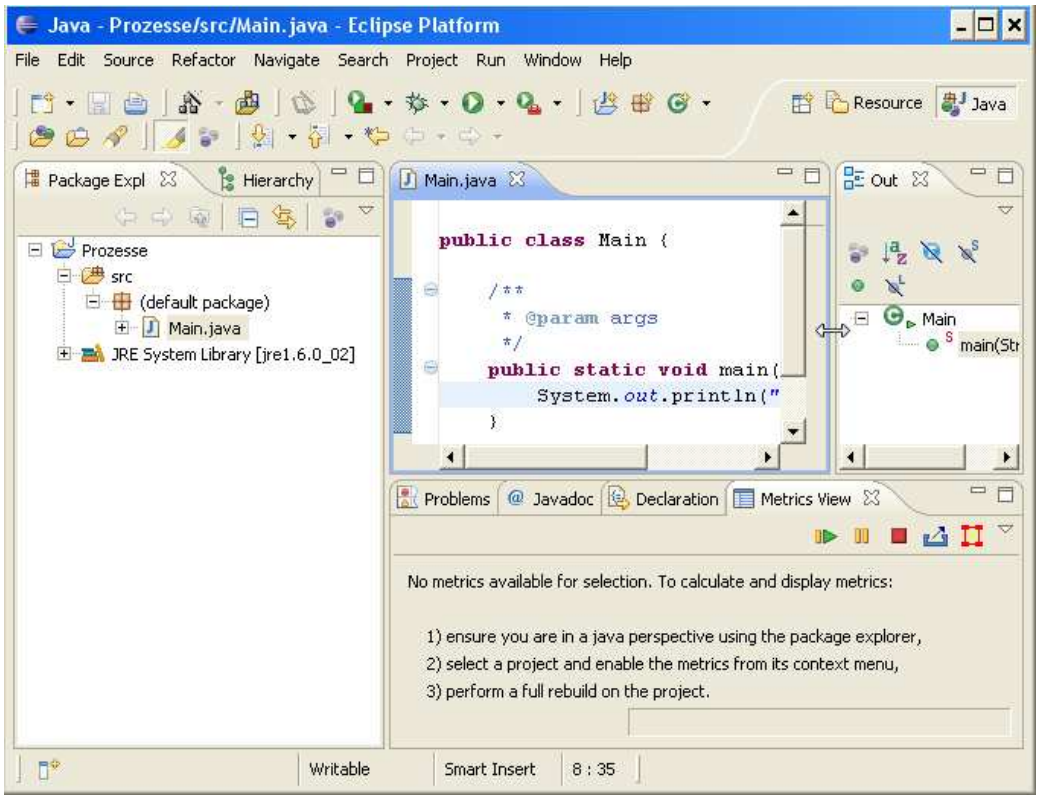
Wählt man z. B. wie im folgenden Bild gezeigt unter „Other“ den „Metrics View“

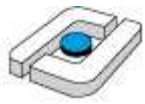


dann wird ein zusätzlicher Reiter mit diesem View angezeigt, wie es unten zu erkennen ist.

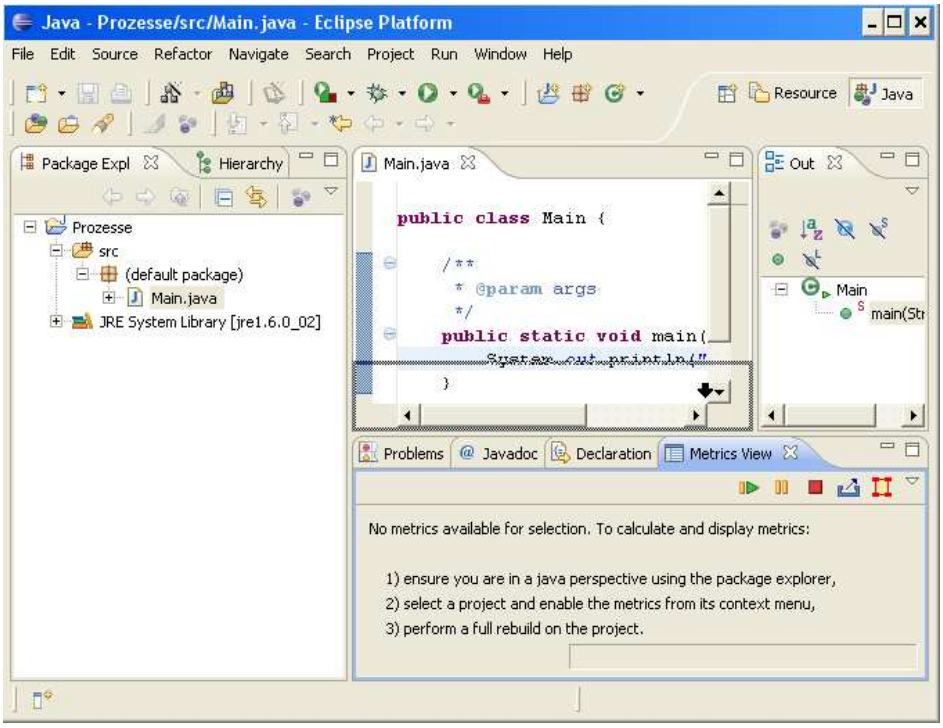


Die Fenstergrößen kann man grundsätzlich durch das Verschieben der Verbindungslinien ändern, wie man im folgenden Bild rechts in der Mitte mit dem Doppelpfeil sieht.

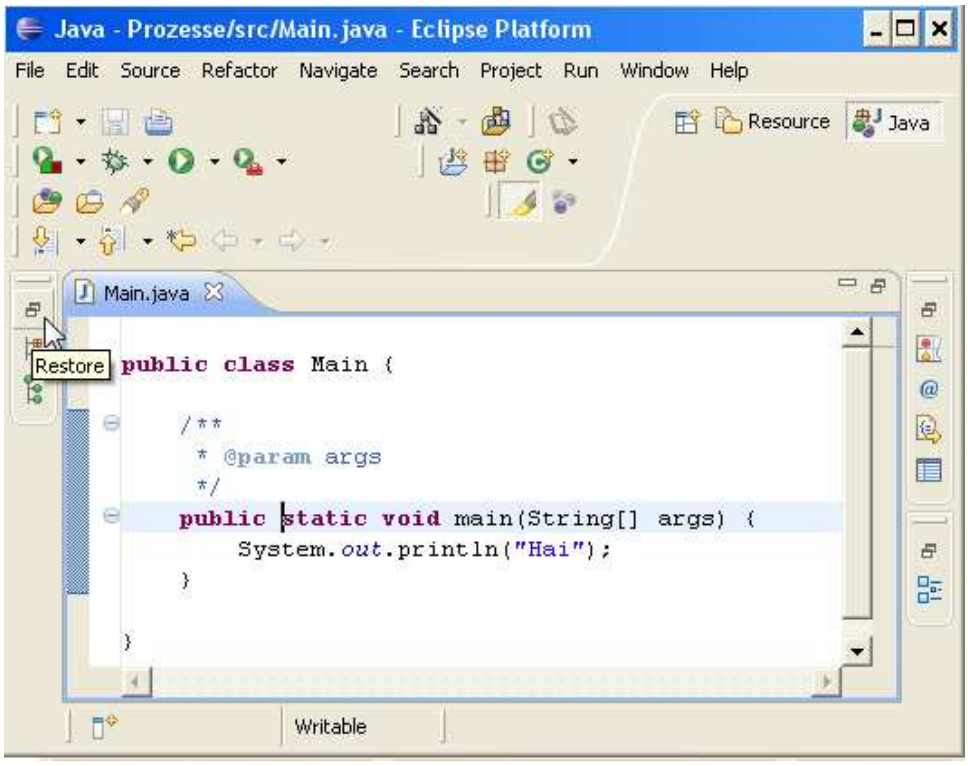




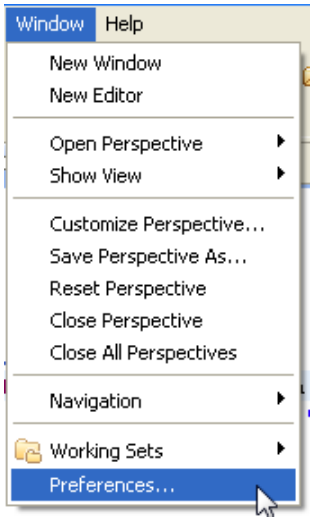
Klickt man mit der linken Maustaste in der Mitte auf den Namen eines Reiters, kann dieser im Fenster verschoben werden und einen neuen Platz einnehmen. Damit ist der Reiter verschiebbar oder es wird ein zusätzliches Detailfenster, wie in der Mitte des folgenden Bildes mit den Rahmenlinien angedeutet, geöffnet.



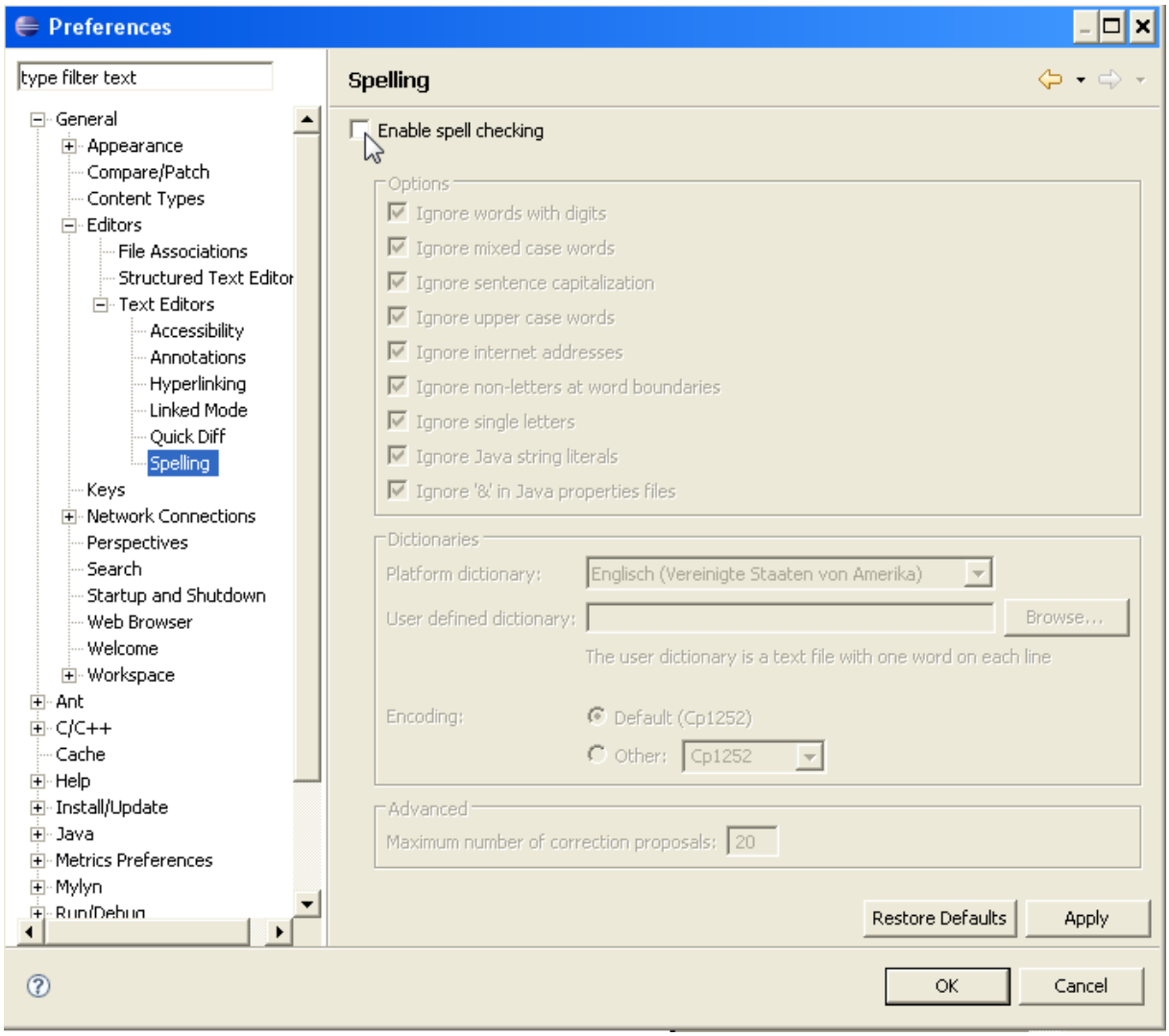
Durch einen Doppelklick auf einen Reiter nimmt dieser, wie folgt gezeigt, die gesamte Eclipse-Oberfläche ein. Durch einen erneuten Doppelklick oder einen Klick auf Restore links oben am Rand, kehrt man zur normalen Fensteranordnung zurück.



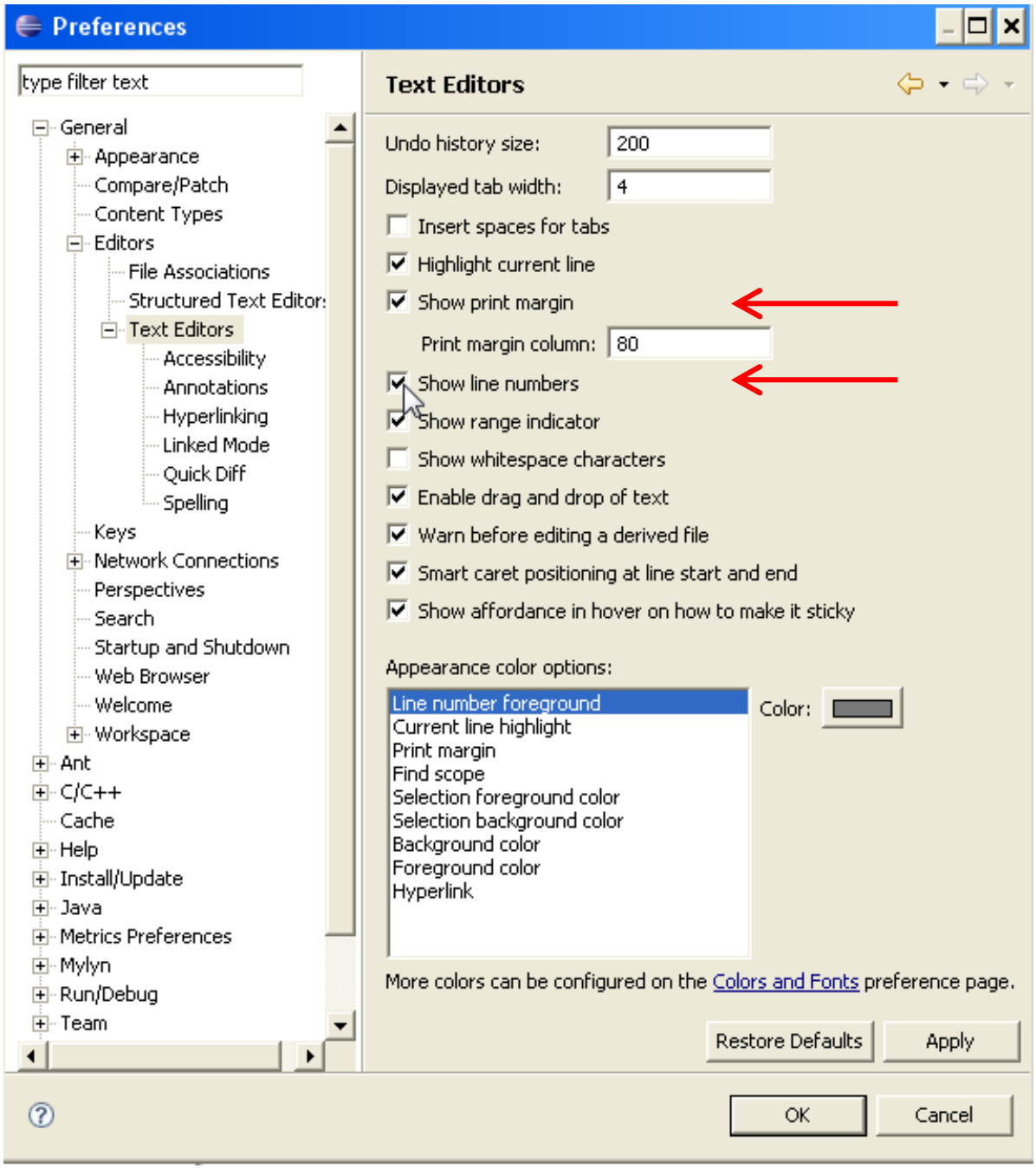
Detaileinstellungen von Eclipse werden unter „Window -> Preferences“ vorgenommen.



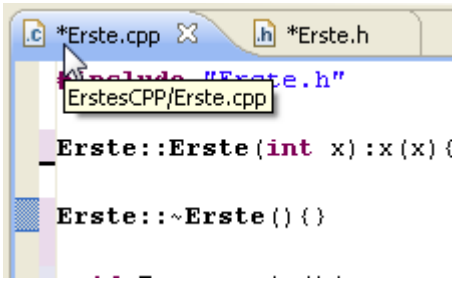
Die Einstellungsmöglichkeiten sind sehr sehr detailliert und hängen von den installierten Plugins ab. Eclipse hat ab 3.3 eine Rechtschreibprüfung integriert, die meist nur für Englisch vorliegt. Unter folgendem Punkt kann diese Prüfung abgeschaltet oder verändert werden.



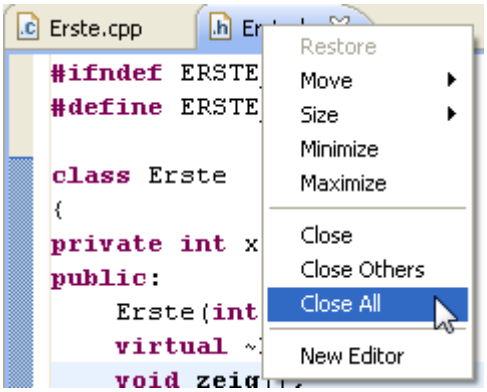
Das folgende Bild zeigt die Möglichkeit, u. a. Zeilennummern und einen Rand für den Druckbereich einzustellen.



Wenn Dateien geändert werden, erkennt man an einem kleinen Stern rechts neben dem Namen, dass diese Datei verändert und noch nicht gespeichert wurde.



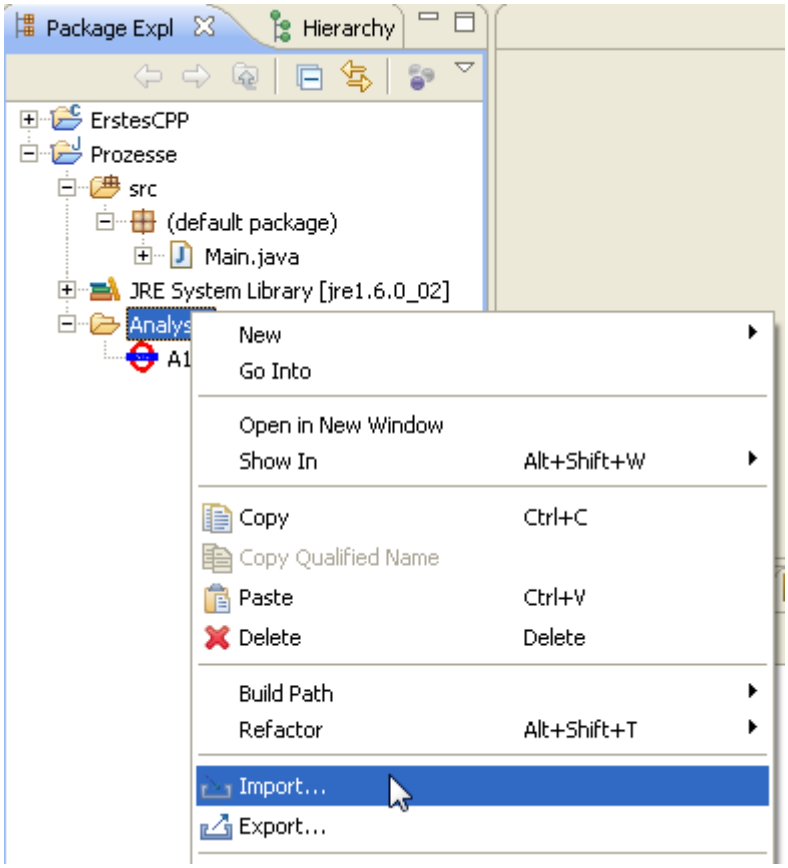
Will man mehrere Dateien auf einmal schließen, so kann man einen Rechtsklick auf einen geöffneten Reiter einer beliebigen Datei machen und erhält den folgenden Auswahldialog.



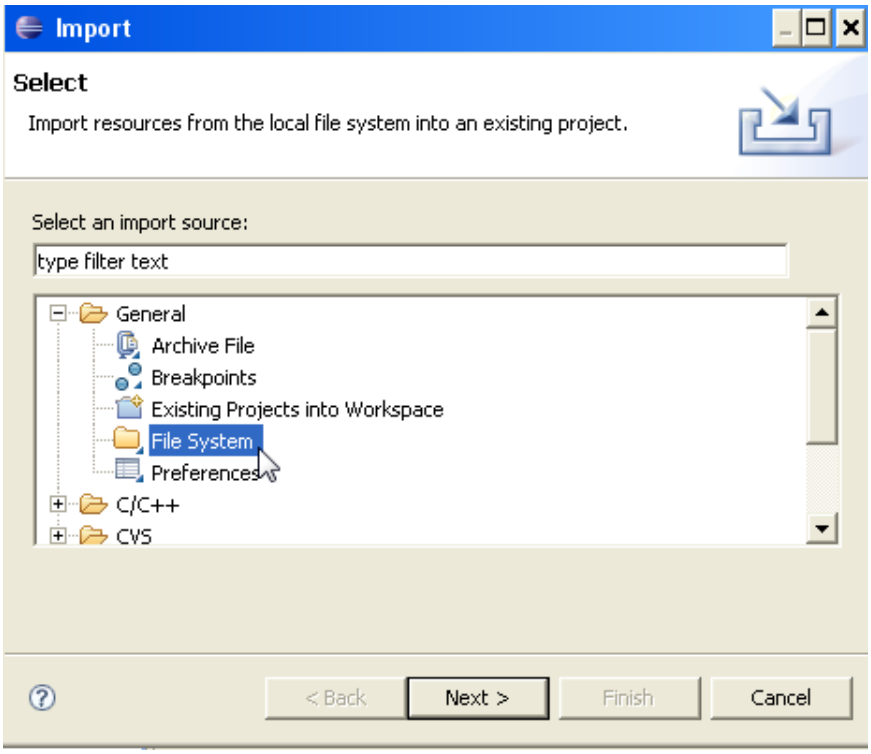
5 Dateien importieren

Es gibt verschiedene Varianten, was für Dateien man in ein Eclipse-Projekt laden kann. Die einfachste Möglichkeit besteht darin, dass eine Datei direkt genutzt werden soll. Die zweite Variante ist, dass zu verändernde Dateien in einer zip- oder jar-Datei vorliegen. Die Einbindung einer jar-Datei zur ausschließlichen Nutzung wird bei der Java-Entwicklung beschrieben.

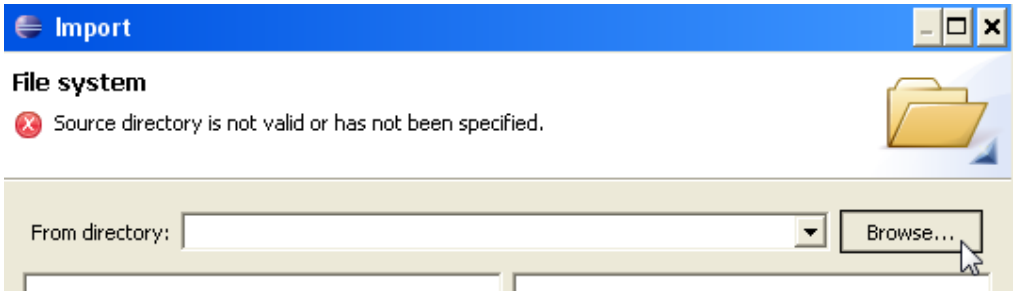
Zunächst soll eine einfache Datei in ein Eclipse-Projekt kopiert werden. Gestartet wird z. B. mit einem Rechtsklick auf den Ordner in den die Datei geladen werden soll und anschließendem Klick auf „Import..“.

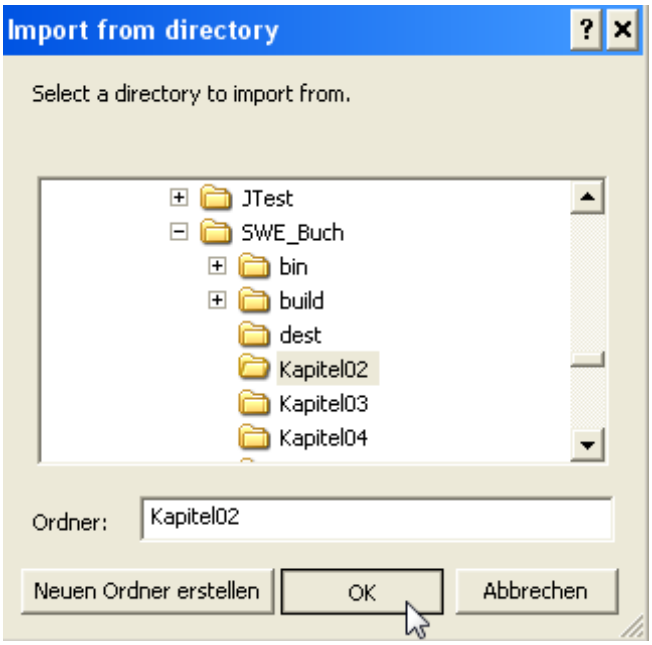


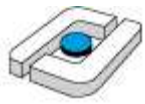
Aus den angezeigten Möglichkeiten wird dann „File System“ ausgewählt.



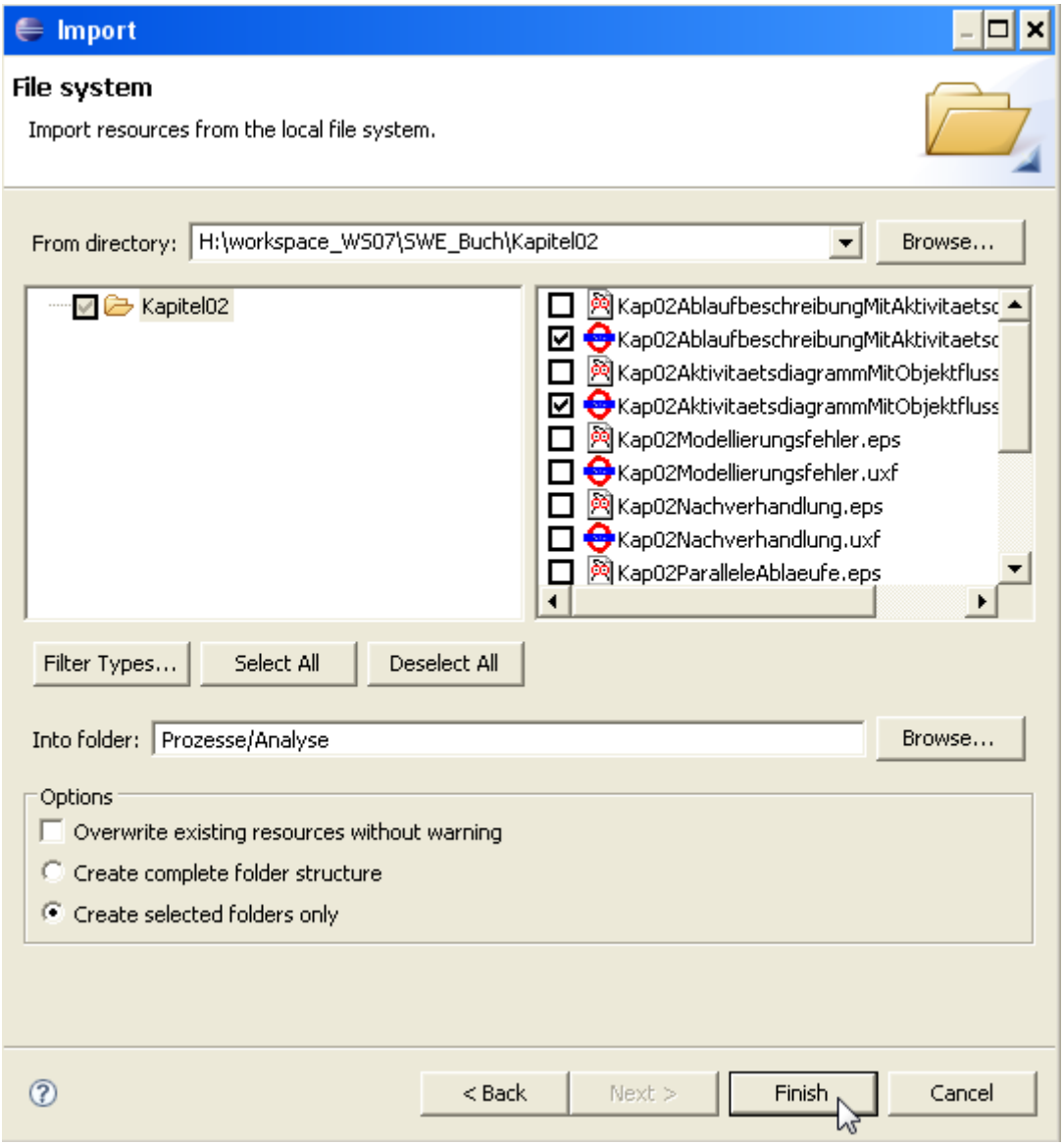
Danach kann man unter „Browse“ zu dem Verzeichnis manövrieren, aus dem man eine Datei oder mehrere Dateien einladen möchte.





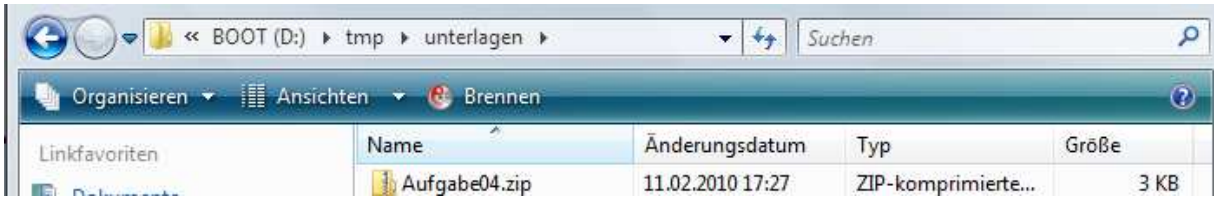


In dem sich dann öffnenden Dialog kann man auf der linken Seite in verschiedene Unterverzeichnisse manövrieren und dann auf der rechten Seite die gewünschten Dateien auswählen. Man beachte noch die meist nicht zu ändernden Einstellungen unten im Dialog.

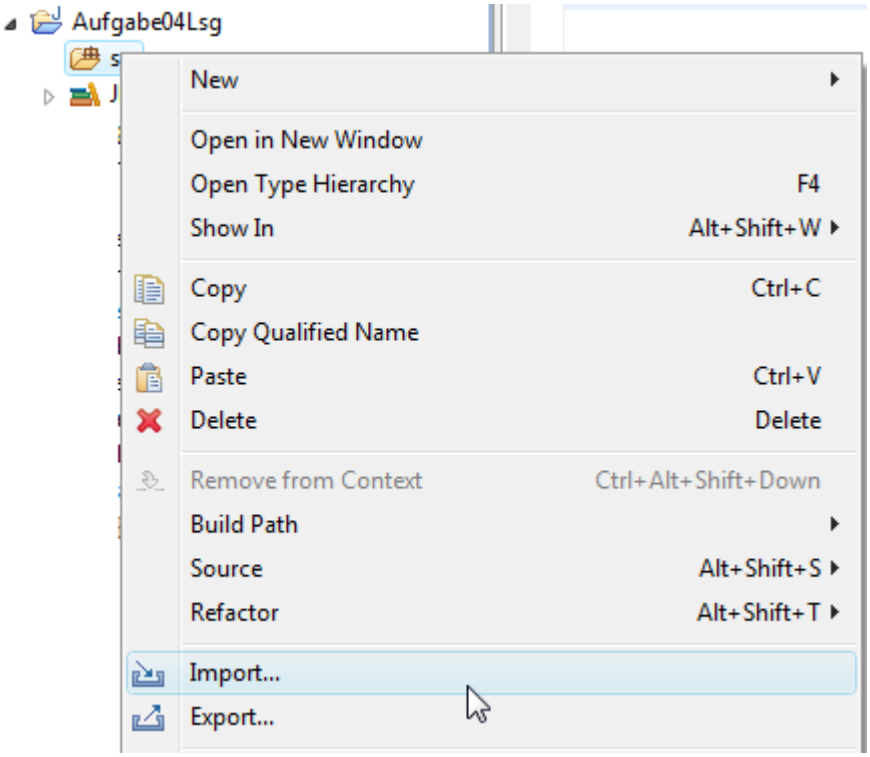


Nach einem Klick auf „Finish“ werden die Dateien dann in das Eclipse-Projekt kopiert.

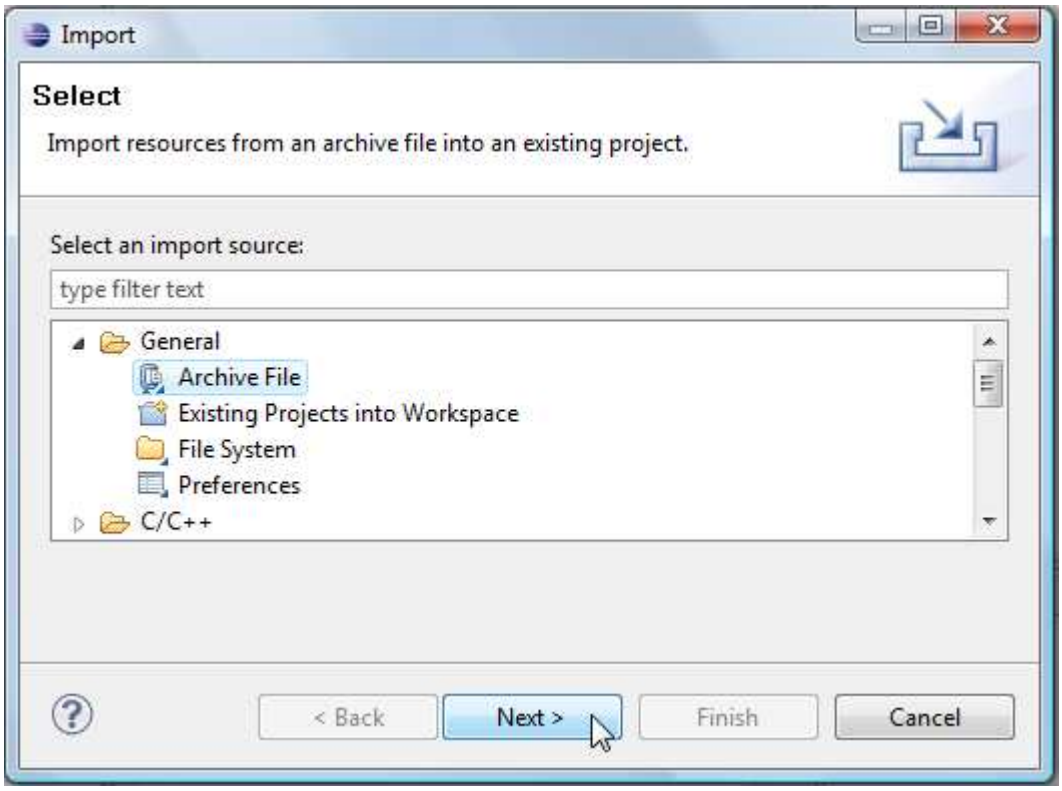
Möchte man Dateien aus einer zip- oder jar-Datei in das Projekt zur Bearbeitung kopieren, ist der Ablauf sehr ähnlich. Es wird angenommen, dass die Dateien in einer Zip-Datei Aufgabe04.zip.



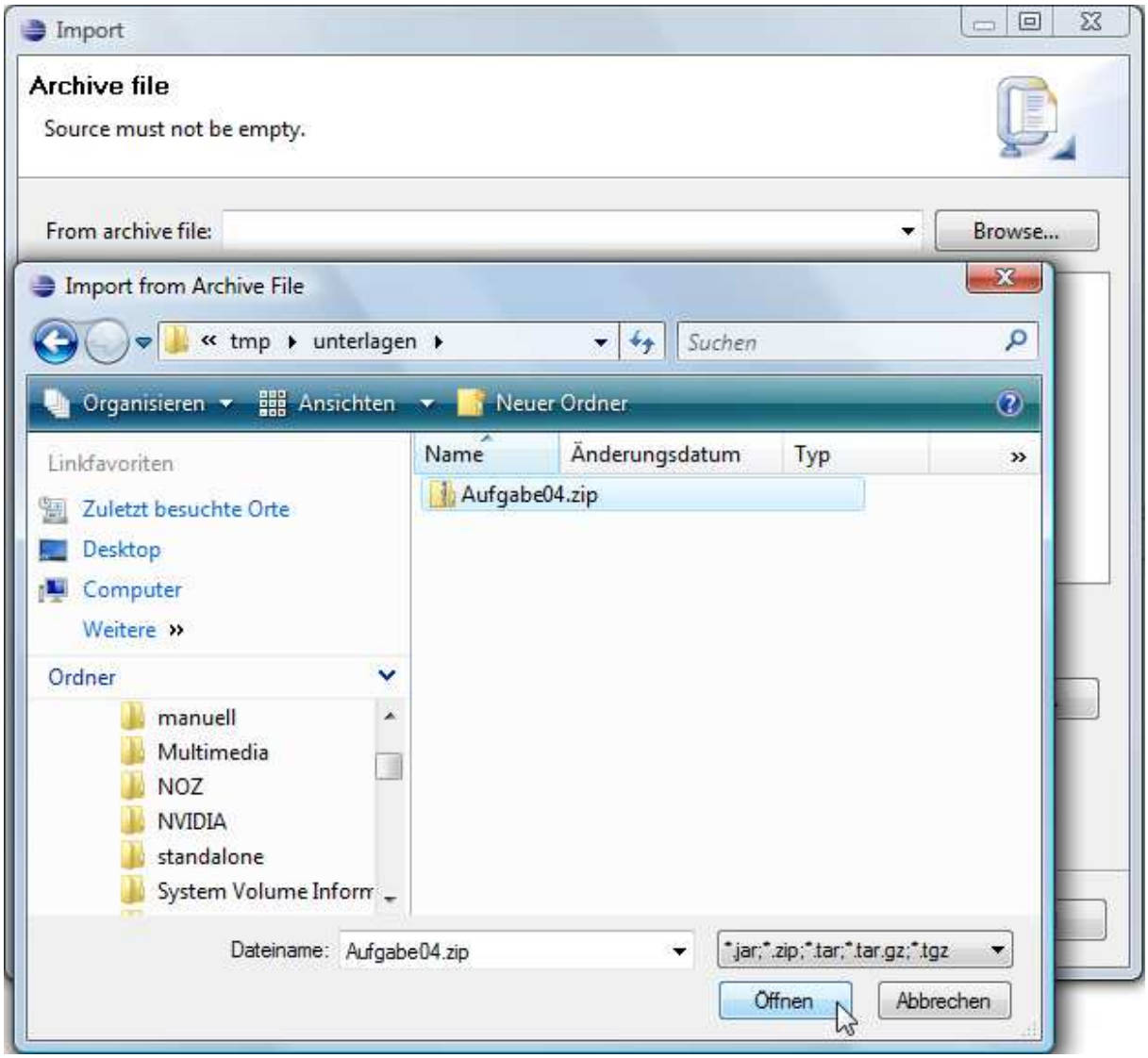
Gegeben Sei ein Projekt Aufgabe04Lsg, dann wird ein Rechtsklick auf dem Ordner src ausgeführt und „Import“ gewählt.



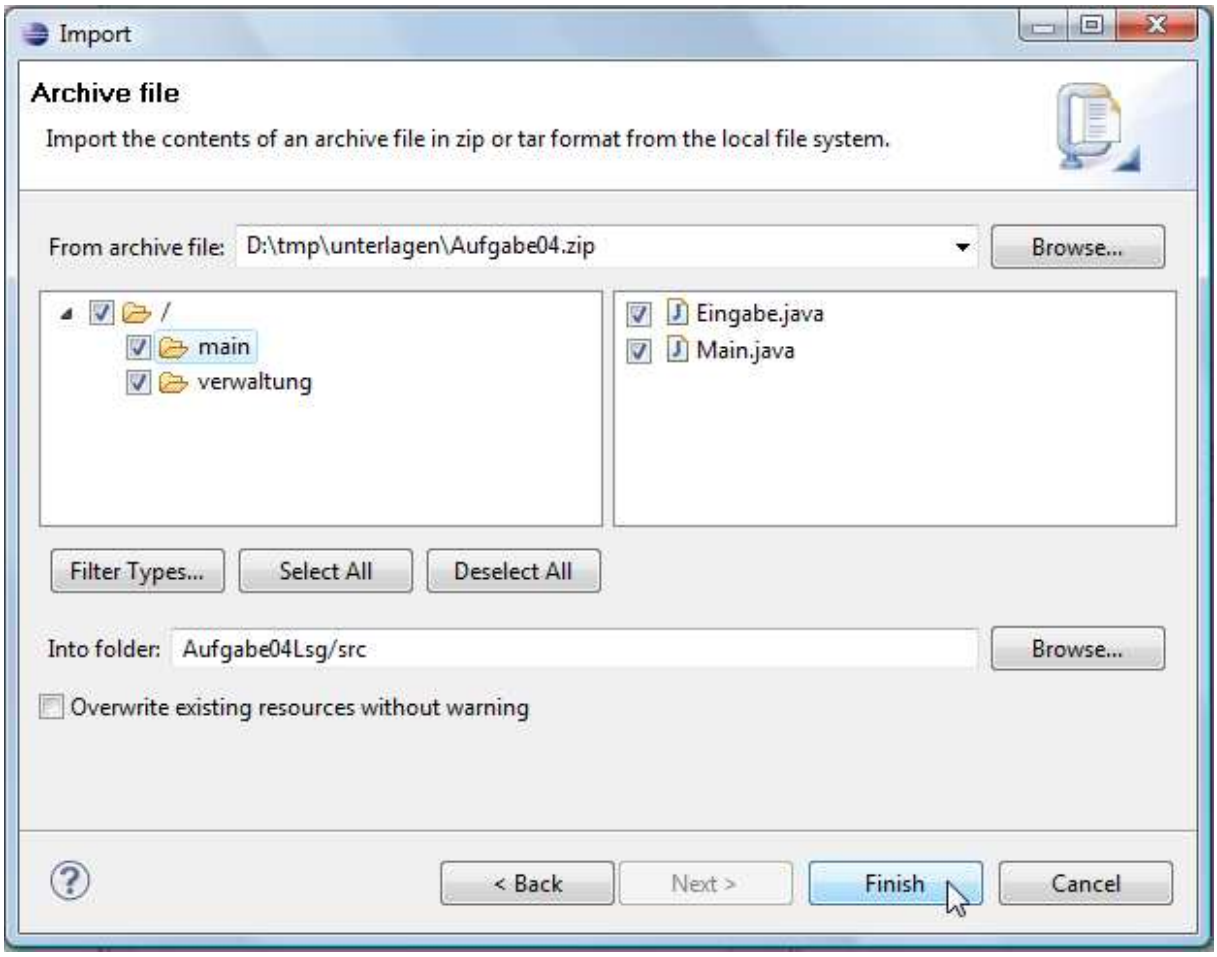
Jetzt wählt man unter „General“ dann „Archive File“ und dann „Next>“



Nun muss unter „Browse“ rechts oben zum Verzeichnis mit der gewünschten zip-Datei manövriert werden.

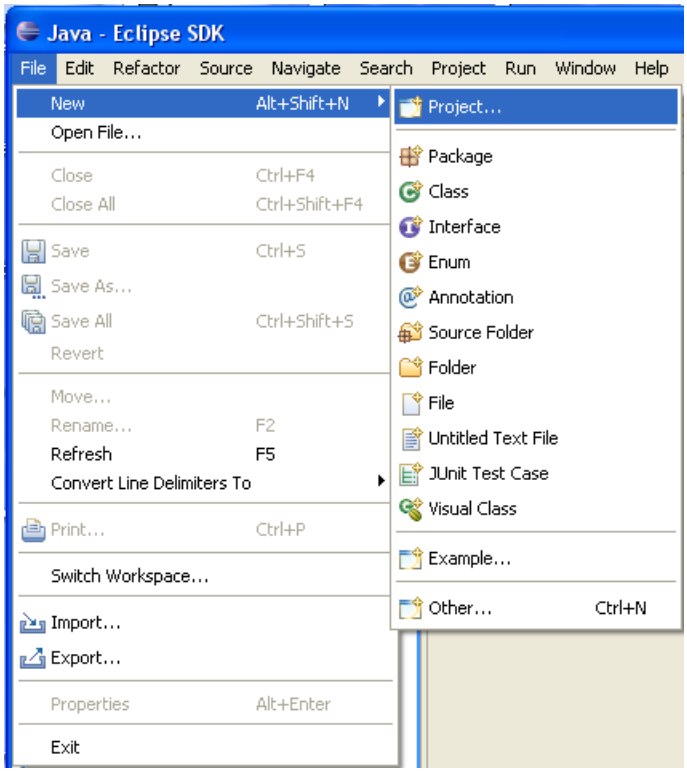


Nun werden die gewünschten Ordner und Dateien markiert und mit „Finish“ in das Projekt kopiert, so dass die Dateien bearbeitet werden können. Man beachte den Ordner unter „Into Folder:“, der zum Archiv passen muss. Als Problem könnte z. B. im Archiv sich noch der Ordner src auf oberster Ebene befinden, so dass nach dem Import eine Ordnerstruktur src/src entsteht. Falls man das Problem übersieht, kann man in Eclipse aber einfach Dateien per Drag&Drop verschieben.

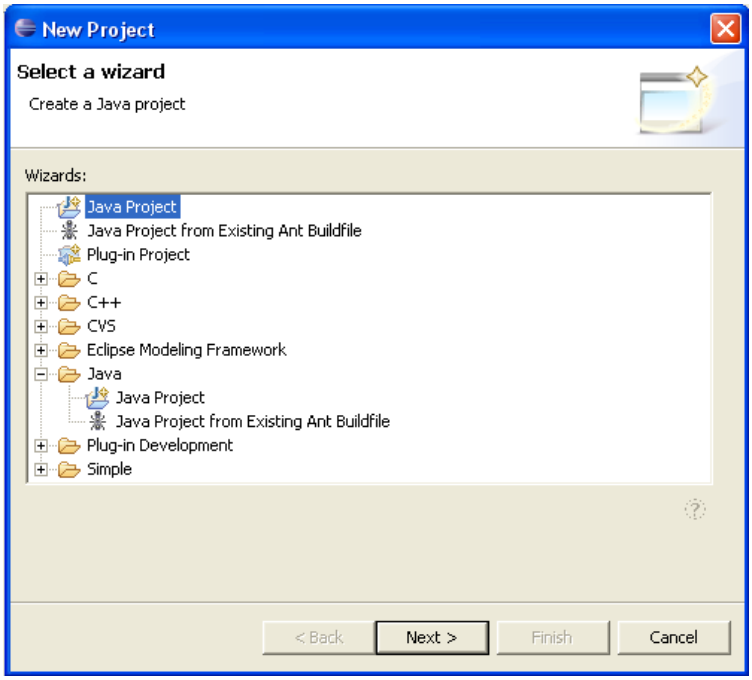

Import
Archive file
Import the contents of an archive file in zip or tar format from the local file system.
From archive file: D:\tmp\unterlagen\Aufgabe04.zip
Browse...
/
main
verwaltung
Eingabe.java
Main.java
Filter Types...
Select All
Deselect All
Into folder: Aufgabe04Lsg/src
Browse...
Overwrite existing resources without warning
< Back
Next >
Finish
Cancel

6 Anlegen eines Java-Projekts

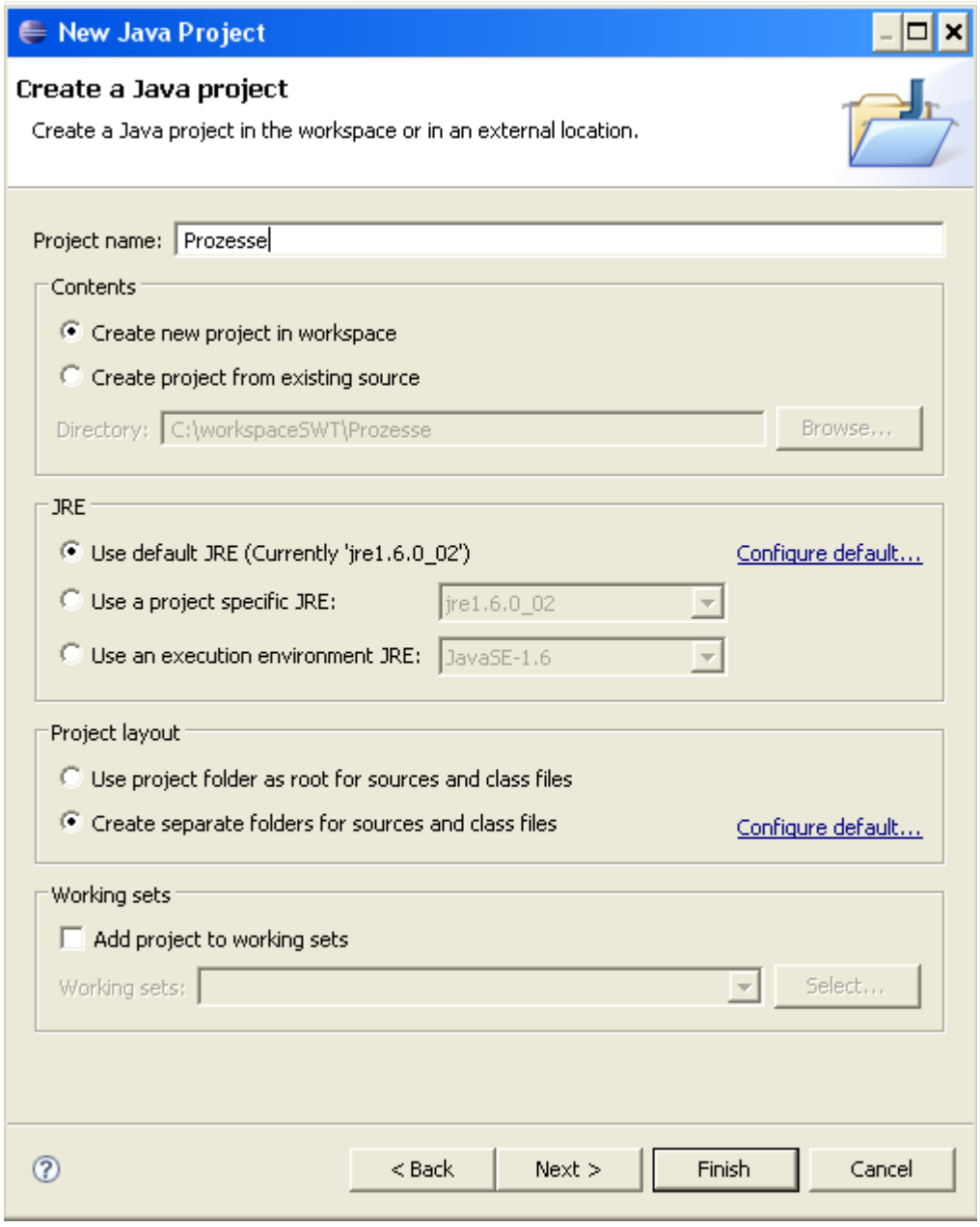
Zunächst wird ein neues Projekt angewählt, wobei der obere Teil dieses Menüs sich den zuletzt getroffenen Auswahlen anpasst.



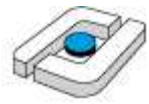
Dann wird „Java Project“ angewählt. Dieser Punkt kann mit gleicher Bedeutung mehrfach im Auswahlbaum erscheinen.



Im folgenden Fenster wird oben der Projektname eingetragen. Die weiteren Einstellungen sollten so übernommen werden.



Danach kann die Projekt-Einrichtung für das neue Java-Projekt unter „Finish“ abgeschlossen werden.

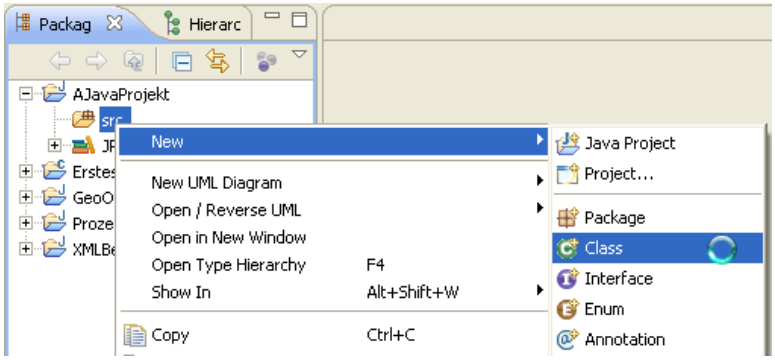


7 Java-Entwicklung mit Eclipse

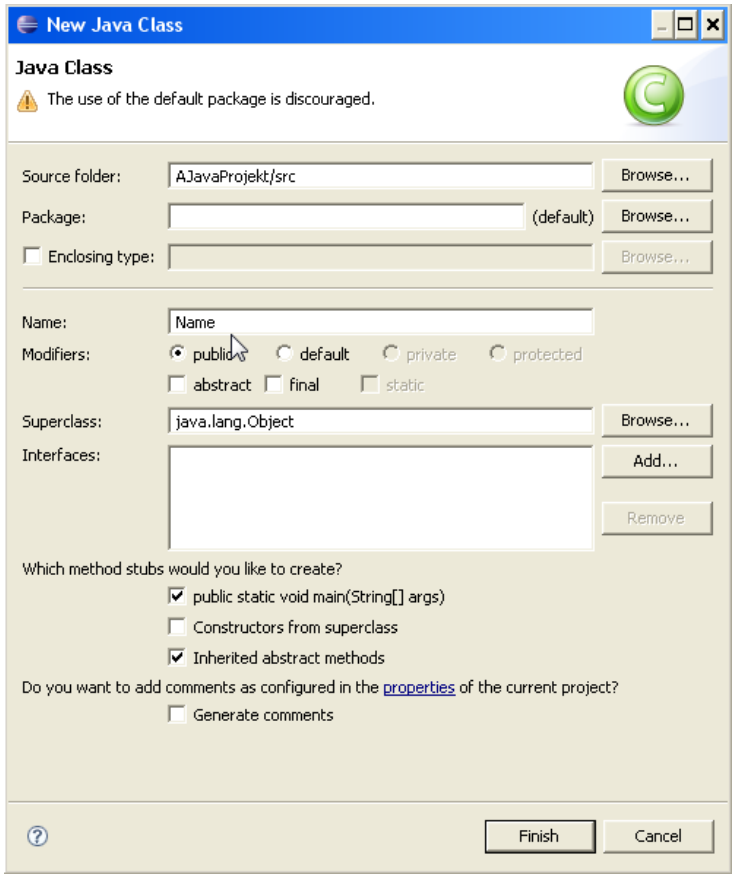
Es wird davon ausgegangen, dass ein Java-Projekt angelegt wurde und man sich in der Java-Sicht befindet (in den vorherigen Kapiteln erklärt).

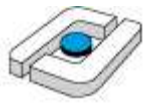
7.1 Anlegen einer Klasse

Zum Anlegen einer neuen Klasse wird z. B. ein Rechtsklick auf einen Source-Ordner, d. h. src oder ein Paket, gemacht und „New > Class>“ ausgewählt.

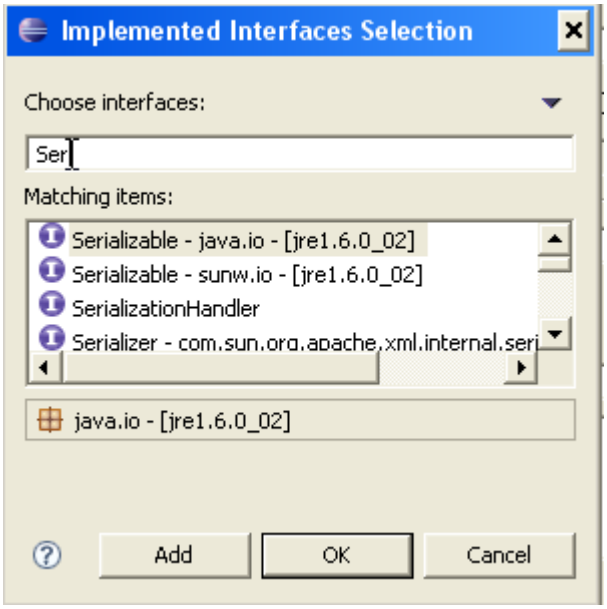


Im aufgehenden Fenster hat man bereits einige Einstellungsmöglichkeiten. Minimal reicht es aus, den Klassennamen im Feld „Name“ einzutragen. Weiterhin kann oben der Paketname nachträglich eingegeben werden. Unter Superclass steht, von welcher Klasse die neue Klasse erbt.





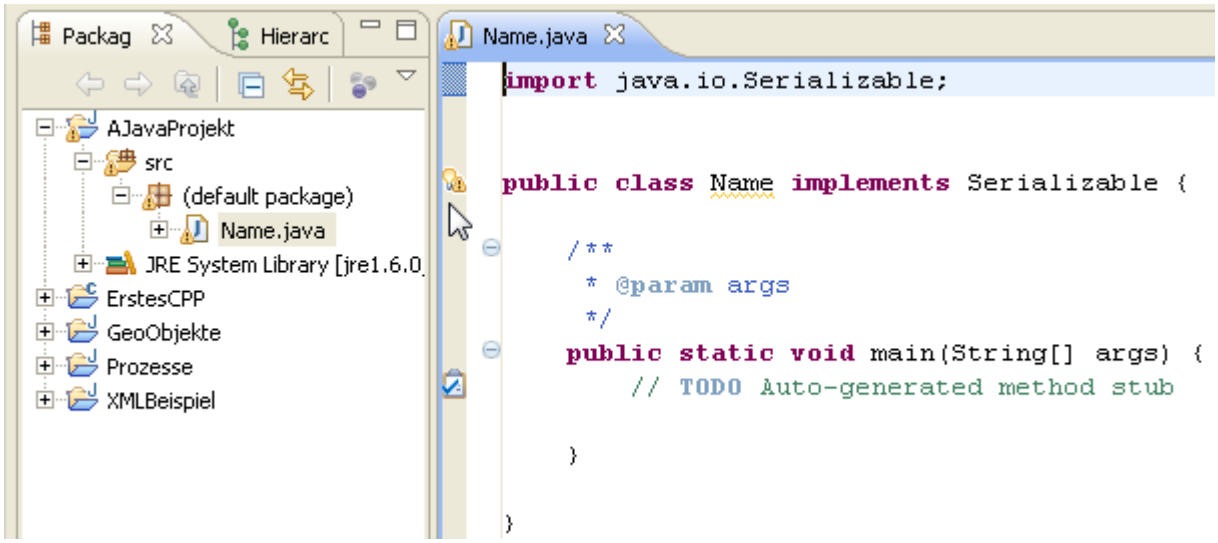
Weiterhin kann unter Interfaces angegeben werden, welche Interfaces (Schnittstellen) von der neuen Klasse implementiert werden. Die konkrete Auswahl findet über den Button am rechten Rand „Add...“ statt. Im sich öffnenden Fenster kann man dann oben den Namen des Interfaces eingeben, wobei unten eine Auswahl von Interfaces mit passenden Namen steht, die sich mit der Eingabe der ersten Zeichen des Namens anpasst. Im Beispiel wird Serializable genutzt.

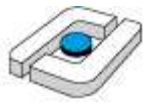


Kehrt man dann mit „OK“ zur Klassenerstellung zurück, kann weiterhin noch ausgewählt werden, ob die main()-Methode zum Programmstart in dieser Klasse generiert wird. Die Klassenerstellung wird mit „Finish“ abgeschlossen.

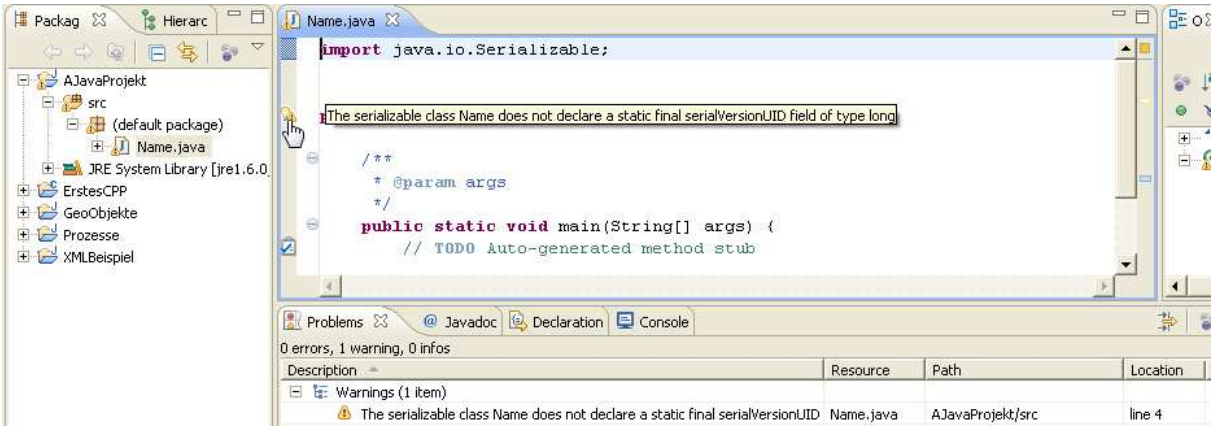
7.2 Automatische Fehlerkorrektur

Im folgenden Beispiel sieht man die generierte Klasse, die auch zeigt, wo der Entwickler noch Kommentare ergänzen, bzw. was er noch machen muss (TODO). Am linken Rand des Editorfensters erkennt man eine Glühbirne mit einem Ausrufungszeichen, wodurch auf ein Problem, aber keinen echten Fehler hingewiesen wird.

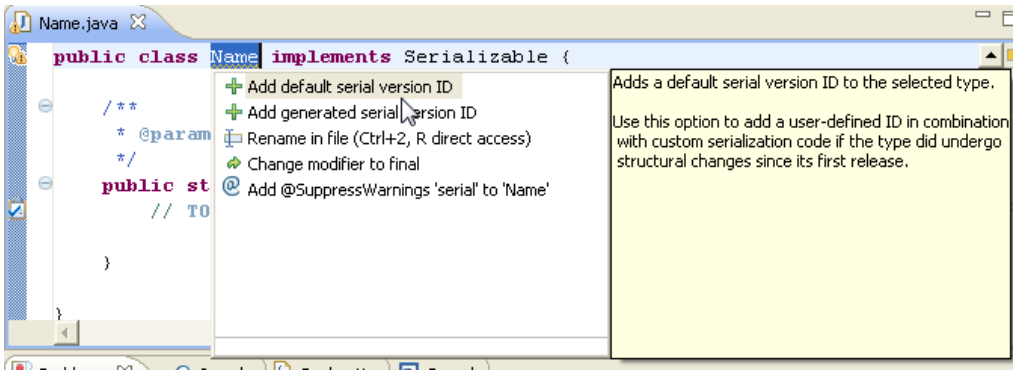




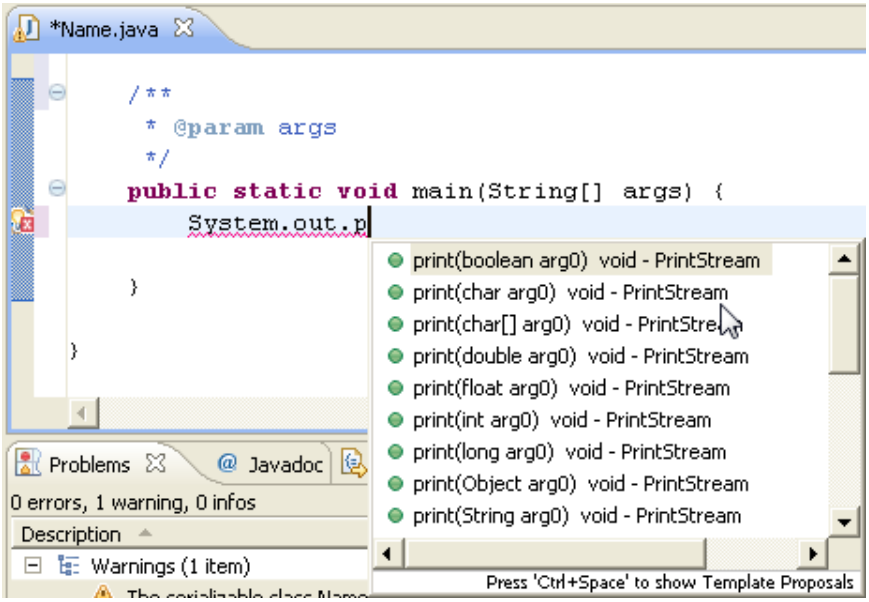
Schiebt man die Maus auf die Fehler- oder Problemmeldung, erhält man detailliertere Angaben, die auch im unteren Fenster „Problems“ sichtbar sind.



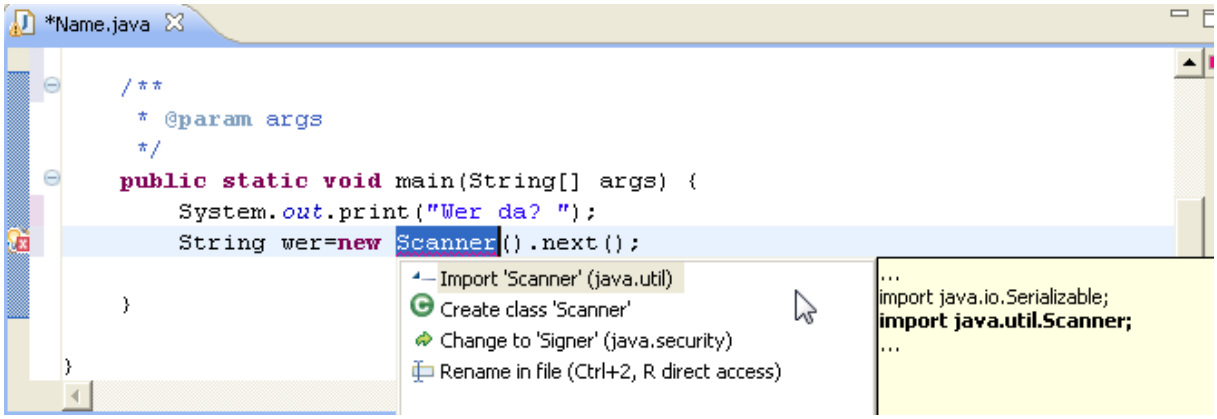
Macht man einen Linksklick auf eine Fehler- oder Problemmeldung, erhält man meist Korrekturvorschläge, von denen einer mit einem Linksklick angenommen werden kann.



Ohne auf die Details des Problems einzugehen, wird hier der erste Vorschlag genutzt und das Programm wie folgt ergänzt. Beim Tippen ist standardmäßig die Code-Vervollständigung eingeschaltet, die Vorschläge z. B. zur Methodenauswahl liefert (die Änderung der Editor-Einstellungen findet unter „Window->Preferences“ statt, was bereits erklärt wurde). Die Vervollständigung kann auch über „Strg+Space-Taste“ angefordert werden.

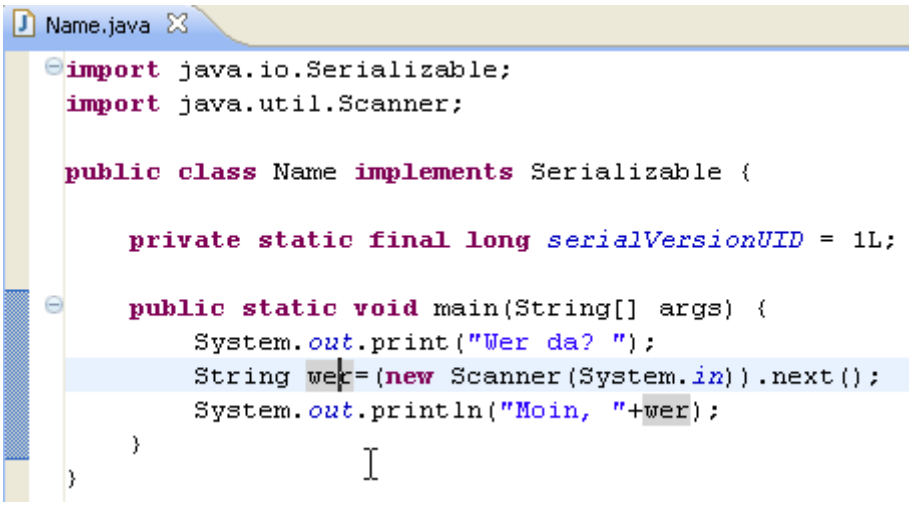


Werden Klassen genutzt, die zuerst importiert werden müssen, kann dies z. B. auch über den Fehlerdialog stattfinden. Man beachte, dass in Bearbeitung befindliche, noch nicht abgespeicherte Klassen oben im Reiter mit einem * gekennzeichnet sind.



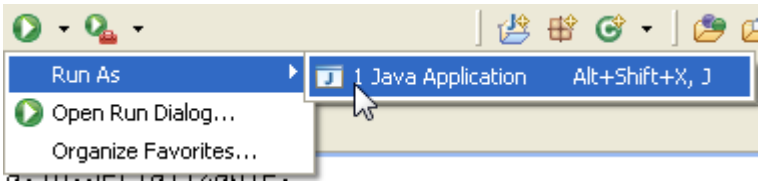
7.3 Starten von Java-Programmen

Das folgende Programm soll jetzt ablaufen. Man erkennt im folgenden Bild weiterhin eines der zahlreichen Features von Eclipse, dass beim Klicken auf eine Variable, hier „wer“, alle ihre Vorkommen angezeigt werden.

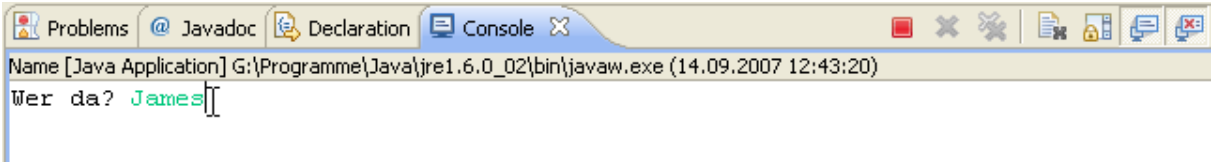


Beim ersten Start muss der kleine Pfeil nach unten neben dem eigentlichen Ausführungspfeil gedrückt werden.

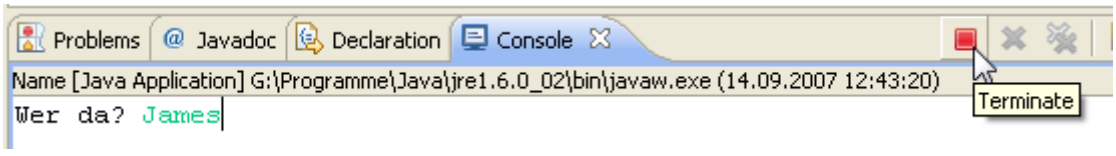
Im einfachsten Fall, der hier zunächst betrachtet wird, wird dann der Start als „Java Application“ ausgewählt. Es ist zu beachten, dass diese Auswahl nur möglich ist, wenn eine Klasse ausgewählt ist, die gerade aktiv im Editor bearbeitet werden kann und eine main()-Methode enthält.



Programmeingaben finden im typischerweise unten platzierten Konsolen-Fenster statt. Durch einen Klick auf dieses Fenster wird der Cursor automatisch richtig platziert. Weiterhin zeigt die Kopfzeile, was aktuell ausgeführt und welche Java-Variante genutzt wird.



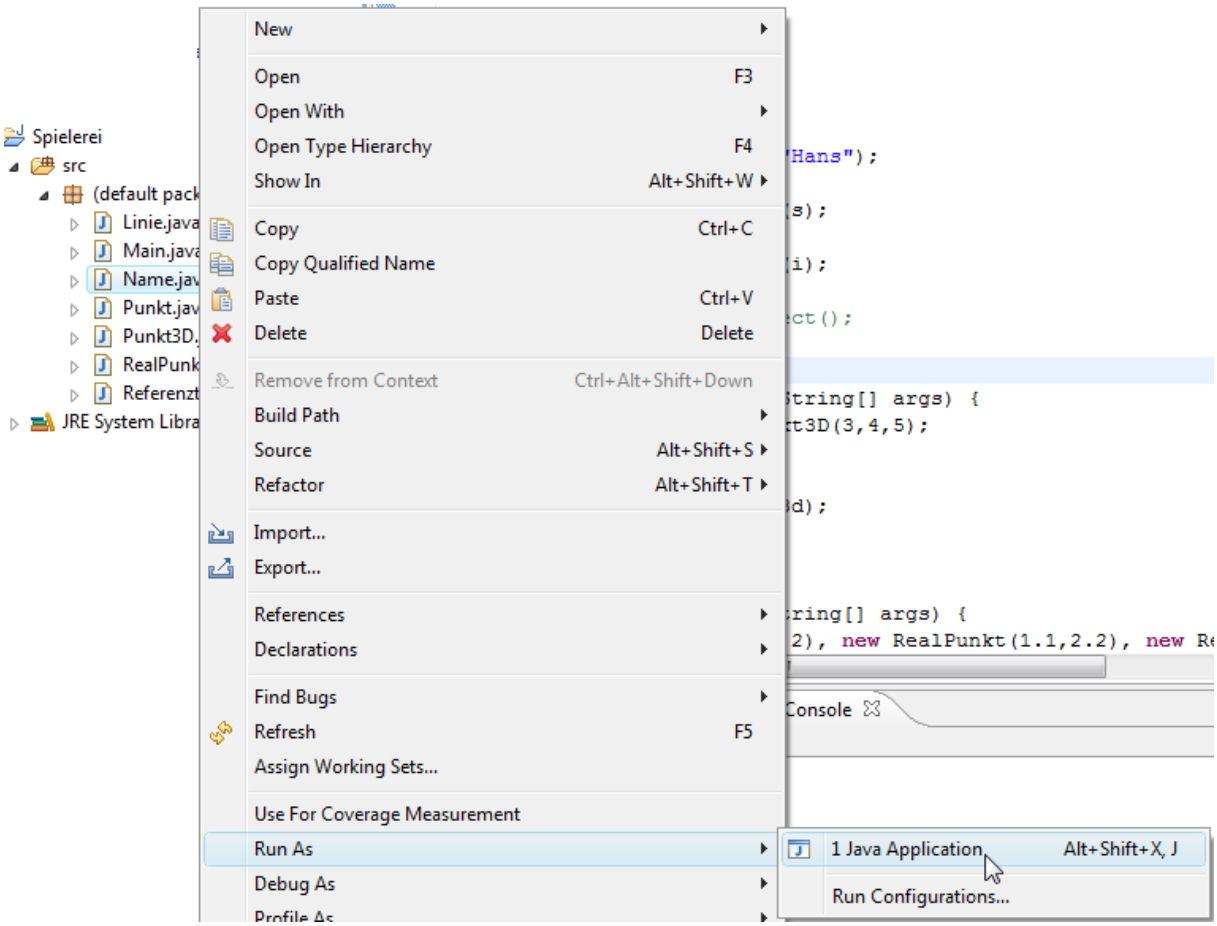
Das rote Quadrat zeigt an, dass dieses Programm noch läuft. Durch einen Klick auf dieses Quadrat könnte das Programm terminiert werden.



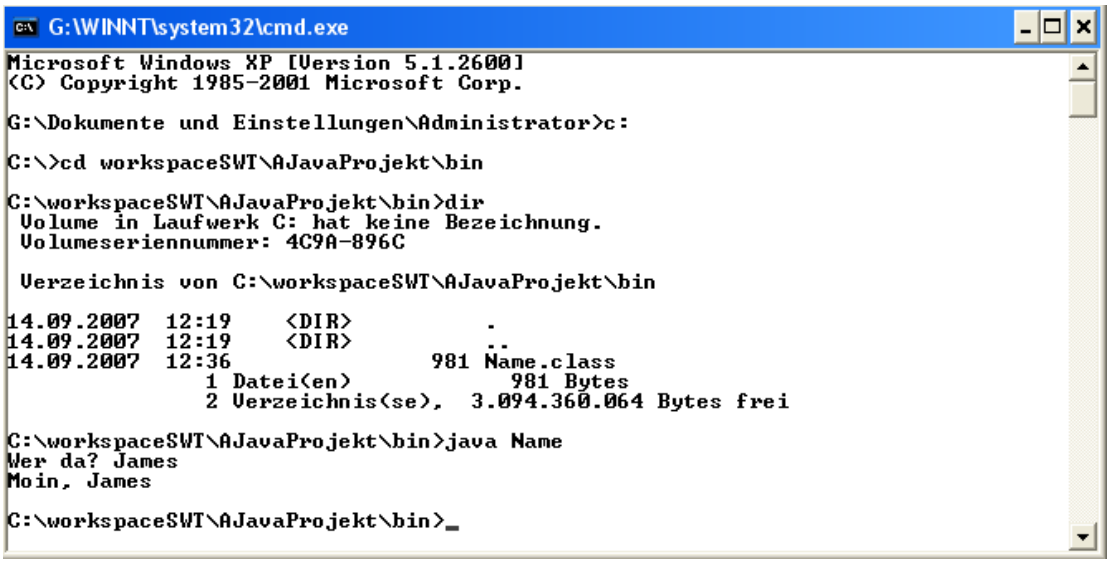
Ausgaben finden ebenfalls im Consolen-Fenster statt, weiterhin erkennt man an der Information <terminated>, dass das Programm beendet wurde.

Der erneute Programmaufruf mit den gleichen Randbedingungen wie vorher ist durch einen Klick direkt auf den Ausführungspfeil möglich.

Eine Variante mit der man genauso gut Programme starten kann, ergibt sich durch einen Rechtsklick auf die auszuführende Datei, gefolgt von „Run As -> Java Applikation“.

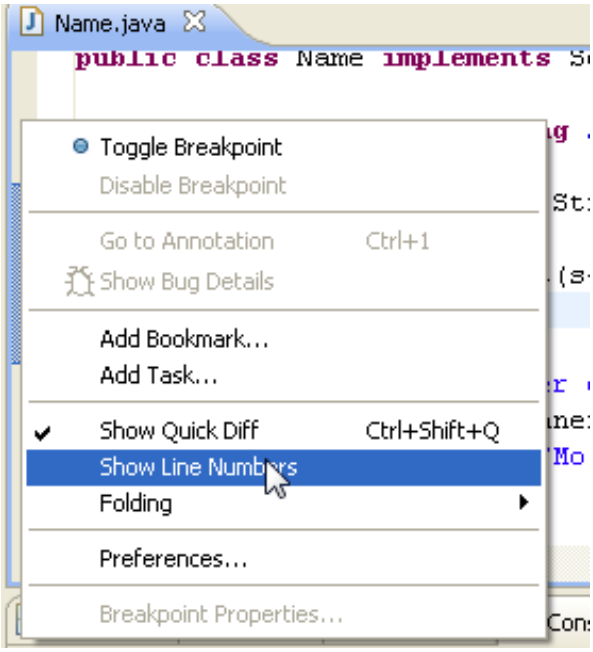


Java-Programme, die innerhalb von Eclipse laufen, können auch außerhalb gestartet werden. Dazu muss man wissen, wo das Projekt gespeichert ist und in das dortige bin-Verzeichnis wechseln.



7.4 Einschalten von Zeilennummern

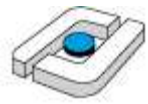
Hilfreich ist das bereits vorher beschriebene Einschalten der Zeilennummern, das auch über einen Rechtklick auf den linken Editorrand erreichbar ist.



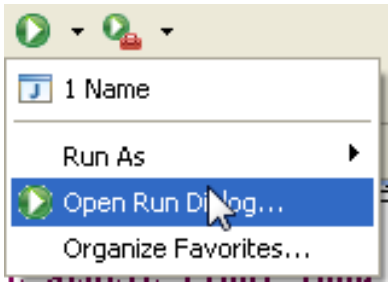
7.5 Parameterübergaben und Assertions einschalten

Die Main-Methode wird jetzt wie folgt abgeändert.

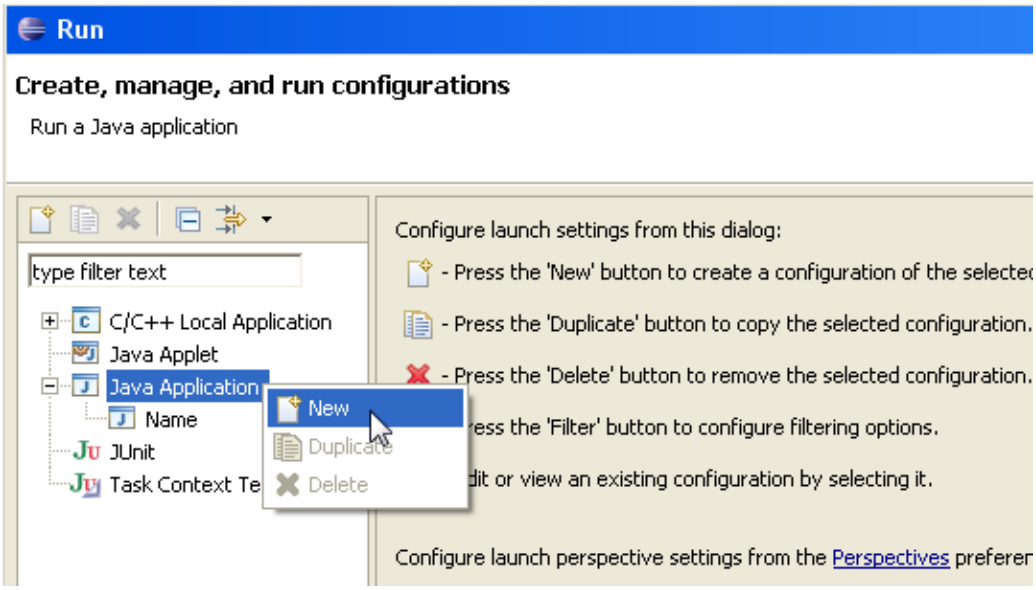
```
public static void main(String[] args) {
      for(String s:args)
            System.out.print(s+"\t");
      assert 42==0;

}
```



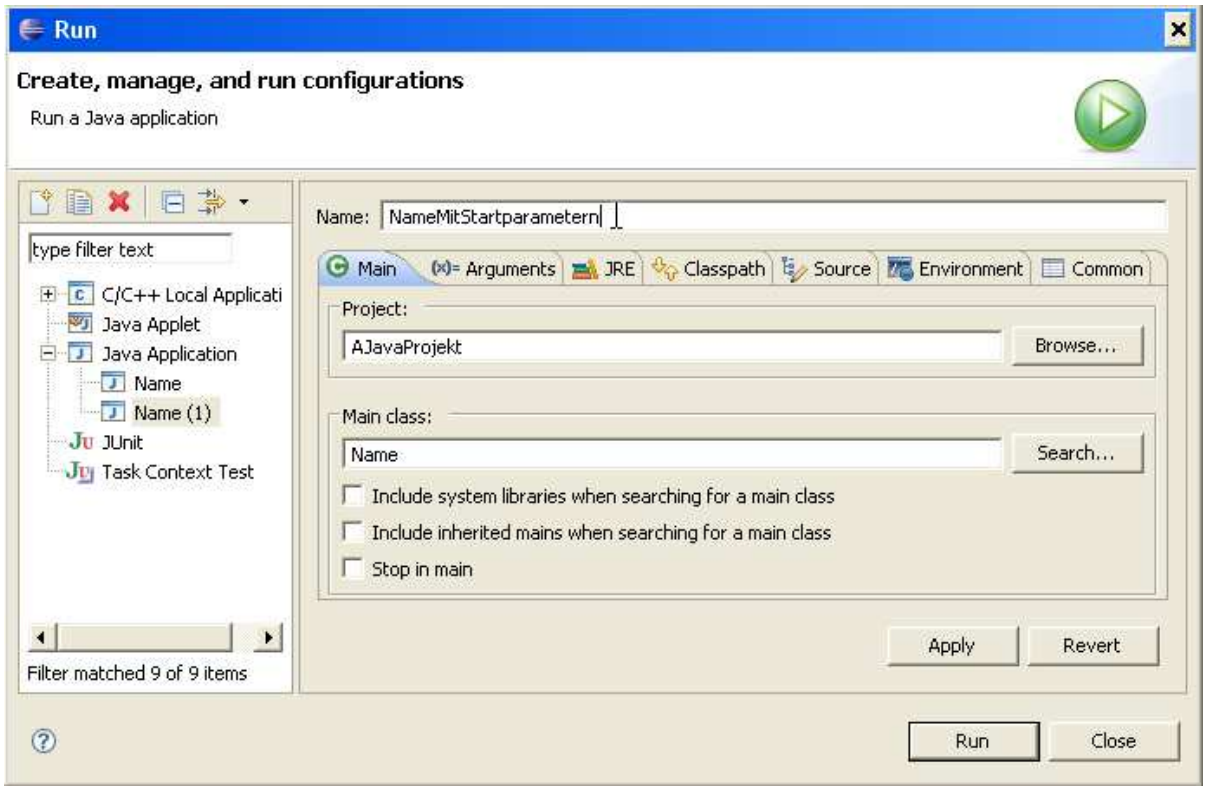
Möchte man in Java Parameter verarbeiten, die eigentlich über die Kommandozeile übergeben wurden, muss man zum Starten zunächst wieder den kleinen Pfeil nach unten neben dem Ausführungspfeil nutzen und „Open Run Dialog...“ wählen.



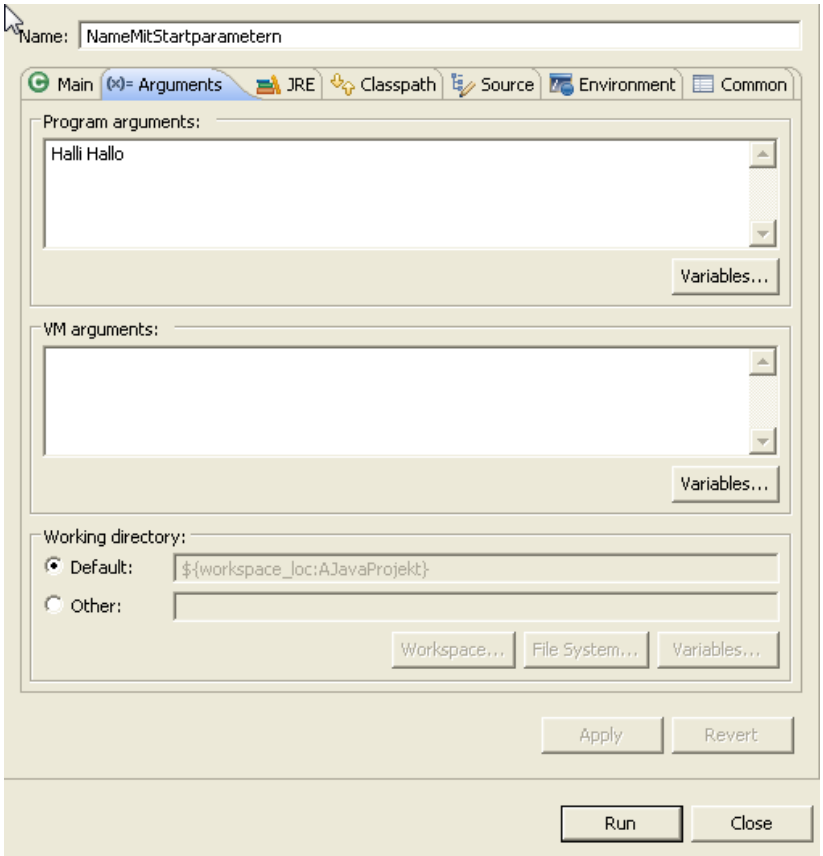
Man erreicht dann die Einstellungsmöglichkeiten, die für einen Programmstart gewählt werden sollen. Diese Einstellungen werden auch *Konfiguration* genannt. Zunächst wird durch einen Rechtsklick auf „Java Application“ „New“ ausgewählt.



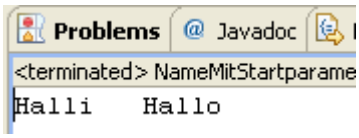
Jeder Konfiguration kann oben ein Name gegeben werden, so dass sie immer wieder genutzt werden kann. Auswählbare Konfigurationen wurden im vorherigen Fenster, dort z. B. nur „Name“, angezeigt.



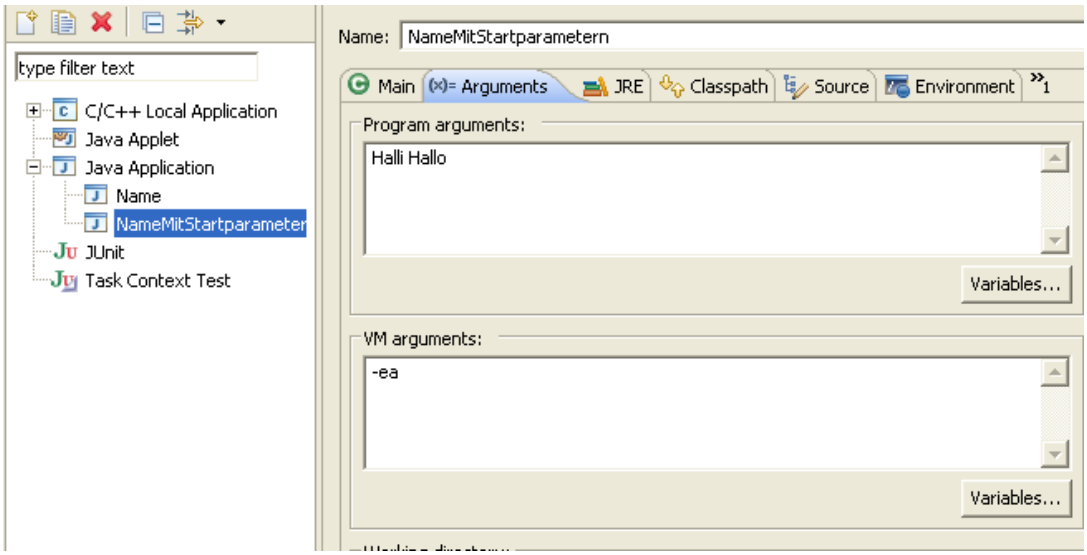
Um die Eingaben von der Kommandozeile anzugeben, gibt es den Reiter „(x)= Arguments“.



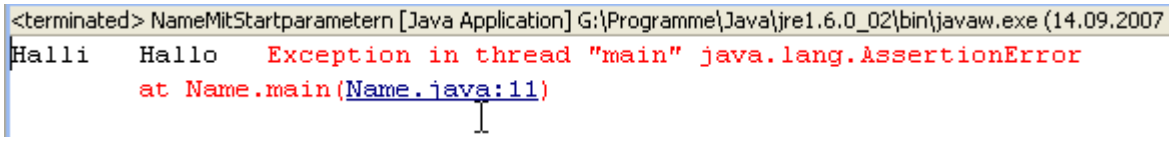
Wird das Programm jetzt mit „Run“ gestartet, erhält man folgendes Ergebnis.



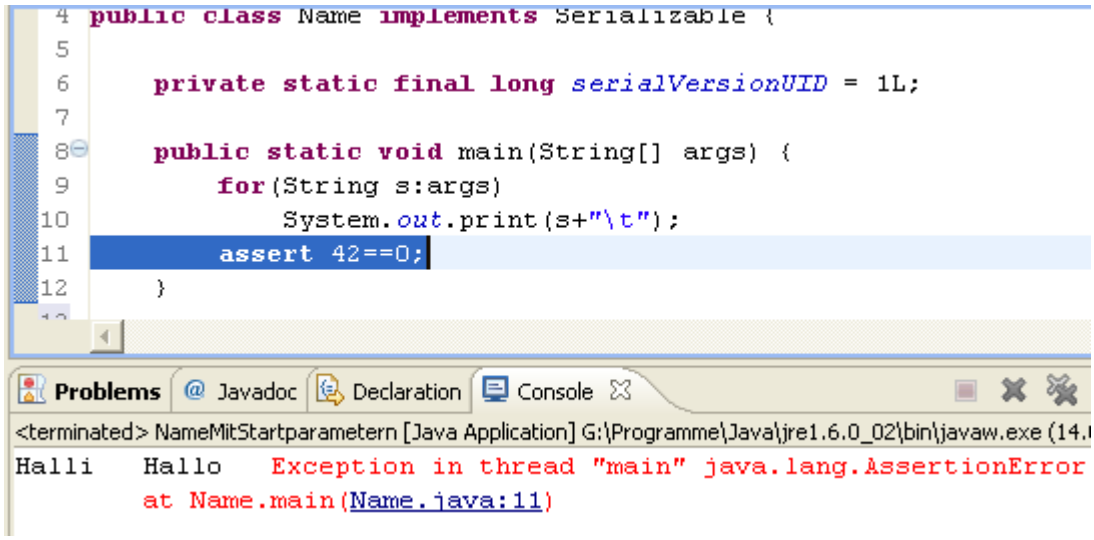
Es fällt auf, dass die Assertion nicht ausgeführt wird. Assertions müssen erst eingeschaltet werden. Dazu wechselt man wieder mit „Open Run Dialog“ zu der jetzt auswählbaren Konfiguration. In der Konfiguration wird dann der virtuellen Maschine mitgeteilt, dass die Assertions eingeschaltet werden sollen (-ea).



Die Ausgabe sieht dann wie folgt aus.

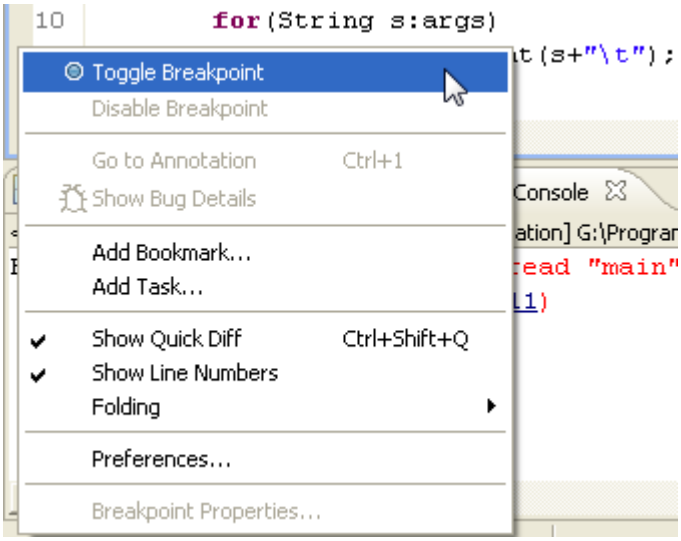


Durch einen Klick auf die markierte Fehlermeldung erreicht man sofort die dazugehörige Fehlerstelle.

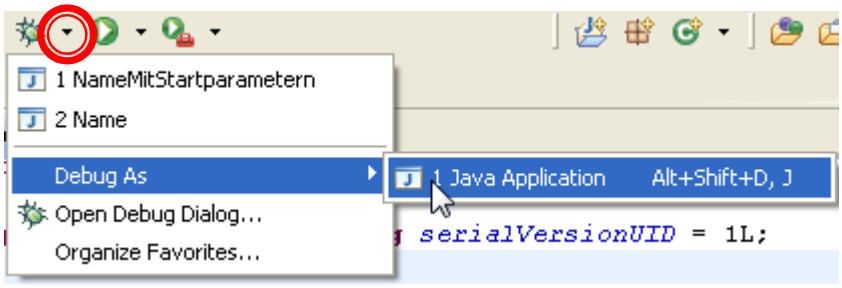


7.6 Kurzeinführung in den Debugger

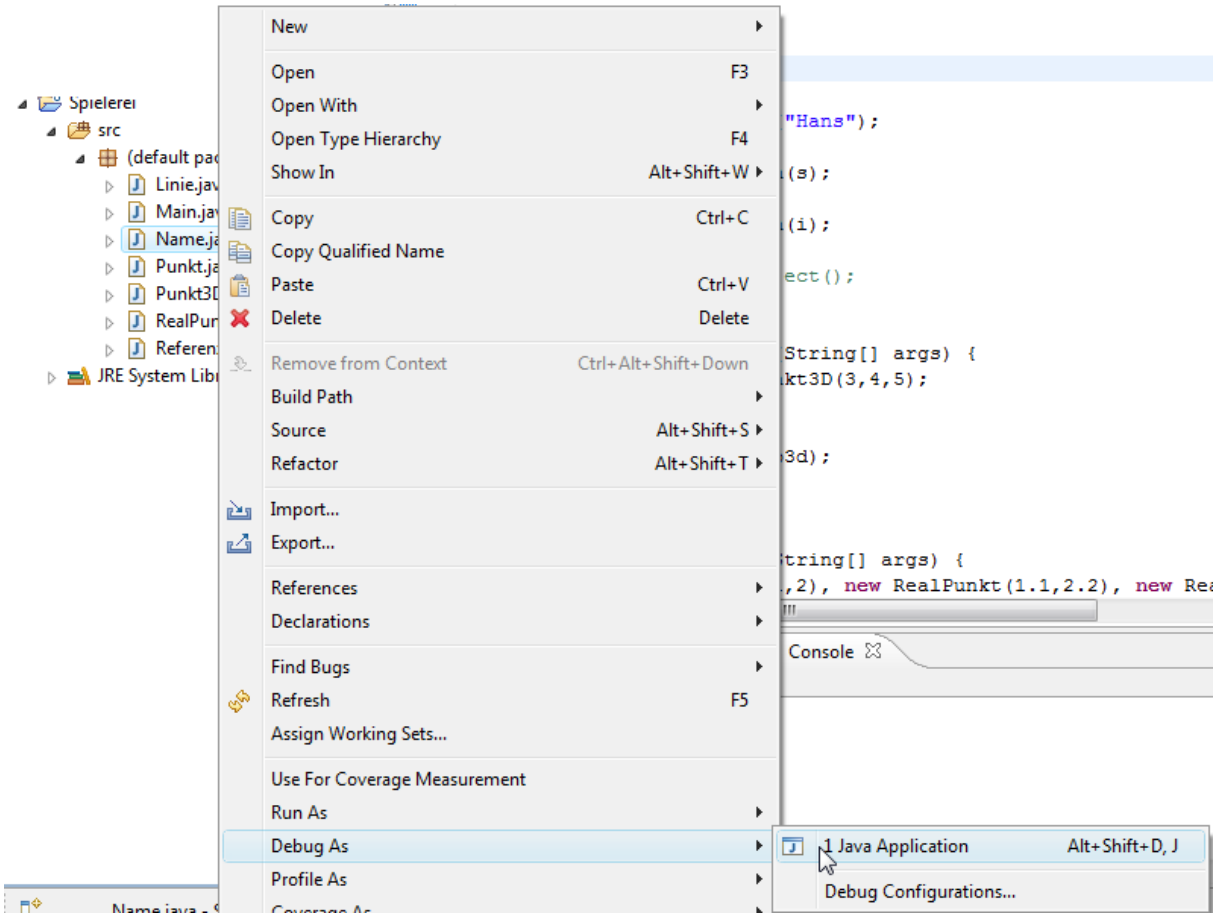
Neben dem bei recht genau identifizierbaren Fehlern sicherlich legitimen Ansatz, bei einem Fehler sich Werte mit einem print auszugeben, besitzt Eclipse einen sehr mächtigen Debugger, mit dem der Programmablauf genauestens verfolgt werden und man jederzeit die genauen Werte der Variablen angezeigt bekommen kann. Man kann direkt in die Debug-Sicht wechseln oder in der Java-Sicht mit einem Rechtsklick auf dem Rand einen sogenannten Break-Point setzen. Kommt das Programm dann zu dieser Stelle, wird es angehalten und kann im Debugger verfolgt und verändert werden.



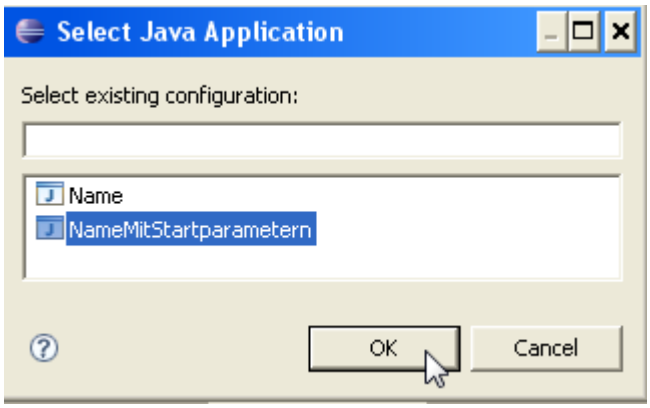
Zum Debuggen muss das Programm beim ersten Mal über den kleinen Pfeil nach unten neben dem Käfer ausgeführt werden.



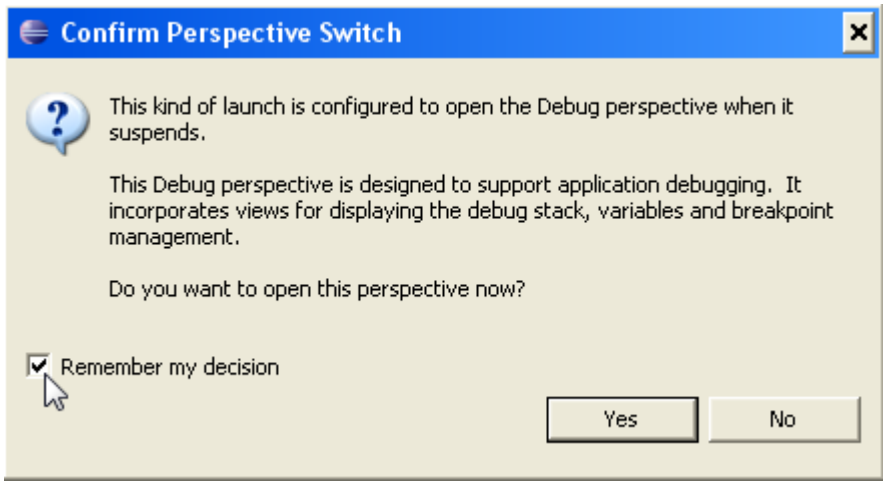
Alternativ kann man wieder über einen Rechtsklick auf der Datei mit „Debug As -> Java Application“ den Debugger starten.



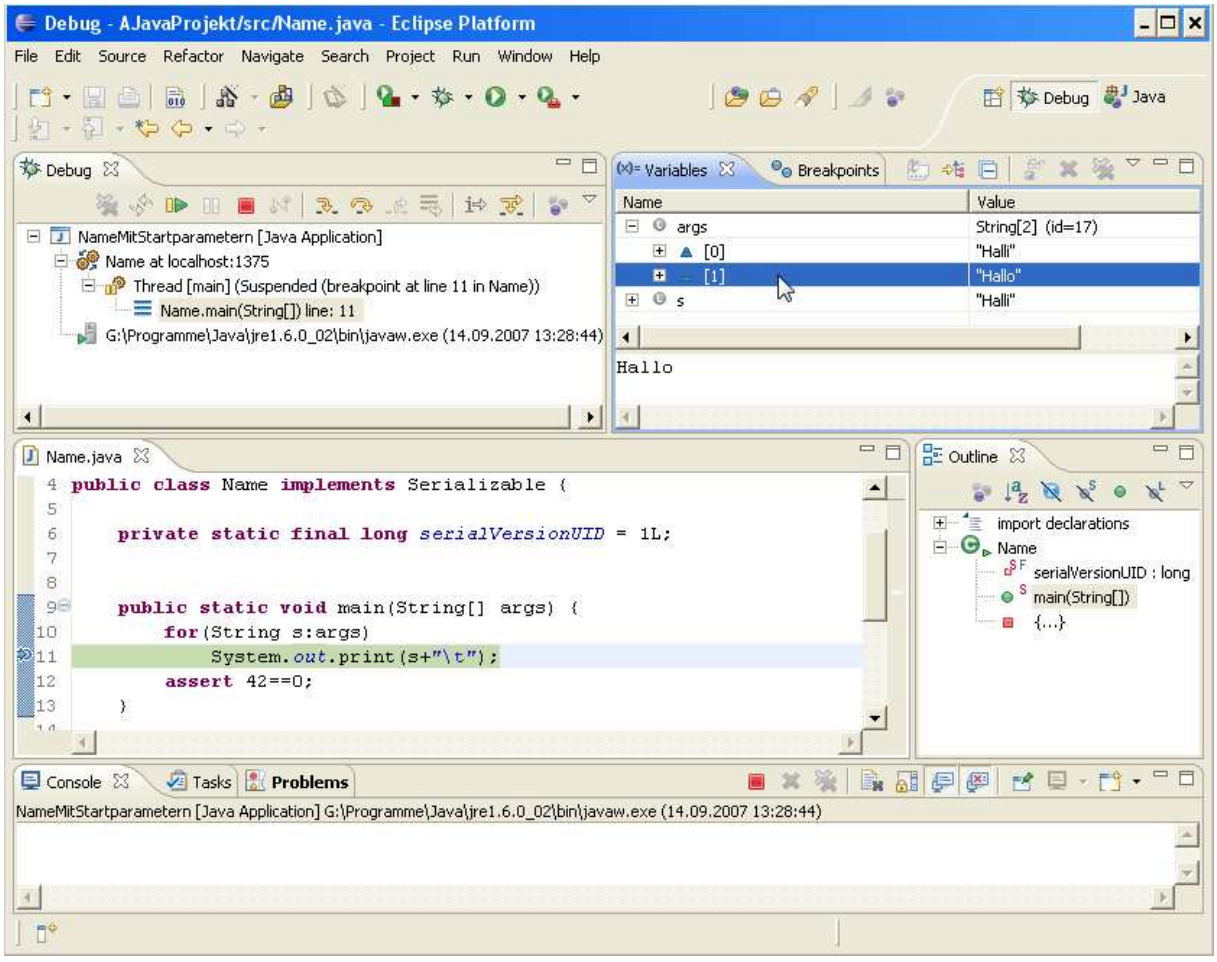
Falls es mehrere Konfigurationen für ein Programm gibt, wird man zur Auswahl einer Konfiguration aufgefordert.



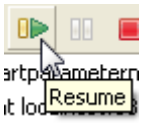
Das Programm hält dann bei der Ausführung beim Breakpoint an und man erhält die folgende Anfrage, die man „für immer“ beantworten kann.



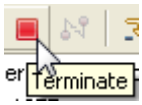
Das Debug-Fenster sieht wie folgt aus. Der Reiter „Debug“ zeigt die laufenden Programme und die zum Java-Programm gehörenden Threads. Links in der Mitte sieht man das aktuelle Programm mit einer Markierung, an welcher Stelle es sich befindet. Rechtsoben hat man unter dem Reiter „Variables“ die Möglichkeit, die aktuellen Variablenwerte im Detail anzuschauen. Bei der Darstellung wird die toString()-Methode der jeweiligen Variable genutzt.



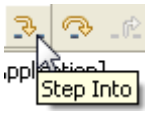
Im Debug-Fenster kann die Programmausführung im Detail gesteuert werden. Es gibt folgende Möglichkeiten.



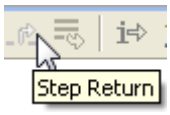
Die Programmfortsetzung bis zum Erreichen des nächsten Breakpoints.



Der Abbruch des Debugvorgangs.



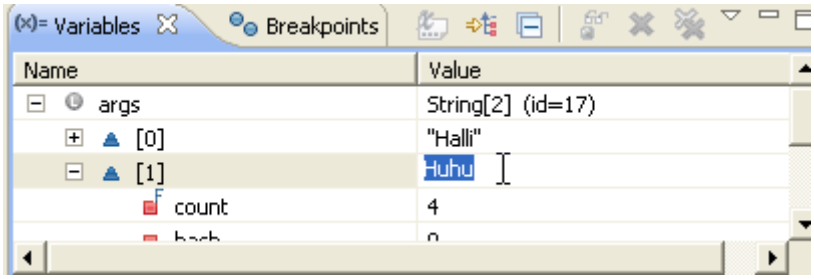
Das Hineingehen in eine aufgerufene Methode, um deren Ablauf zu verfolgen.



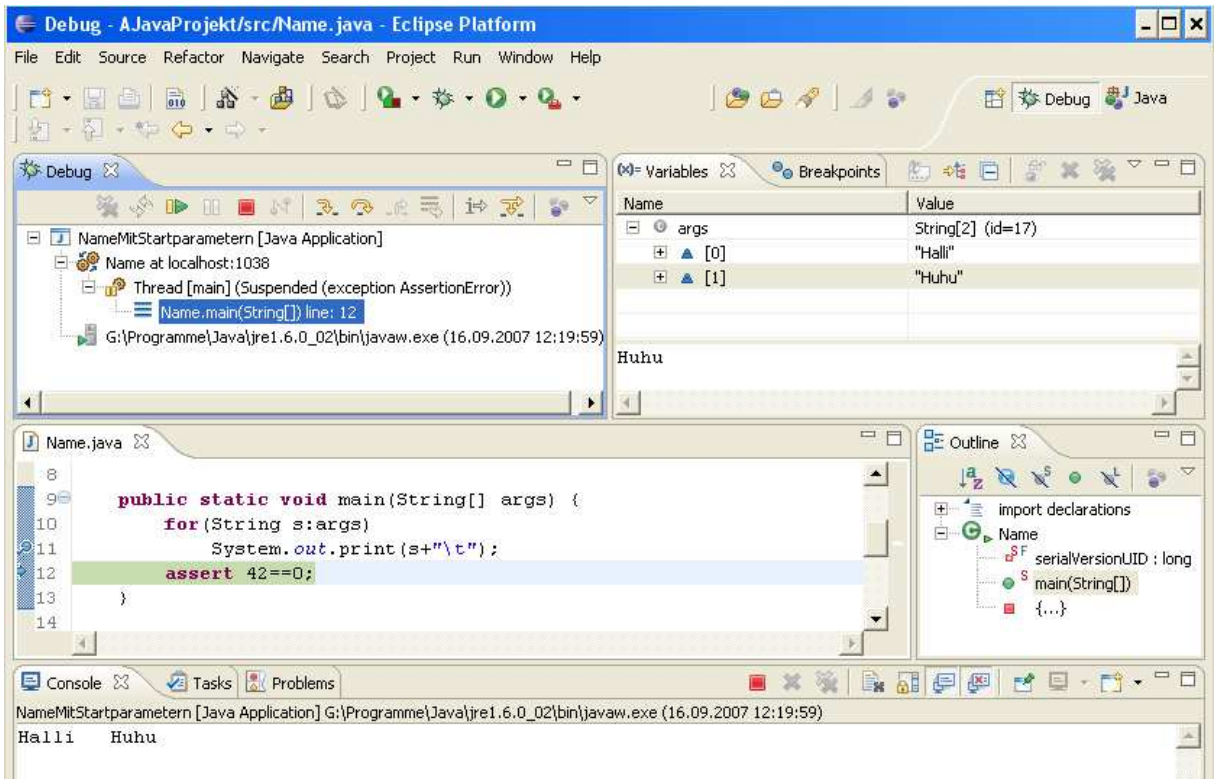
Der Rücksprung aus der aktuell bearbeiteten Methode (die dann abgeschlossen wird). Das Debuggen geht nach dem Methodenaufruf weiter.

Die Nichtverfolgung eines Methodenaufrufs, es wird nach der Beendigung der Methode fortgesetzt.

Unter dem Reiter Variables können die Werte der Variablen zur Laufzeit geändert werden, wie folgendes Beispiel zeigt, bei dem einfach auf den Wert der Variablen geklickt wurde.

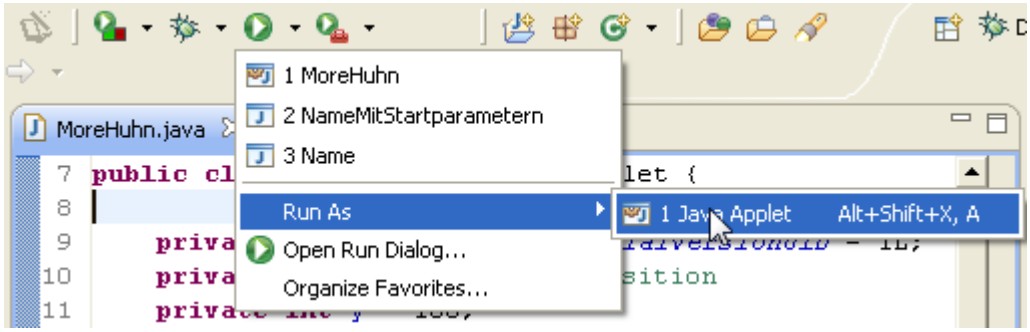


Die Veränderung der Variable wird im Programm berücksichtigt, was folgende Ausgabe zeigt.

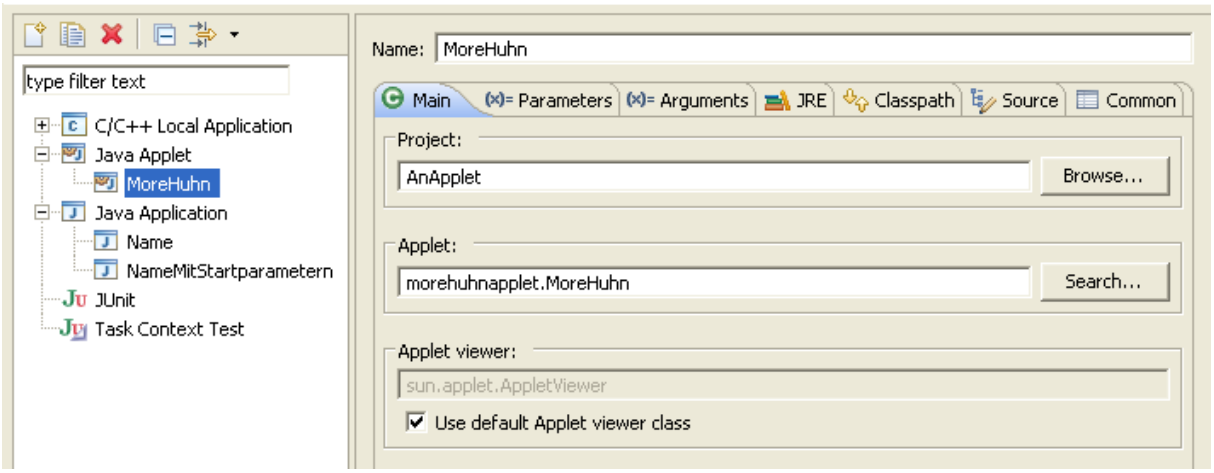


7.7 Starten von Applets

Von Eclipse aus kann auch direkt ein Applet aufgerufen werden. Die Startmöglichkeit bietet sich unmittelbar in der Auswahl neben dem Ausführungspfeil, wenn die Klasse von Applet erbt.

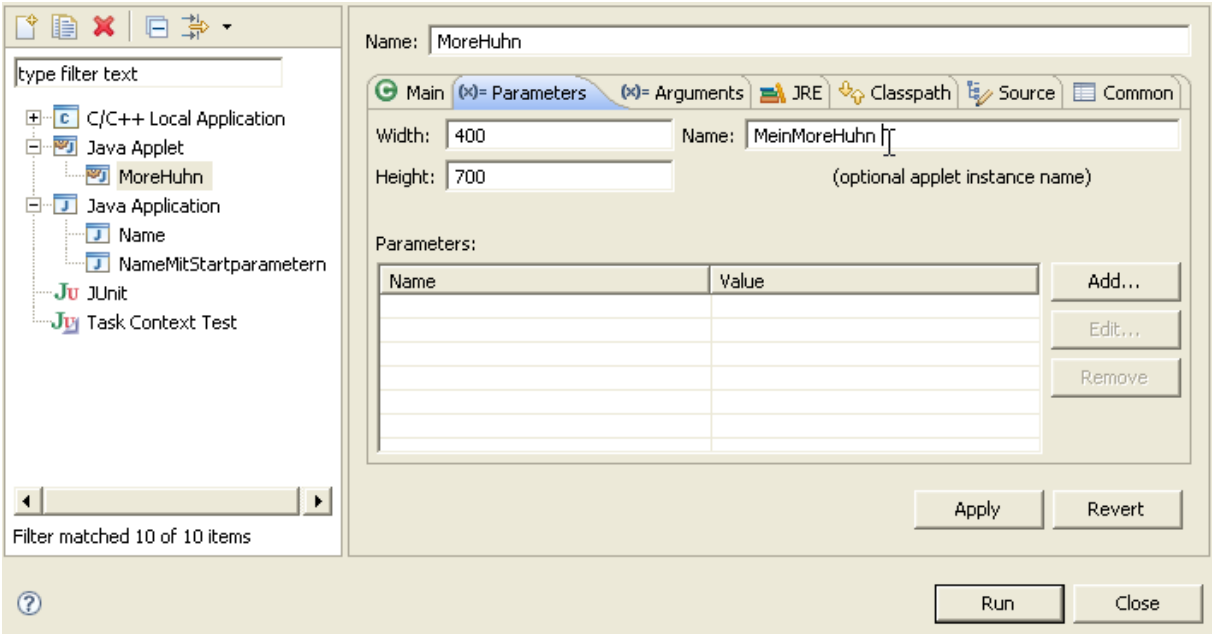


Es wird eine Standardeinstellung genutzt, die zwar änderbar ist, wobei besser die Appletparameter in der Startkonfiguration projektindividuell festgelegt werden. Dazu erfolgt der Start einmal über „Open Run Dialog“.



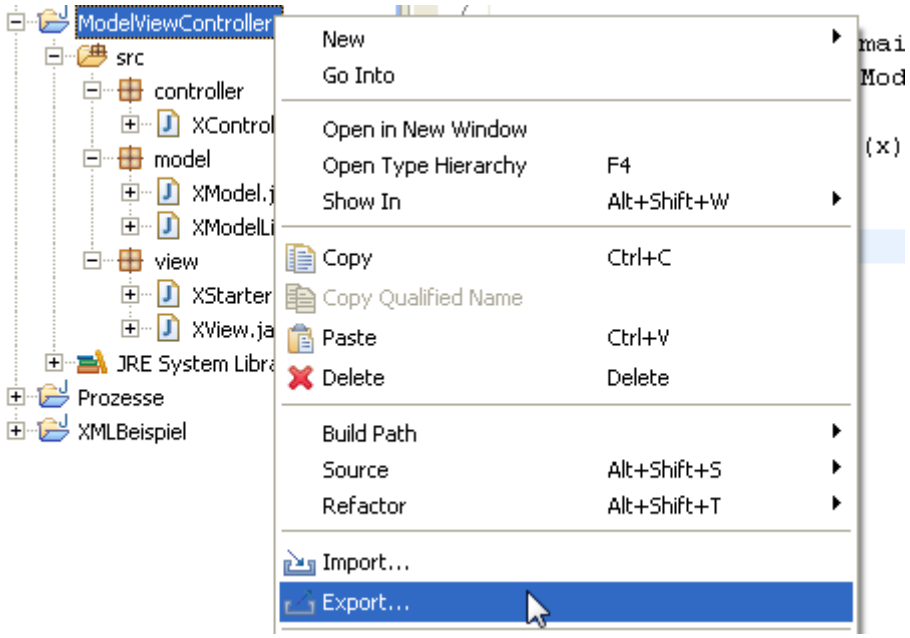
Hier kann mit einem Rechtsklick auf „Java Applet“ mit „New“ eine neue Konfiguration angelegt werden. Alternativ kann, falls vorhanden, eine bereits vorhandene Konfiguration angepasst werden.

Unter „(x)= Parameters“ kann die Größe des Applets angegeben werden. Weitere Parameter, die auch im HTML-Code stehen können, werden hier mit „Add...“ über die üblichen Parametername/Wert-Paare ergänzt. Das Applet wird über „Run“ gestartet. Diese Einstellungen müssen nur einmal eingegeben werden, danach ist die Konfiguration z. B. über das Ausführungsmenü, das man unter dem kleinen Pfeil rechts neben dem Ausführungspfeil findet, auswählbar.

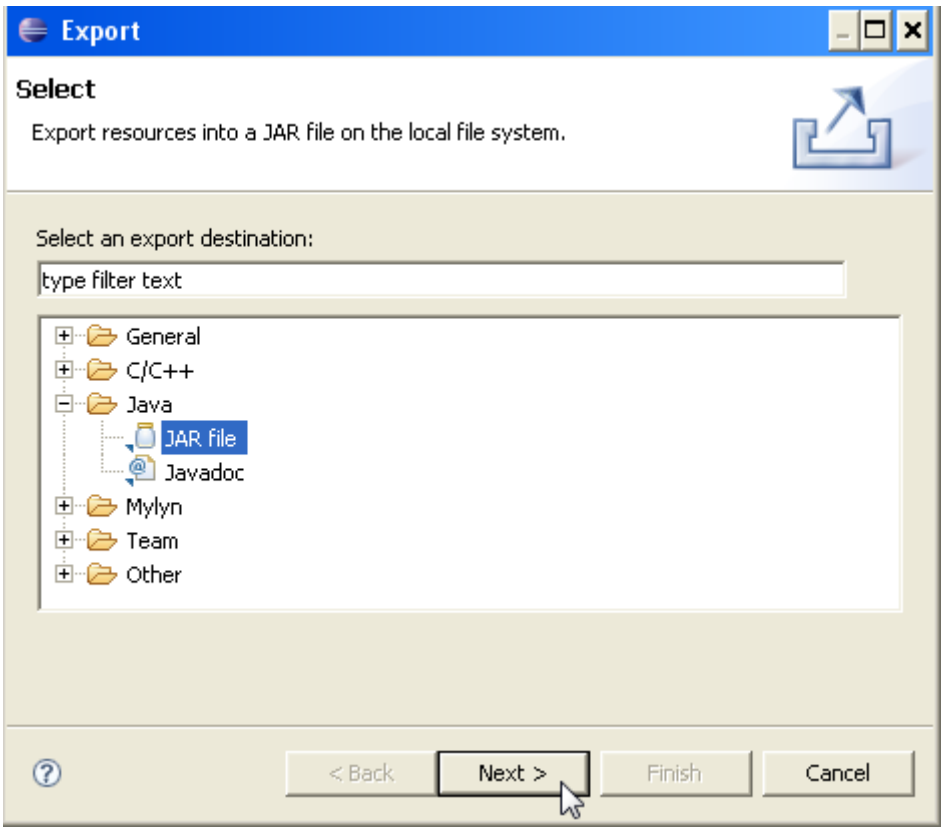


7.8 Packen von JAR-Dateien

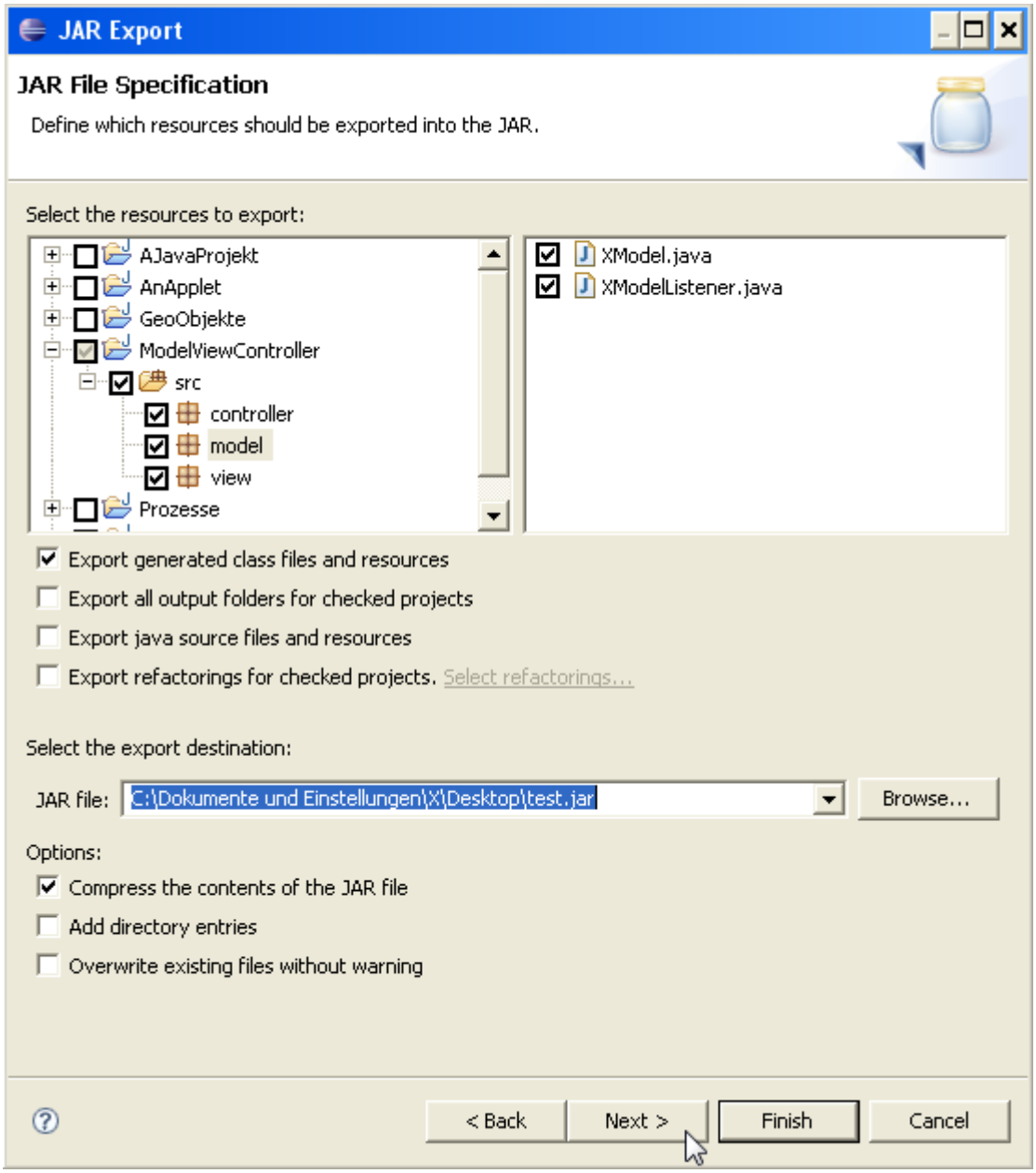
Hat man seine Klassen in Paketen entwickelt, kann man das Ergebnis in einem jar-File zusammenpacken, dass dann teilweise direkt zum Starten des Programms mit einem Doppelklick genutzt werden kann. Das Packen beginnt z. B. mit einem Rechtsklick auf das Projekt, es wird „Export...“ gewählt.



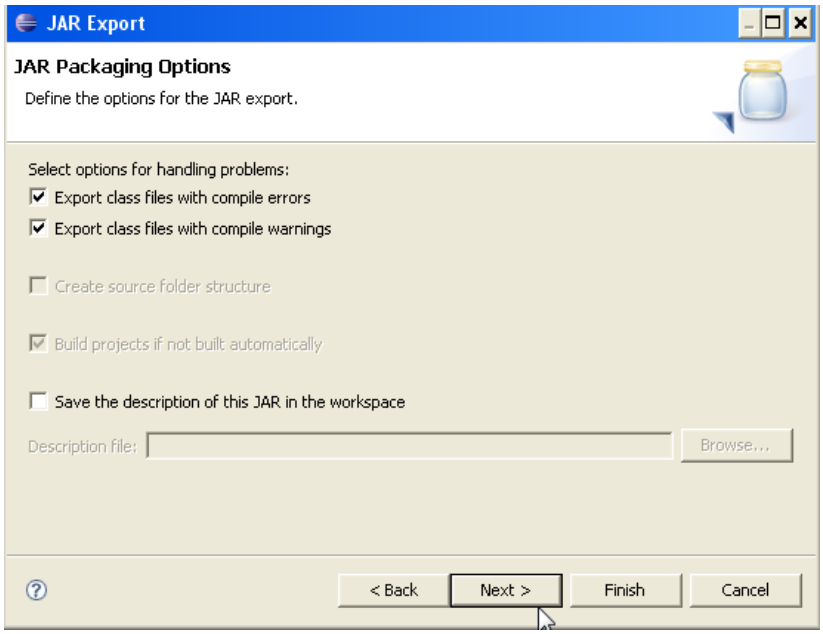
Unter dem Punkt „Java“ dann „JAR file“ auswählen und „Next>“ drücken.



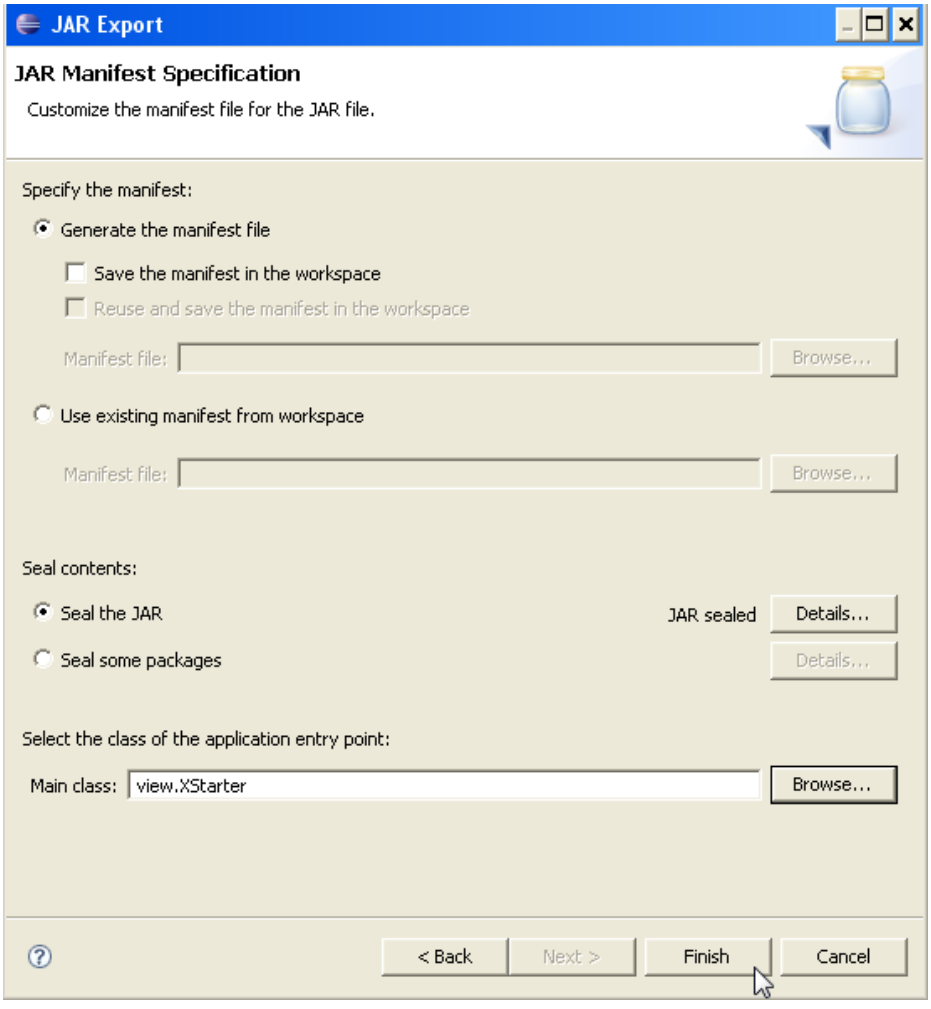
Danach können einige meist selbsterklärende Einstellungen vorgenommen werden. Im oberen Teil des Fensters werden die Dateien zum Packen ausgewählt. Dabei wird bei Endversionen auf die mit einem Punkt beginnenden Einstellungsdateien von Eclipse meist verzichtet. In der Auswahl darunter kann man festlegen, ob nur die ausführbaren Dateien, oder auch der Source-Code mit eingepackt werden soll. Dann wird der Speicherort für die Jar-Datei festgelegt und „Next>“ gedrückt.



Die folgenden Einstellungen können beim ersten Experimentieren so übernommen werden. Es wird einfach „Next>“ gedrückt.

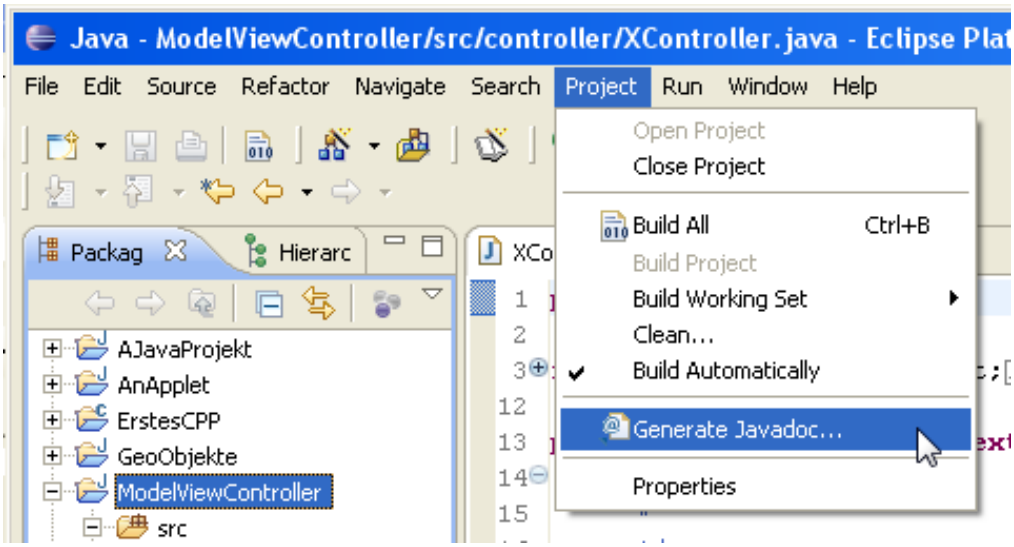


Im abschließenden Fenster muss oben zunächst festgelegt werden, woher die Manifest-Datei stammt. Am Anfang kann man diese generieren lassen. Weiter unten muss eine Startdatei angegeben werden, damit das Jar-File automatisch starten kann. Mit „Finish“ wird die Erstellung abgeschlossen.



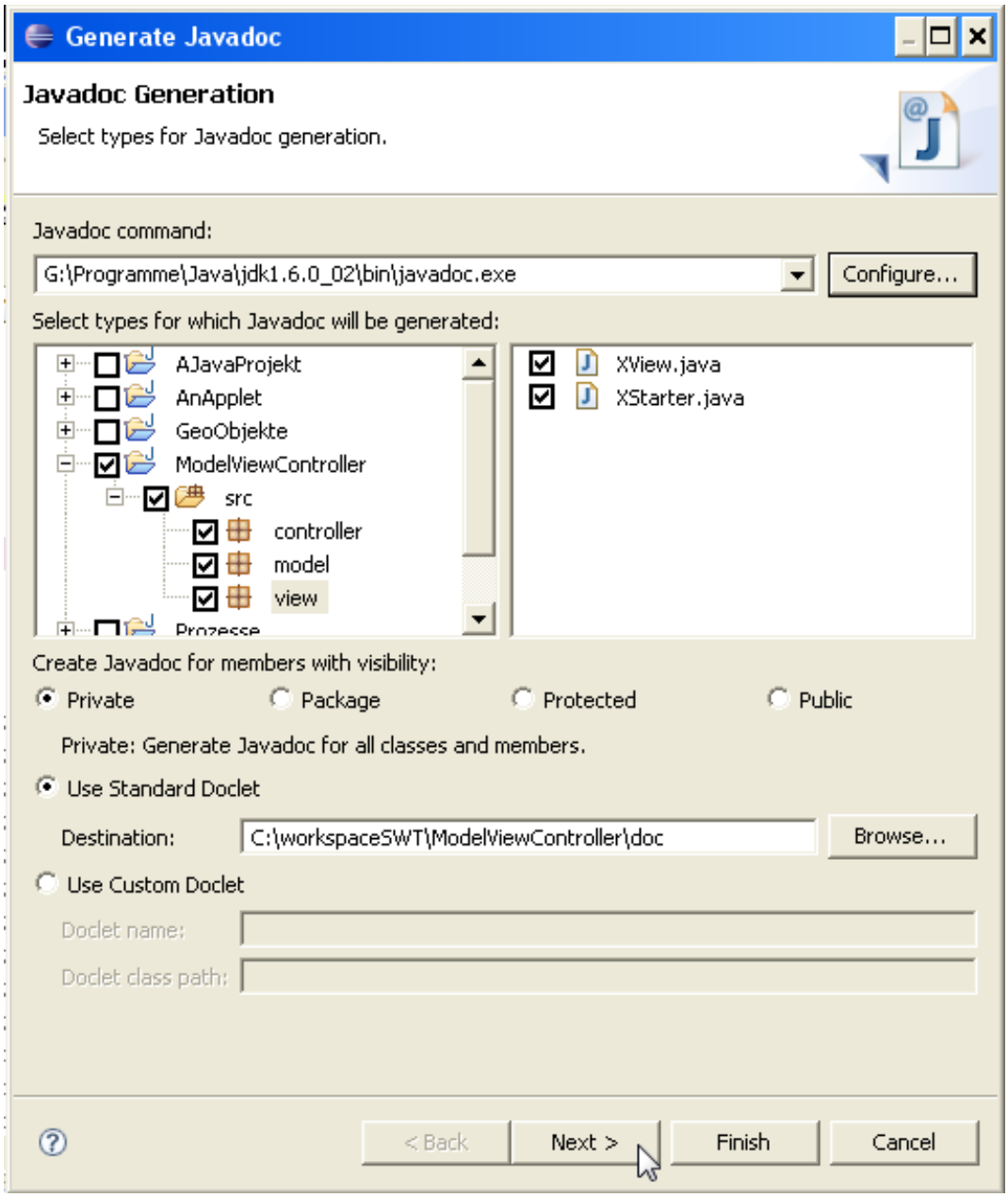
7.9 Dokumentengenerierung

Die Dokumentationserzeugung kann recht einfach für ein Projekt von Eclipse aus gestartet werden. Dazu wird z. B. ein Projekt ausgewählt und oben unter Projekt der Punkt „Generate Java-Doc...“ gewählt.

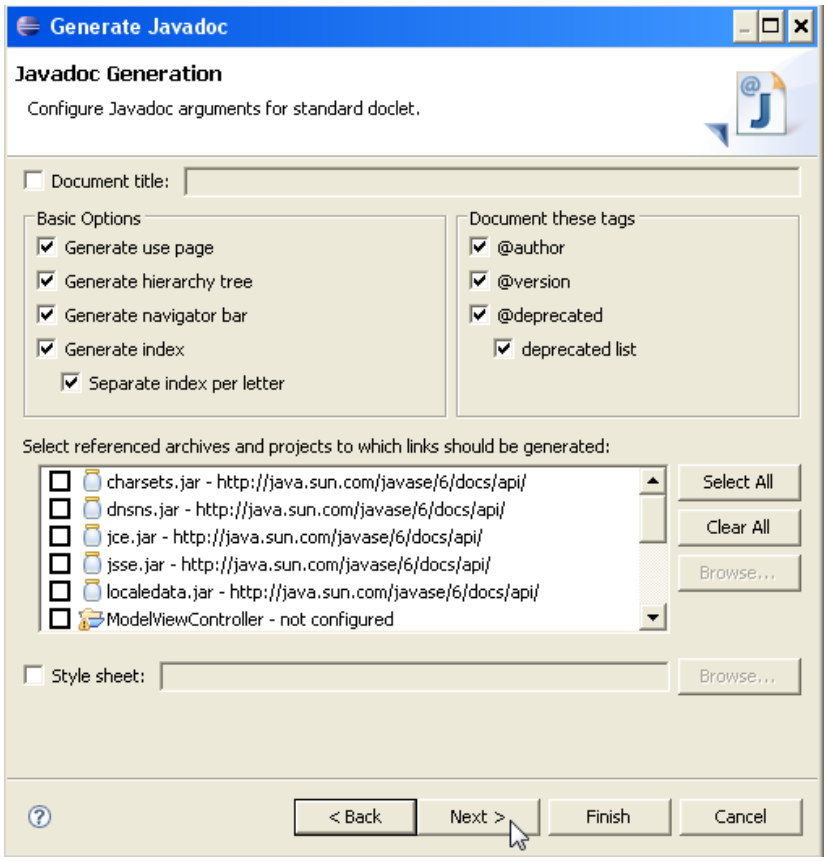


Etwas abhängig von der Java-Installation, muss in der obersten Zeile der Pfad zum genutzten javadoc-Programm eingestellt werden.

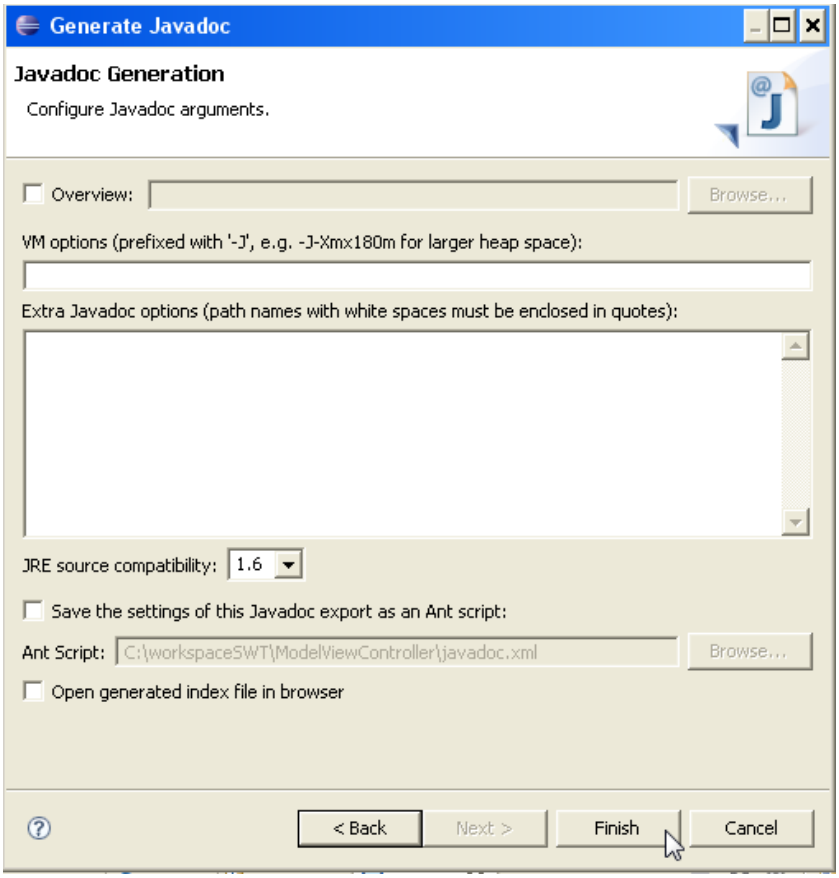
Im oberen Bereich können dann die Details ausgewählt werden, was aus welchen Projekten in die Generierung einfließen soll. Weiter unten muss angegeben werden, welche Sichtbarkeiten beim Generieren genutzt werden sollen. Während der Entwicklung wird dies meist auf private gesetzt, bei Auslieferungen auf public oder protected. Man kann das gesamte Layout der Dokumentation anpassen, wird dies nicht gewünscht, ist das „Standard Doclet“ nutzbar. Das vorgeschlagene Zielverzeichnis für die Dokumentation könnte angepasst werden, ist aber zumindest für kleinere Projekte sinnvoll. Für Detaileinstellungen kann dann „Next>“ gewählt werden.

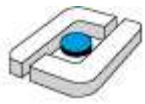


Hier kann u. a. angegeben werden, welche Tags in die Generierung einfließen. Wählt man dann „Next>“, kann man weitere Einstellungen vornehmen.



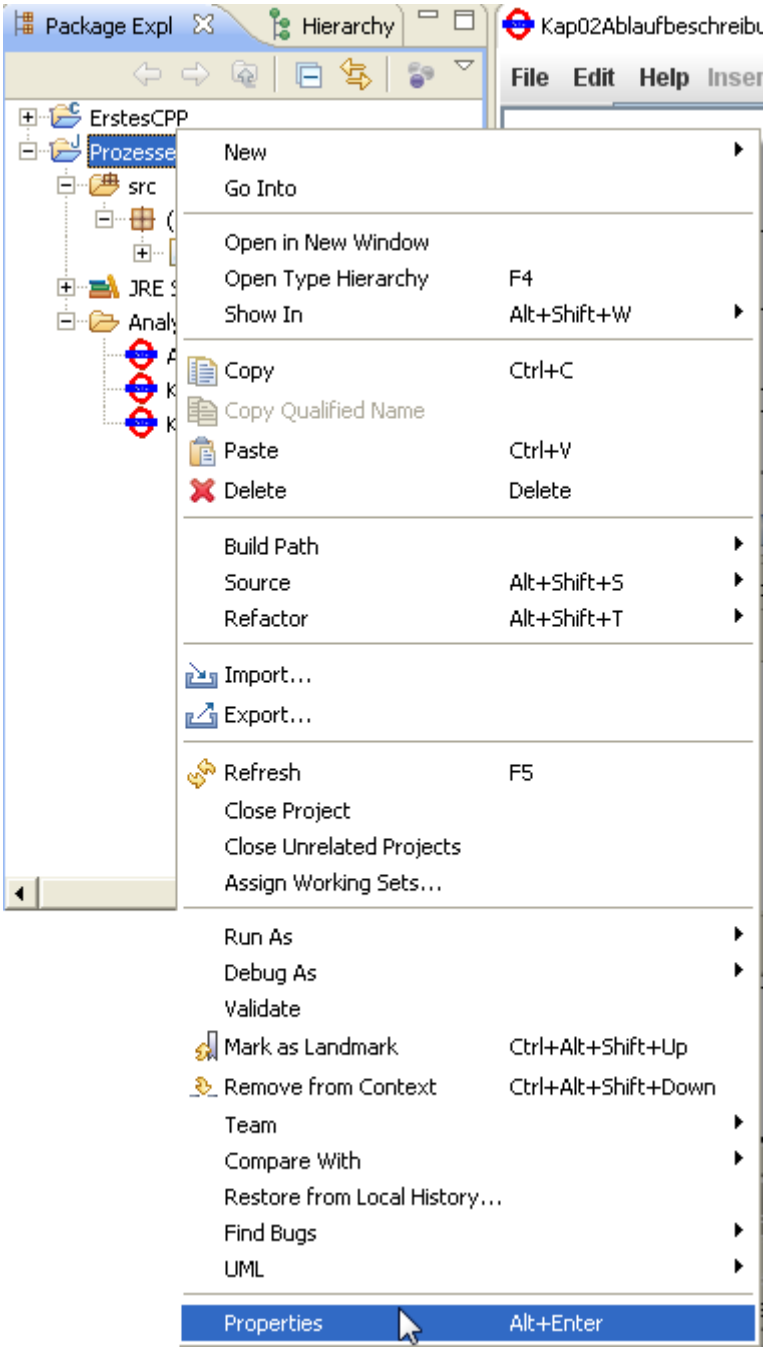
Dabei sind diese Einstellungen meist so übernehmbar und mit „Finish“ abzuschließen.



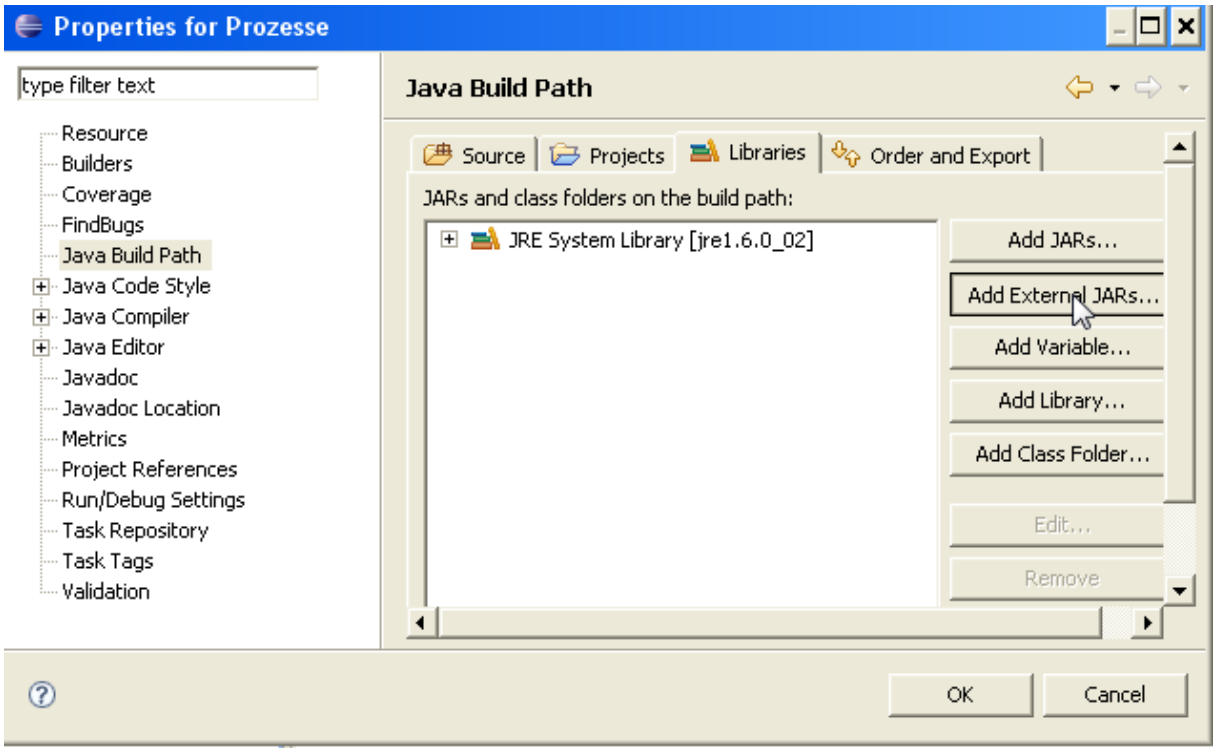


7.10 Nutzung weiterer Jar-Dateien

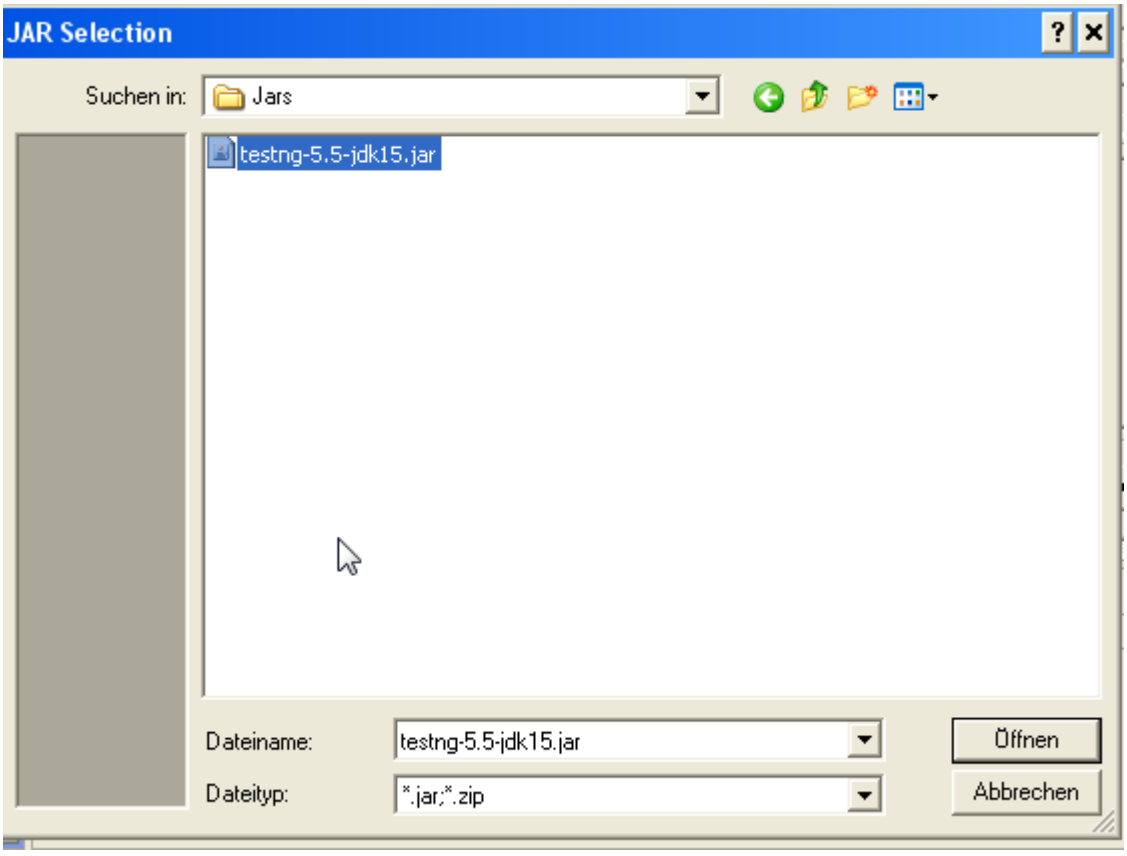
Möchte man weitere Jar-Dateien in einem Projekt nutzen (nicht bearbeiten !), müssen diese Jar-Dateien im Projekt bekannt gemacht werden. Dies ist z. B. so möglich, dass man einen Rechtsklick auf das Projekt macht und „Properties“ wählt.



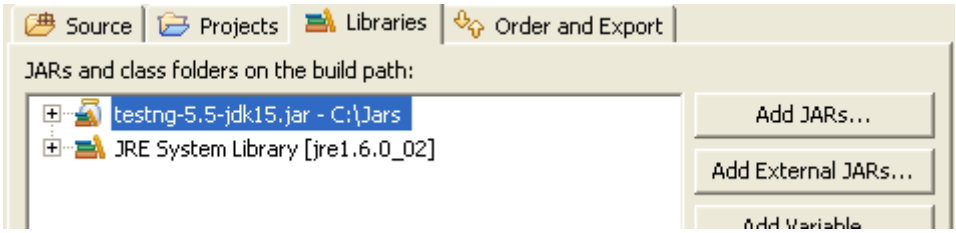
Hier kann unter „Java Build Path“ der Reiter „Libraries“ gewählt und die zu nutzende Jar-Datei unter „Add external JARs...“ ergänzt werden.



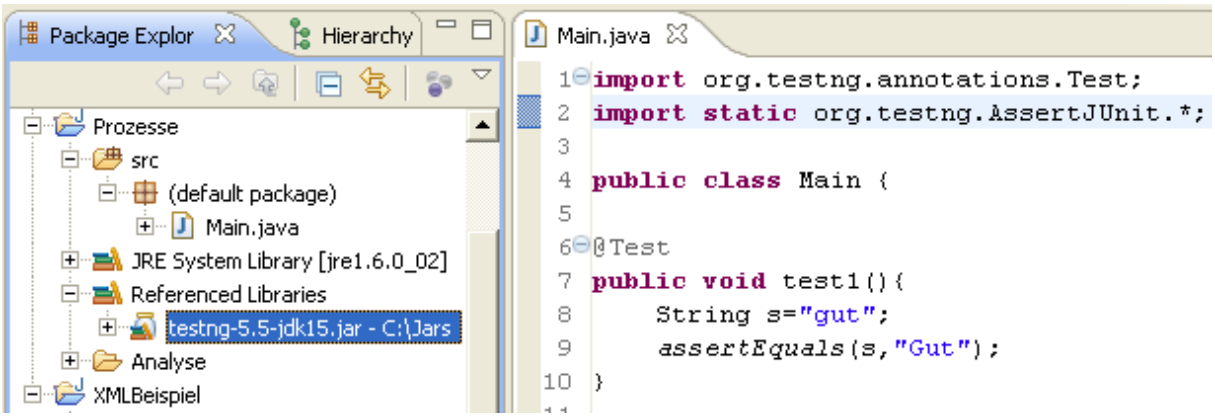
Genauer können hier jar- oder zip-Dateien eingebunden werden.

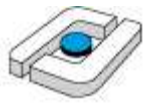


Dies wird dann auch bei den Projekteigenschaften sichtbar.



Weiterhin ist die nun nutzbare Bibliothek auch im Projektordner erkennbar.

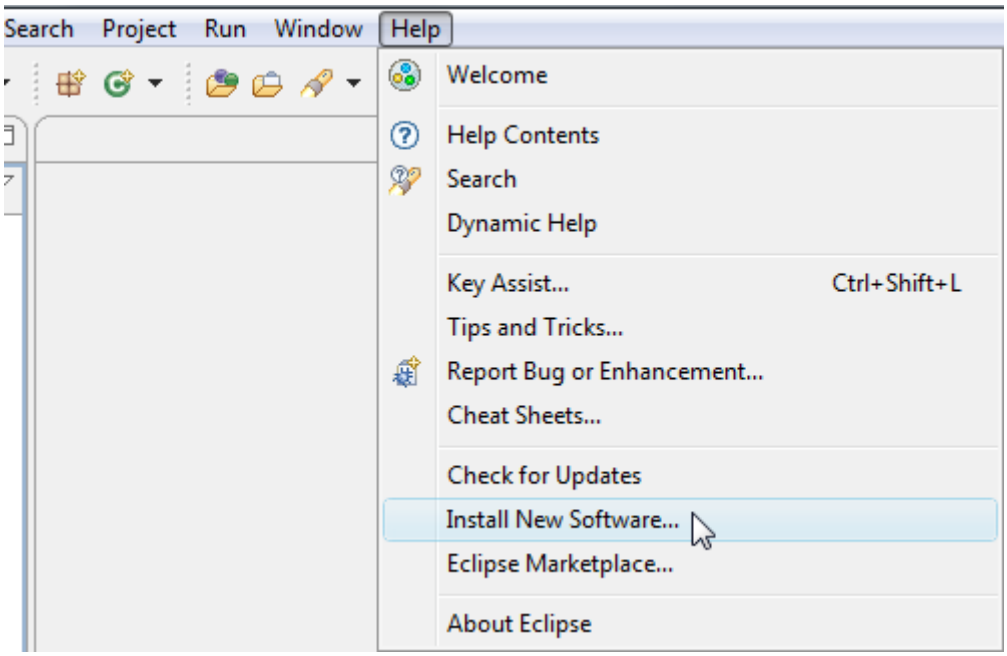




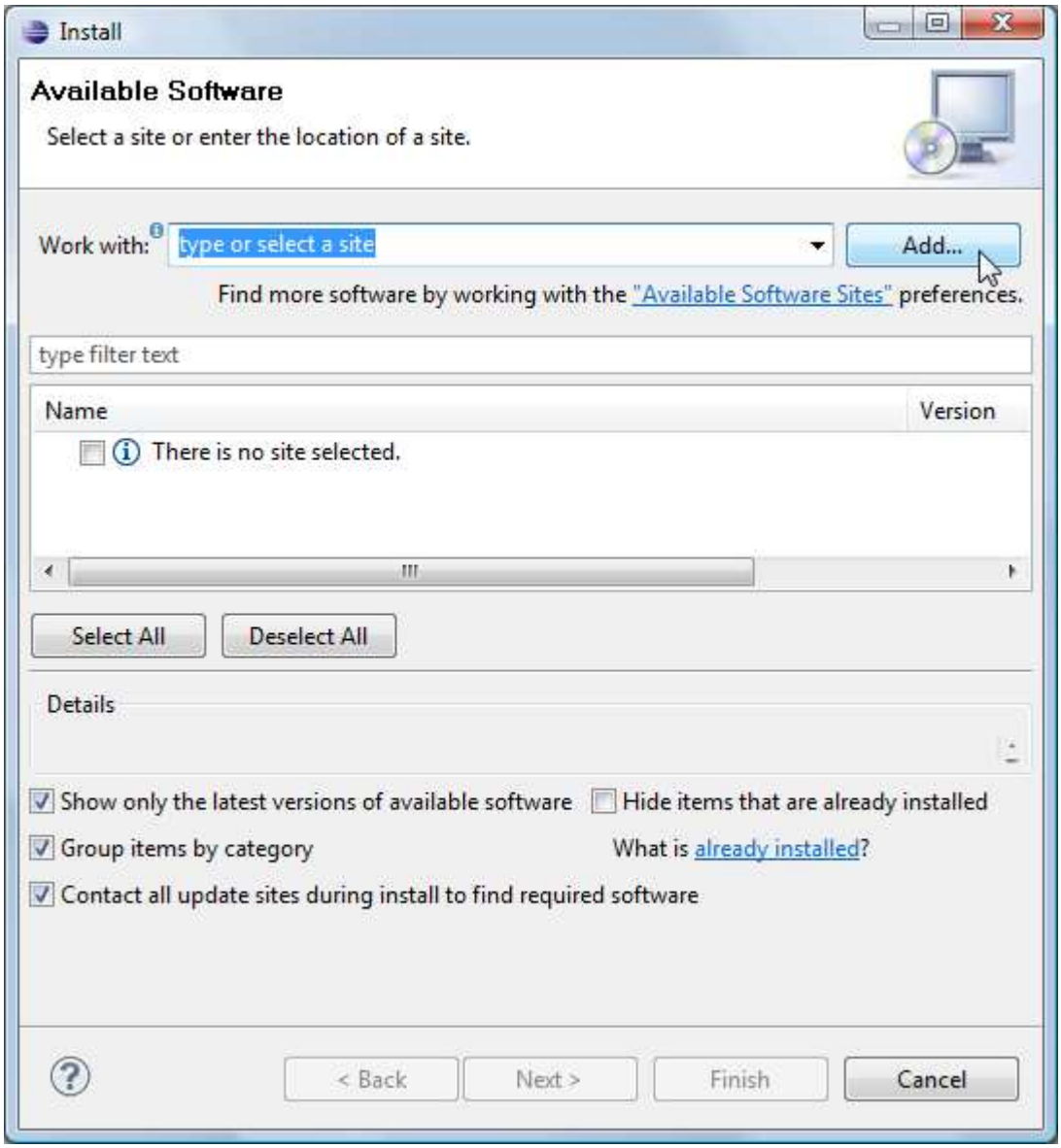
8 Installation von Plugins für Eclipse

Formal gesehen ist Eclipse nur eine Basisplattform, die die einfache Zusammenarbeit verschiedener Werkzeug unter Nutzung einer Komponententechnologie (OSGi) ermöglicht. Eclipse kann mit verschiedenen Komponenten bzw. Erweiterungen, aber hier Plugins genannt, von der Eclipse-Web-Seite geladen werden. Generell können dann weitere Plugins nachinstalliert werden. Die Vorgehensweise ist für fast alle Plugins identisch und wird hier kurz vorgestellt. Die alternative Variante, Eclipse durch das direkte Kopieren von bestimmten Dateien in Eclipse-Verzeichnisse wird hier nicht weiter betrachtet, da es nicht immer unterstützt wird und fehleranfälliger ist.

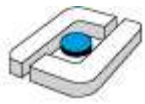
Gestartet wird die Installation über „Help->Install New Software...“.



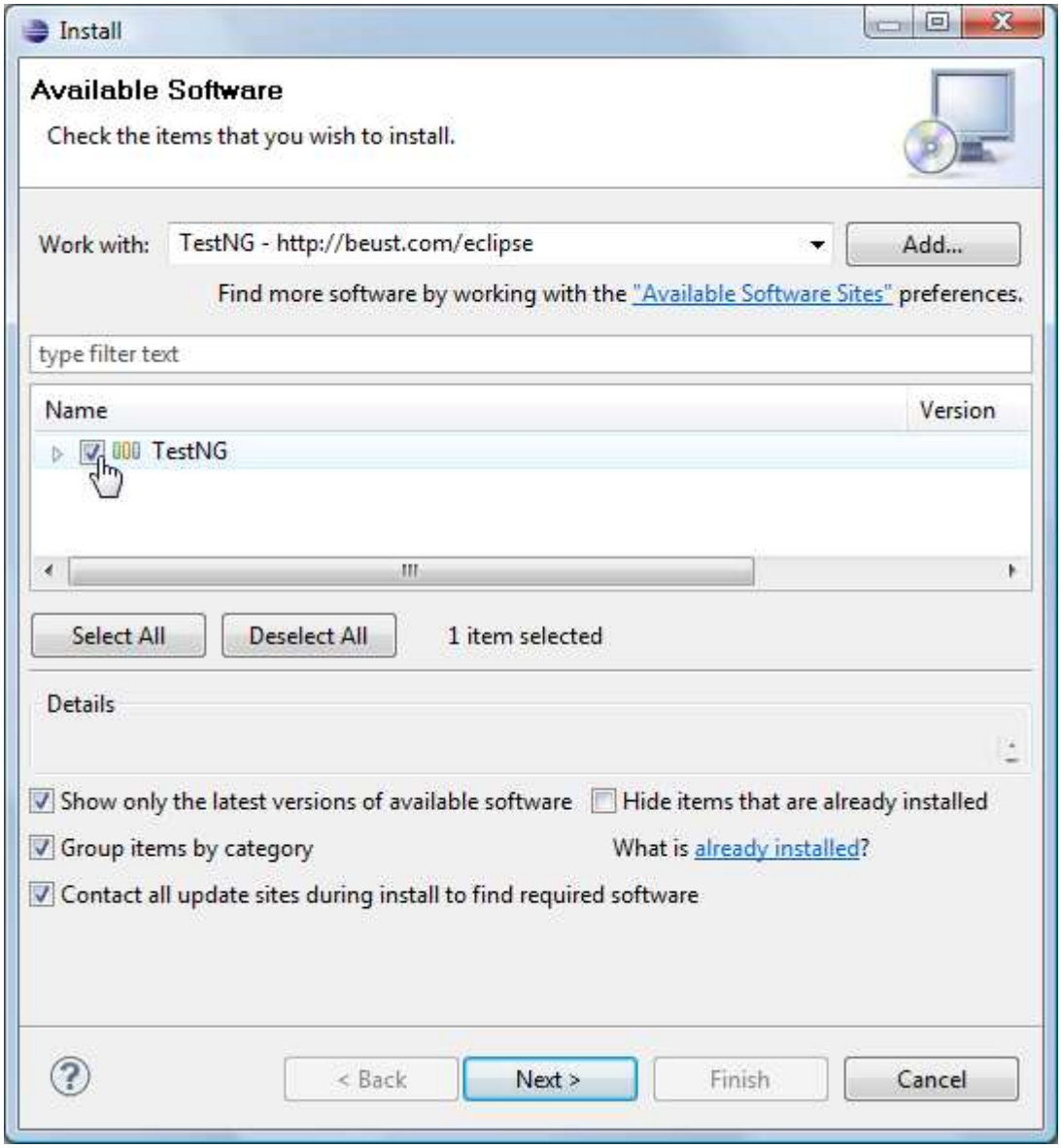
Hier gibt es zwei Möglichkeiten, da bereits einige Web-Seiten existieren, über die Plugins gefunden werden können. Alternativ hat man ein Update-Adresse zur Verfügung. Um eine solche Adresse zu nutzen, wählt man „Add“ am rechten Rand.



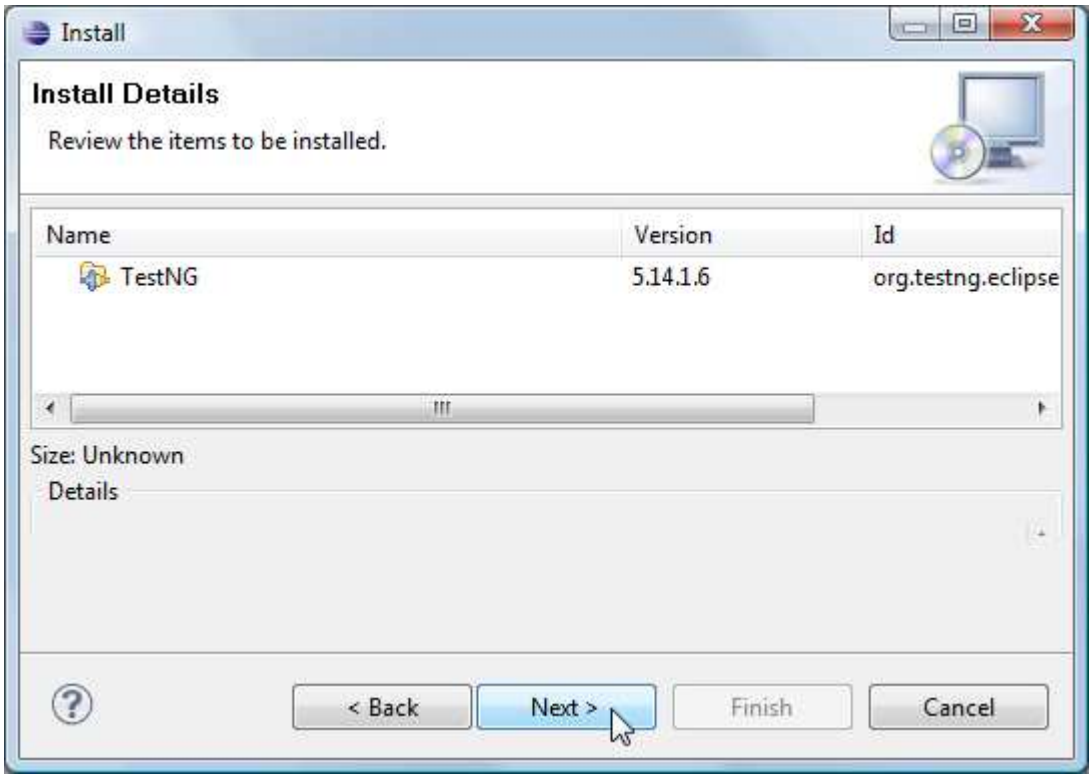
Hier kann man die Web-Adresse mit den Update-Informationen eingeben, der dazu einzugebende Name sollte eindeutig sein und erleichtert die spätere Verwaltung. Der Eintrag wird mit „OK“ abgeschlossen.



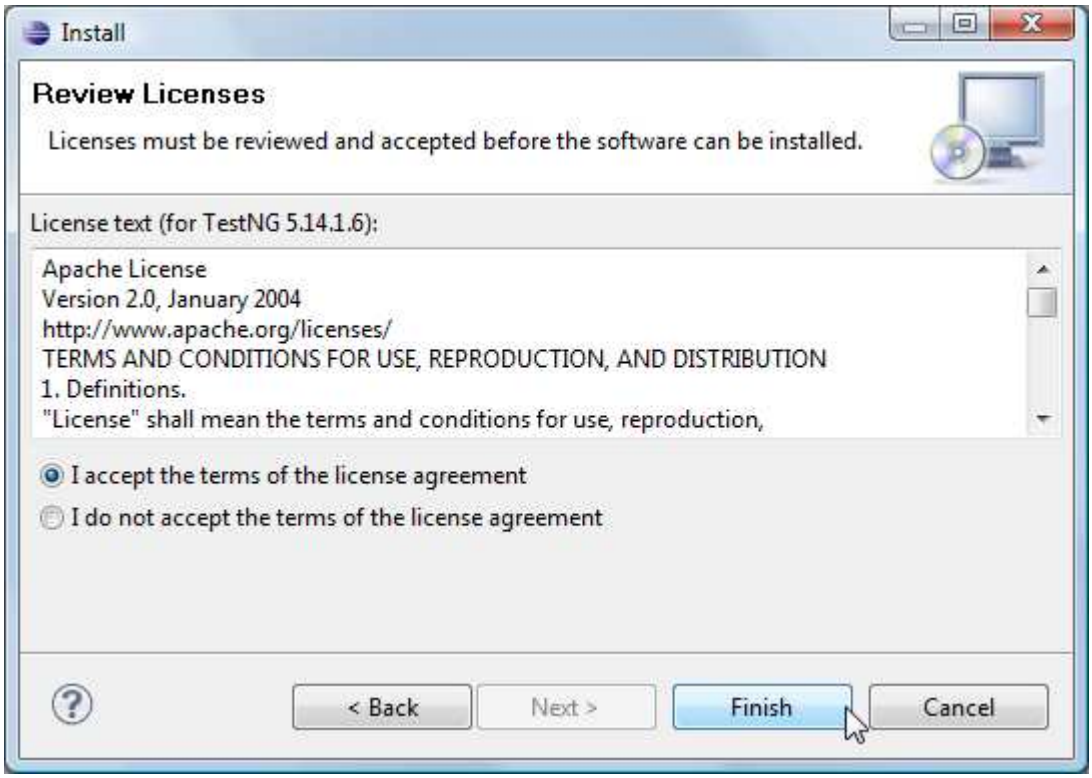
In der Tabelle (Name, Version) werden die gefundenen Erweiterungen angeboten und können mit einem Haken ausgewählt werden. Danach wird „Next>“ gedrückt. Die Schritte danach können bei verschiedenen Plugins etwas variieren, da nach abhängigen Plugins gesucht werden kann, die benötigt werden und Lizenzbedingungen bestätigt werden müssen.



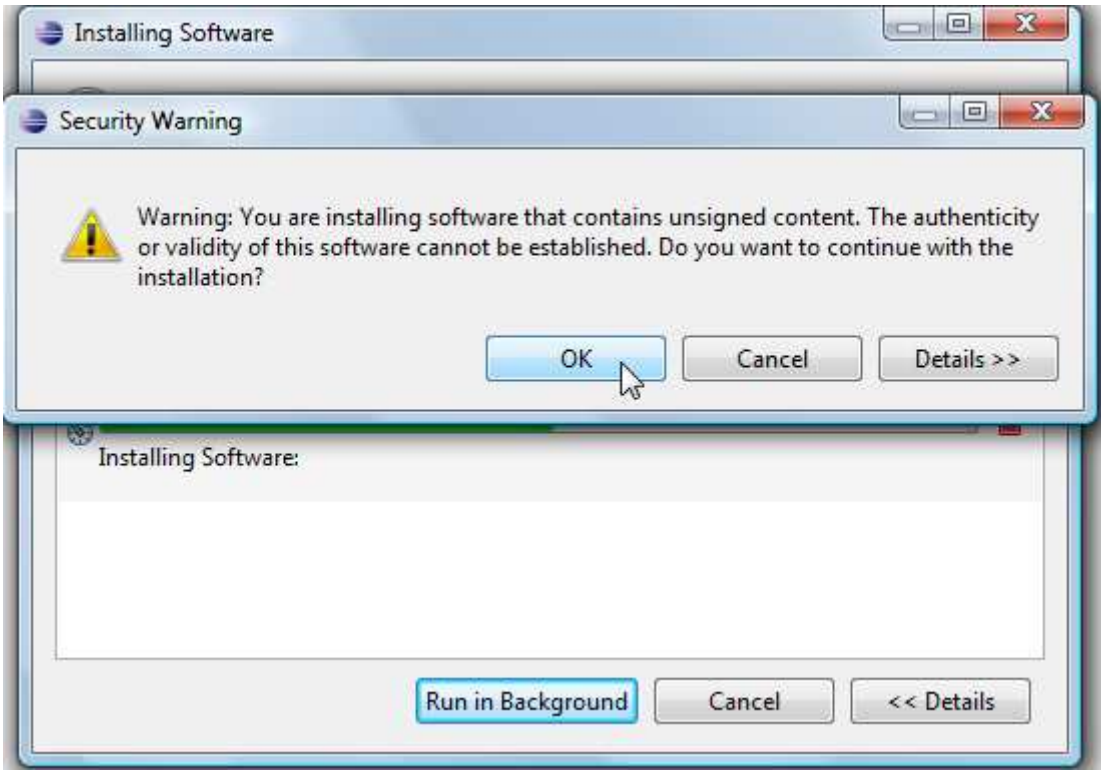
Man erhält eine Übersicht, was installiert werden soll und drückt „Next>“.



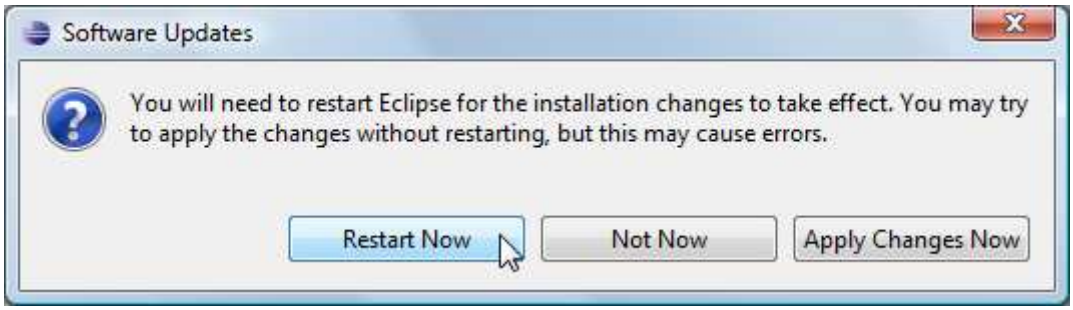
Nun müssen noch die Lizenzbestimmungen gelesen und angenommen und die Installation mit „Finish“ abgeschlossen werden.



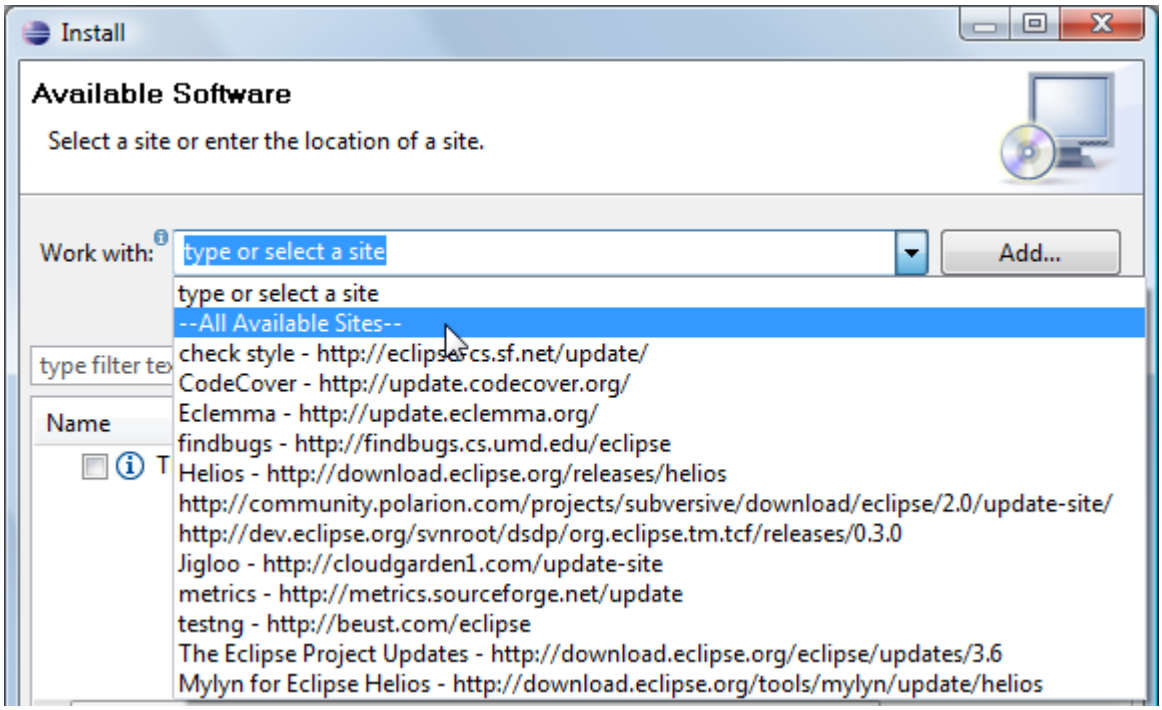
Da eine Signierung relativ teuer ist, wird sie häufiger nicht durchgeführt. Meist sollte man aber die Installation trotzdem mit einem „OK“ fortsetzen.



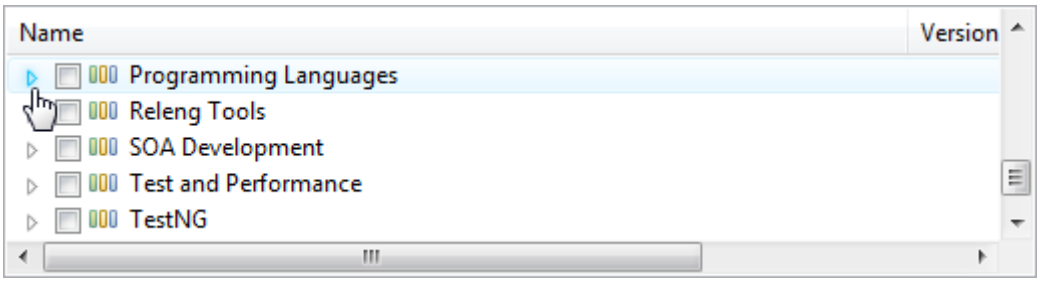
Zum Abschluss der Installation kann man wählen, ob Eclipse neu gestartet werden soll. Generell soll die Installation ohne Neustart möglich sein, trotzdem sollte man einen Neustart durchführen.



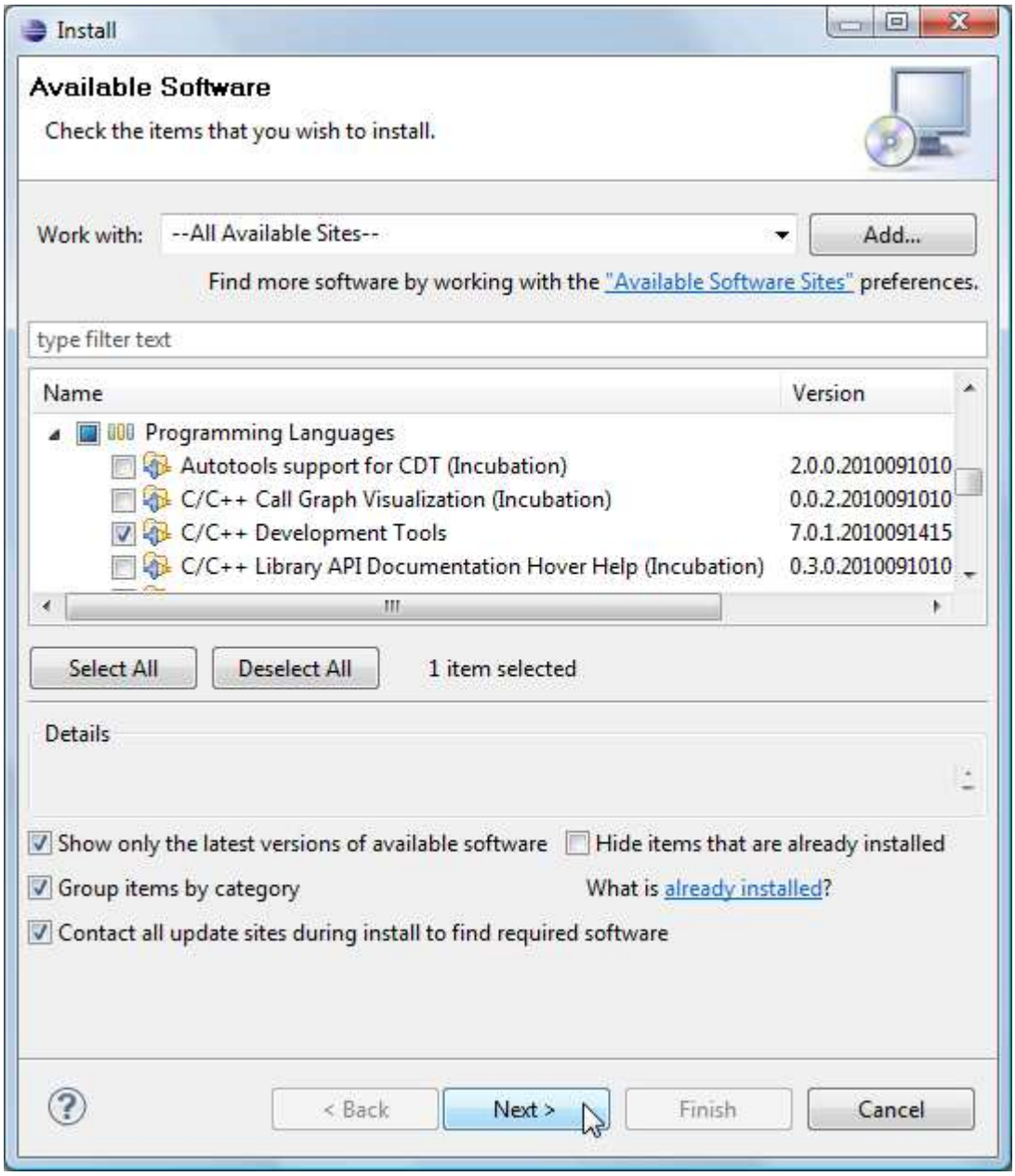
Bereits angedeutet, kann man auch nach Plugins auf bereits vorgegebenen Seiten suchen, was hier ebenfalls gezeigt werden soll. Hierzu wird beim „Install“-Fenster der kleine Pfeil rechts bei der „Work with:“ Box genutzt. Der genaue Inhalt des Fensters hängt von der Eclipse-Version und den bereits vorher genutzten Erweiterungsseiten ab. Im Beispiel wird „--All Available Sites--" ausgewählt.



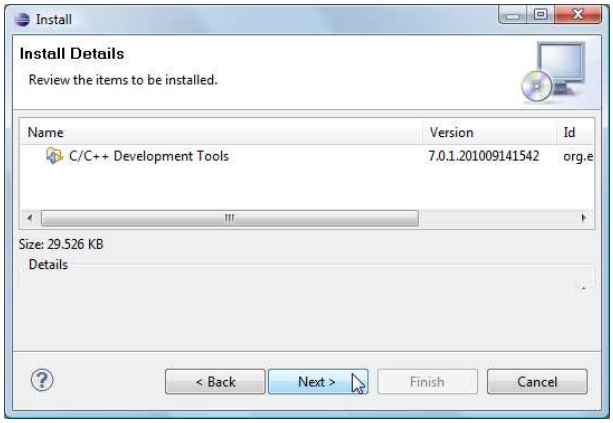
Jetzt ist die Übersicht mit den möglichen Plugins reich gefüllt und kann mit Haken die gewünschten Plugins auswählen. Man beachte, dass man direkt einen Haken setzen oder die Elemente, wie unten angedeutet, weiter aufklappen kann.



Nach der Auswahl, wird zum Abschluss wird „Next>“ gedrückt.



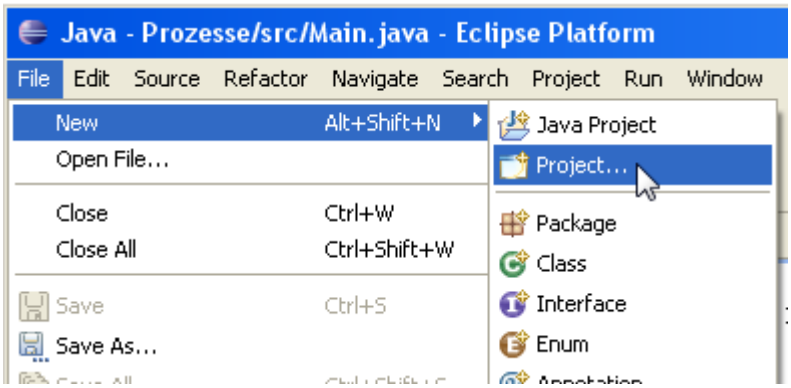
Der weitere Ablauf ist dann, wie bei der vorherigen Erweiterung beschrieben.



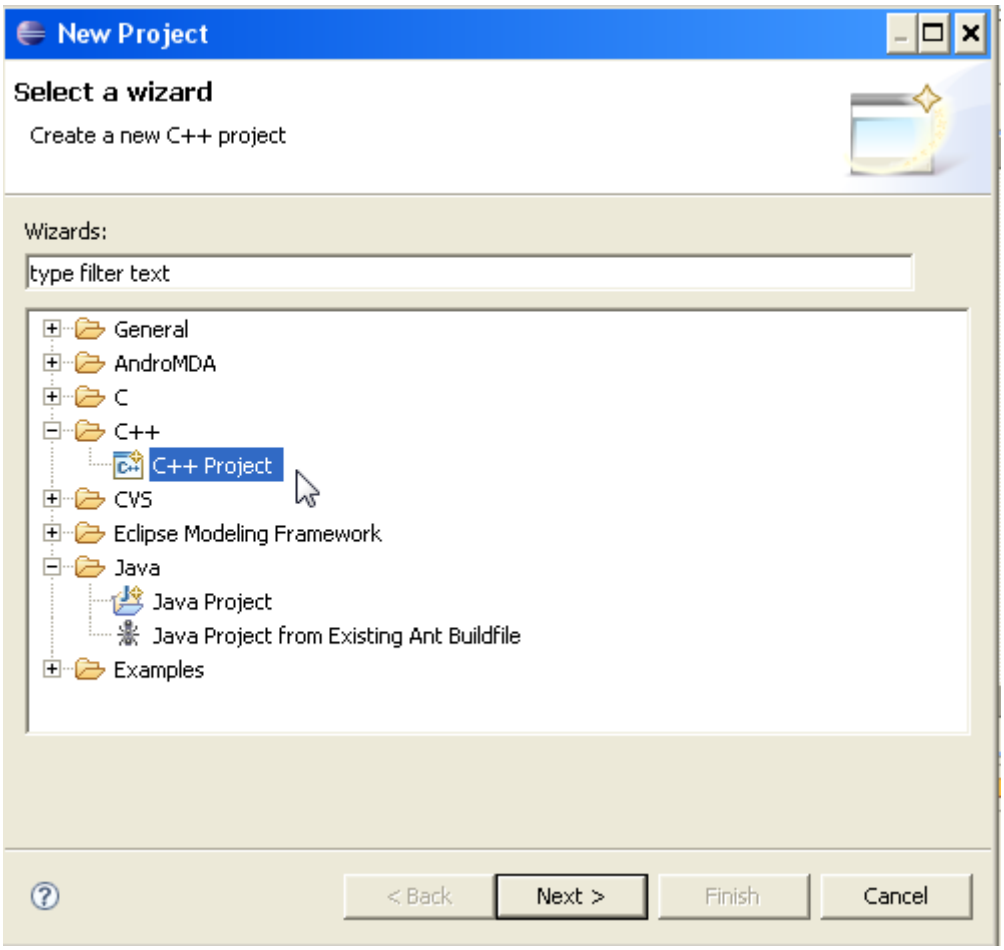
9 Kurzer Einblick in die C++-Entwicklung

Gerade ab der Version 3.3 hat Eclipse einen wesentlichen Sprung zur vollwertigen C++-Entwicklungsumgebung gemacht, wobei allerdings der Ansatz älterer Versionen ziemlich stark geändert wurde. Zur Nutzung der Umgebung muss ein C++-Compiler installiert sein, dabei wird unter Windows der Gnu-Compiler, der über die UNIX-Emulation Cygwin erhältlich ist, und die Mingw-Installation unterstützt.

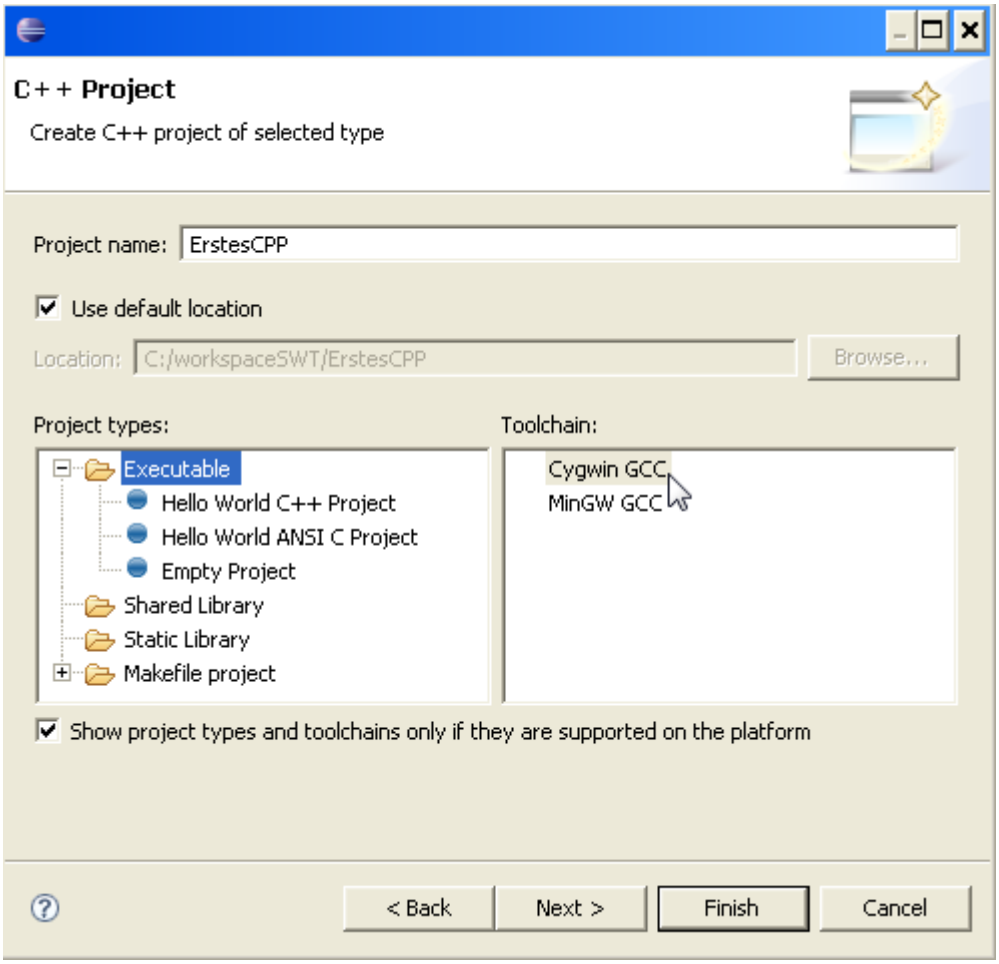
Zunächst wird ein neues Projekt angelegt.



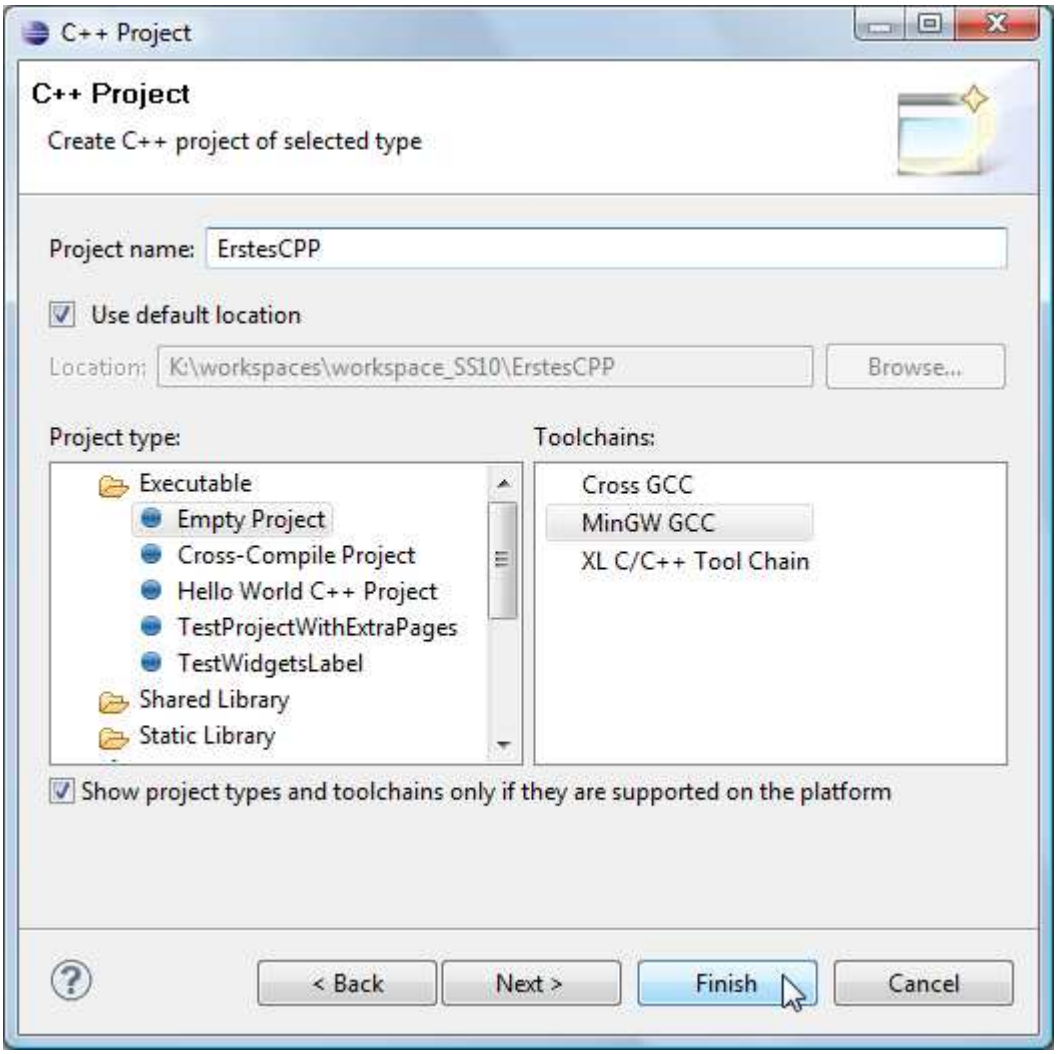
Der Eintrag C++ Project wird gesucht und „Next>“ gedrückt.



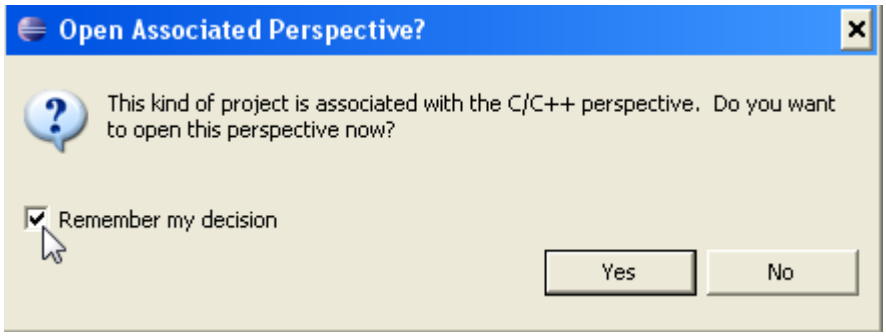
Es bestehen dann verschiedene Möglichkeiten zur Projekteinstellung. Hier wird angenommen, dass Cygwin ordnungsgemäß installiert ist und man das make-File (hoffentlich nur aus Bequemlichkeit) nicht selber schreiben möchte. In diesem Fall werden die folgenden Einstellungen gewählt und Finish gedrückt.



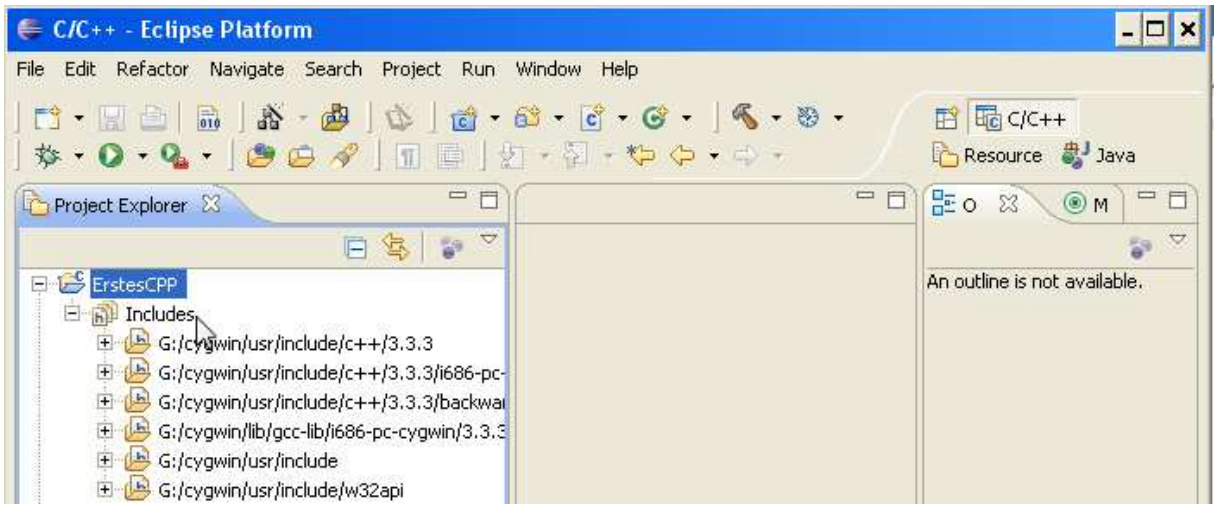
Sollte MinGW installiert sein, sieht die Auswahl wie im folgenden Fenster gezeigt, ähnlich aus.



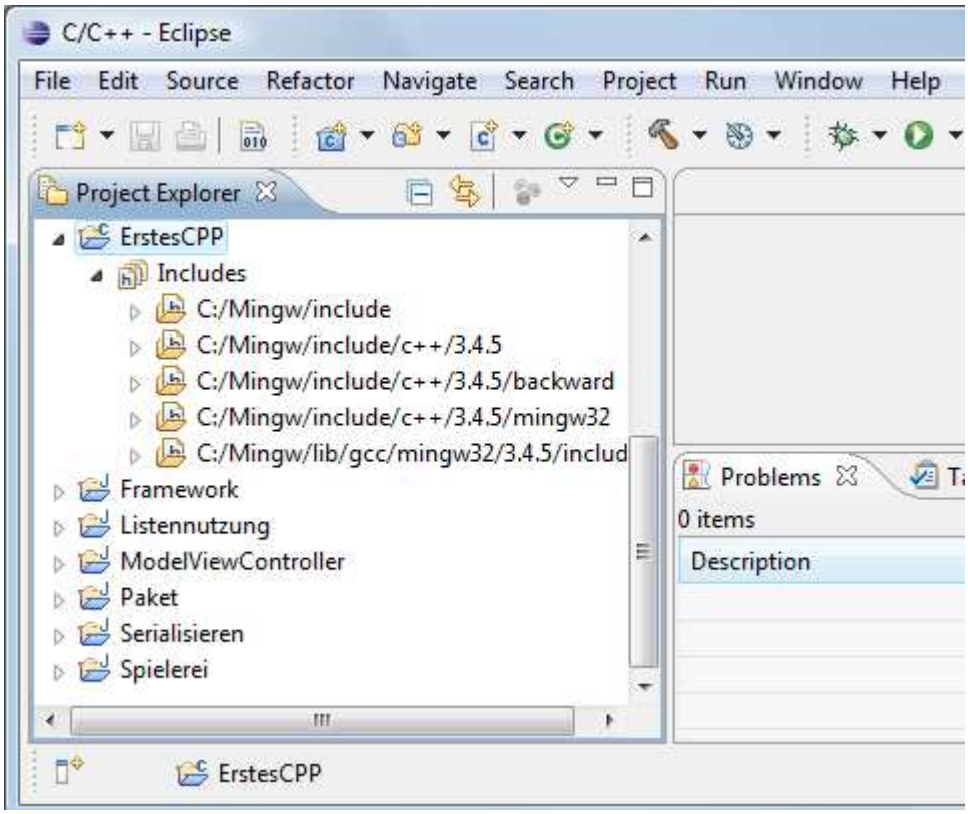
Falls die folgende Meldung kommt, ist diese zu bestätigen und der Haken links unten kann gesetzt werden.



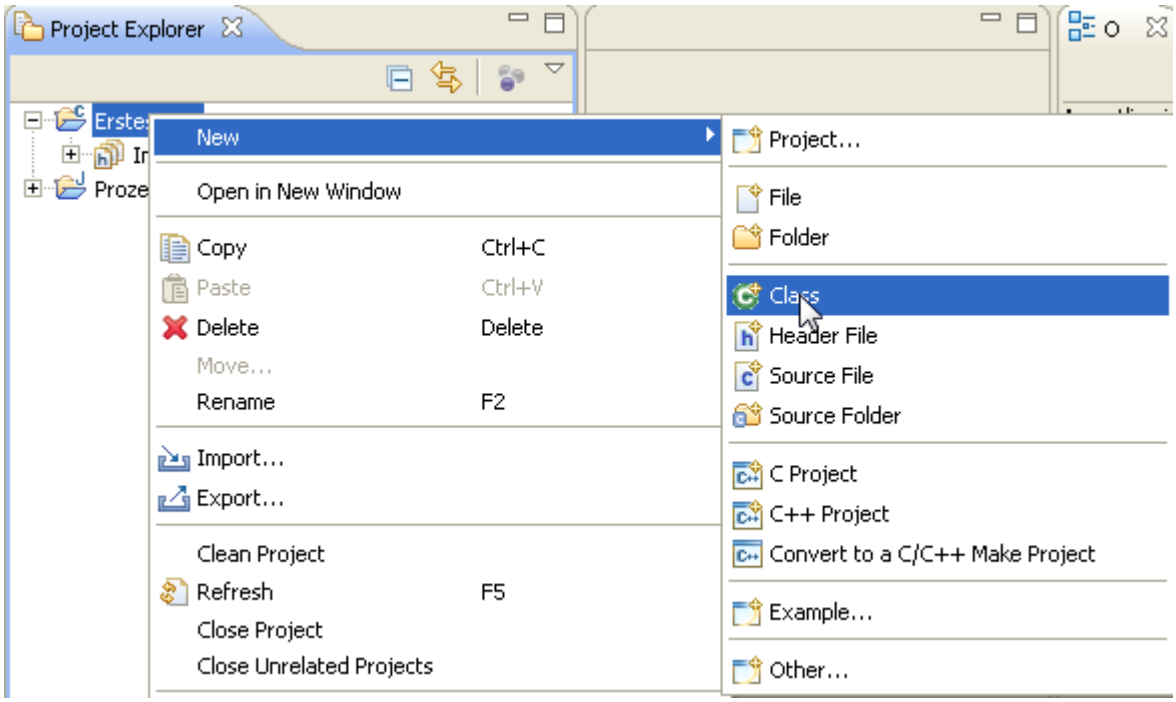
Man erkennt, dass ein Projekt angelegt wird, das einen speziellen Ordner includes enthält. Wird dieser geöffnet, sieht man, welche Libraries zur Verfügung stehen (im Beispiel sind dies relativ alte Versionen).



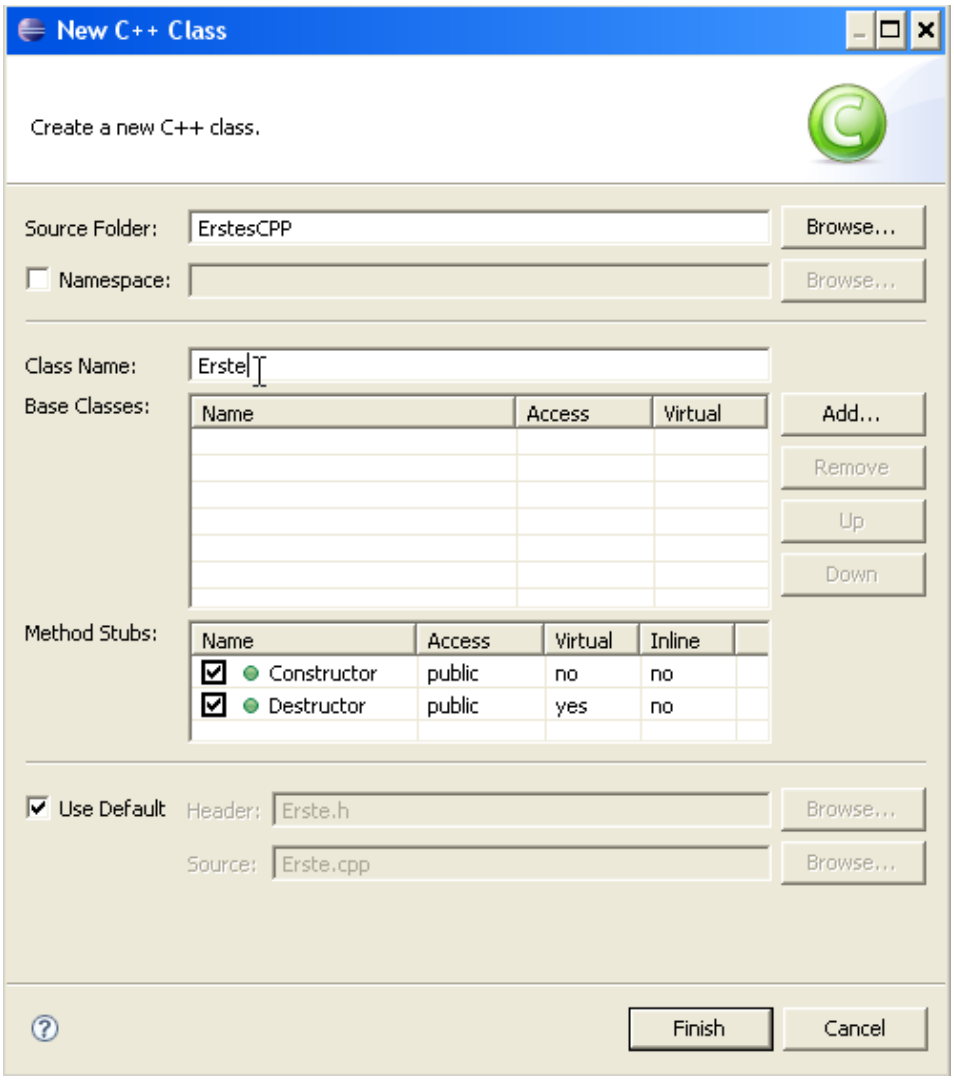
Bei einer MinGW-Installation sieht das Projekt dann wie folgt aus.



Eine Klasse kann z. B. mit einem Rechtsklick auf den Projektnamen angelegt werden.



Im einfachsten Fall reicht es aus, den Klassennamen einzugeben. Man kann hier auch den namespace und die beerbten Klassen angeben. Die Eingabe wird mit „Finish“ beendet.



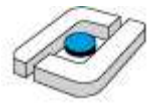
Danach stehen eine Header- und eine Implementierungsdatei zum Programmieren bereit, die bereits im Editor-Feld geöffnet sind, die z. B. wie folgt gefüllt sein können (objektorientiert sinnlos, aber um Funktionalität zu zeigen).

Erste.h

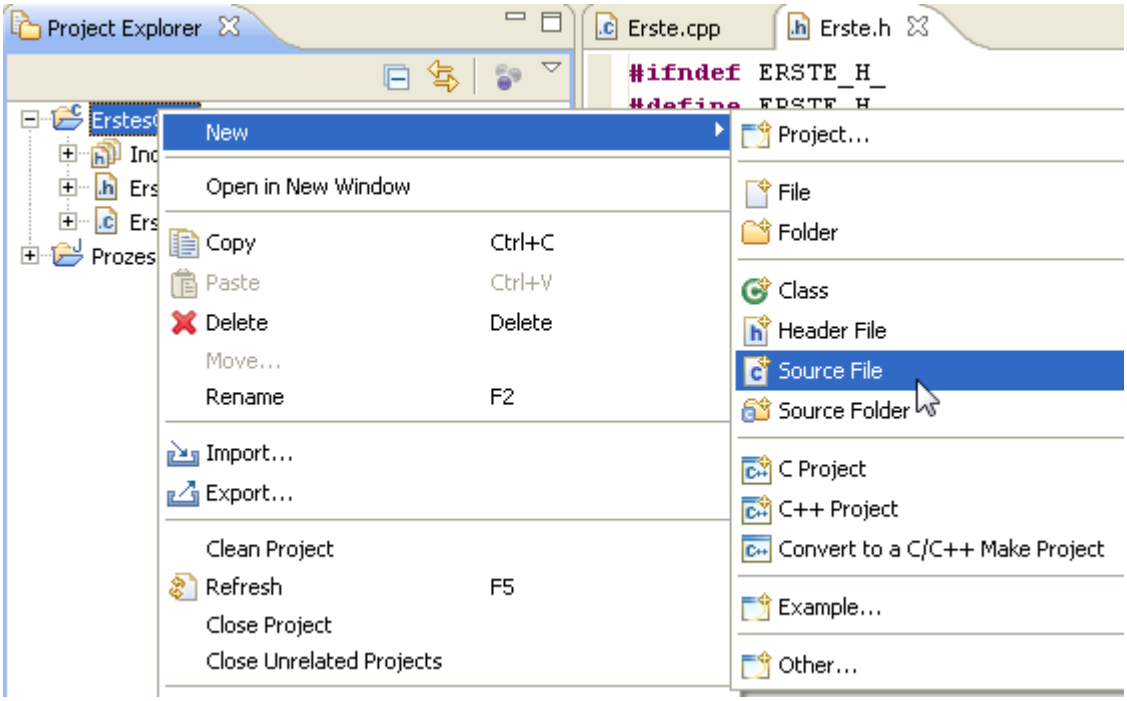
```
#ifndef ERSTE_H_
#define ERSTE_H_

class Erste
{
private:
    int x;
public:
    Erste(int x);
    virtual ~Erste();
    void zeig();
};

#endif /*ERSTE_H_*/
```

Erste.cpp

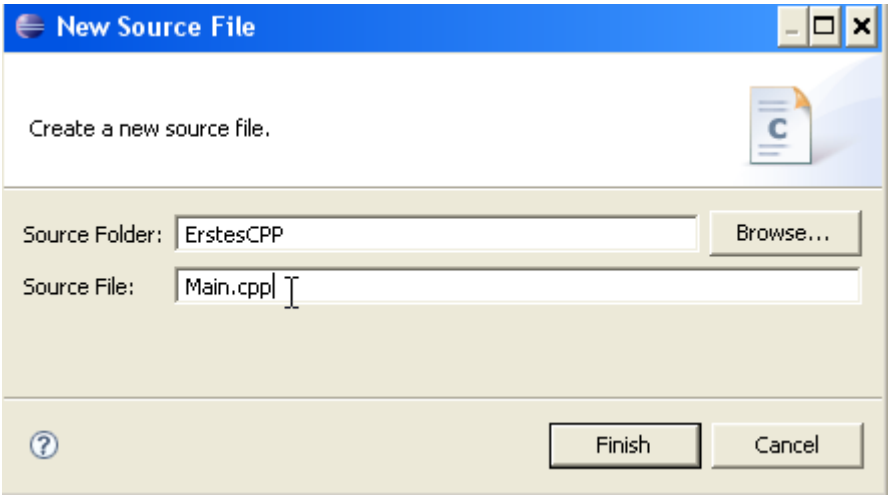
```
#include "Erste.h"
#include<iostream>

Erste::Erste(int x):x(x){}

Erste::~Erste(){}

void Erste::zeig(){
 std::cout<<"Sinn="<<x<<"\n";
}
```



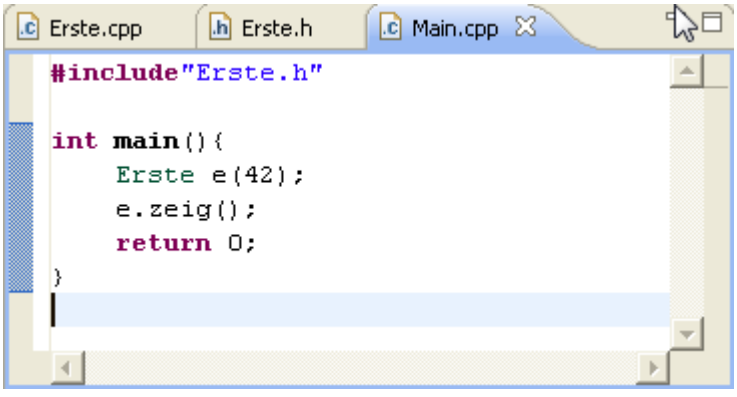
Dann wird eine Datei mit einer Main-Funktion als Source-File angelegt.



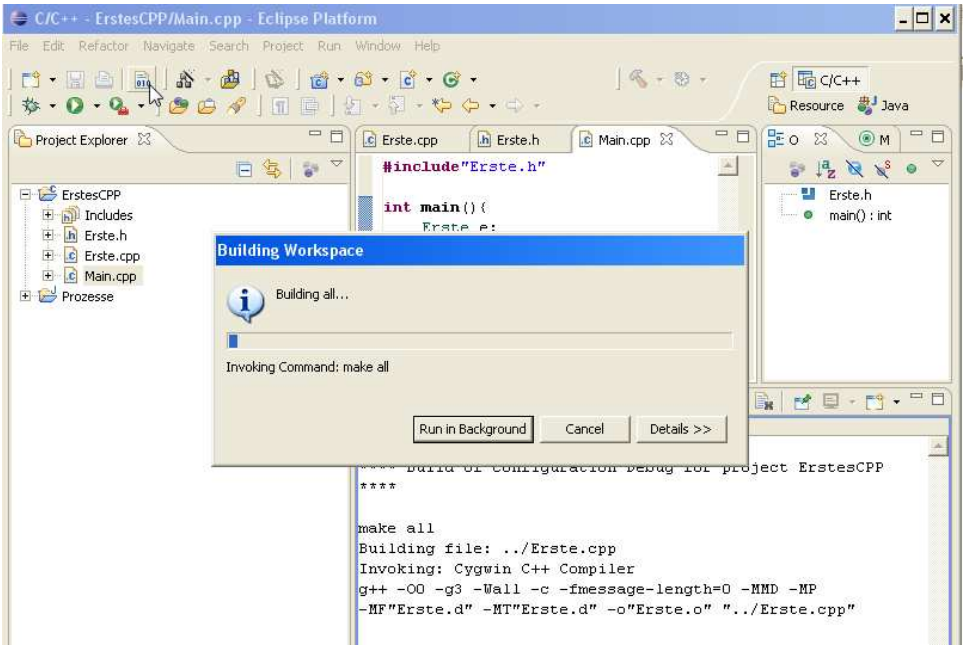
Es muss nur der vollständige Dateiname angegeben werden.



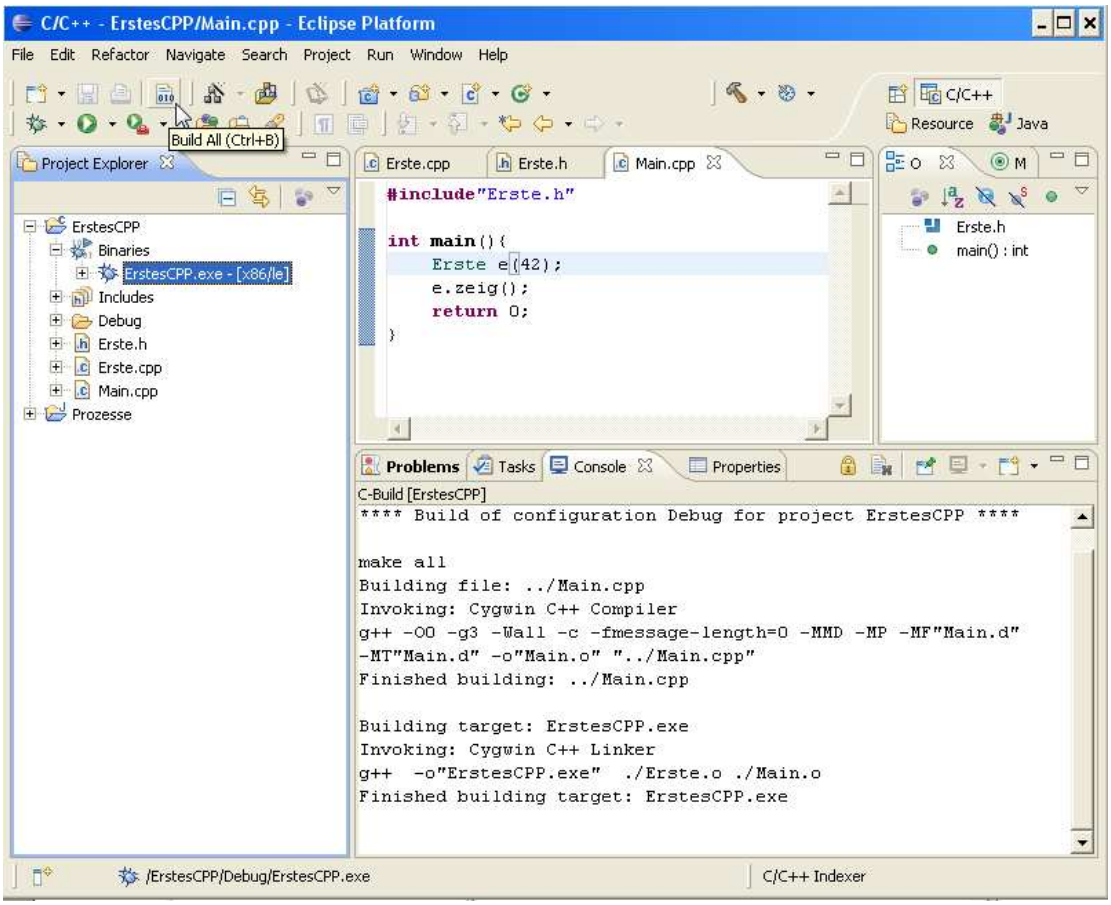
Die Beispieldatei sieht wie folgt aus. Man beachte, dass am Ende der Datei nicht mehr nachzuvollziehenden Gründen immer eine Leerzeile stehen muss.

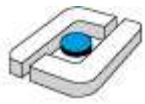


Bis jetzt wurden die Dateien nur angelegt und nicht kompiliert, was theoretisch auch eher möglich gewesen wäre. Zum ersten Kompilieren kann z. B. der mit 010-beschriftete Button genutzt werden. Die Compilermeldungen werden dann unten ausgegeben.

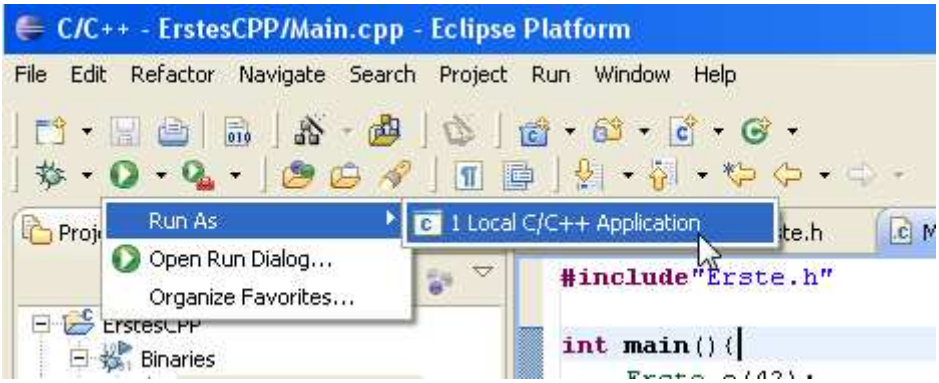


Nach der erfolgreichen Kompilierung ist ein Ordner Binaries entstanden.

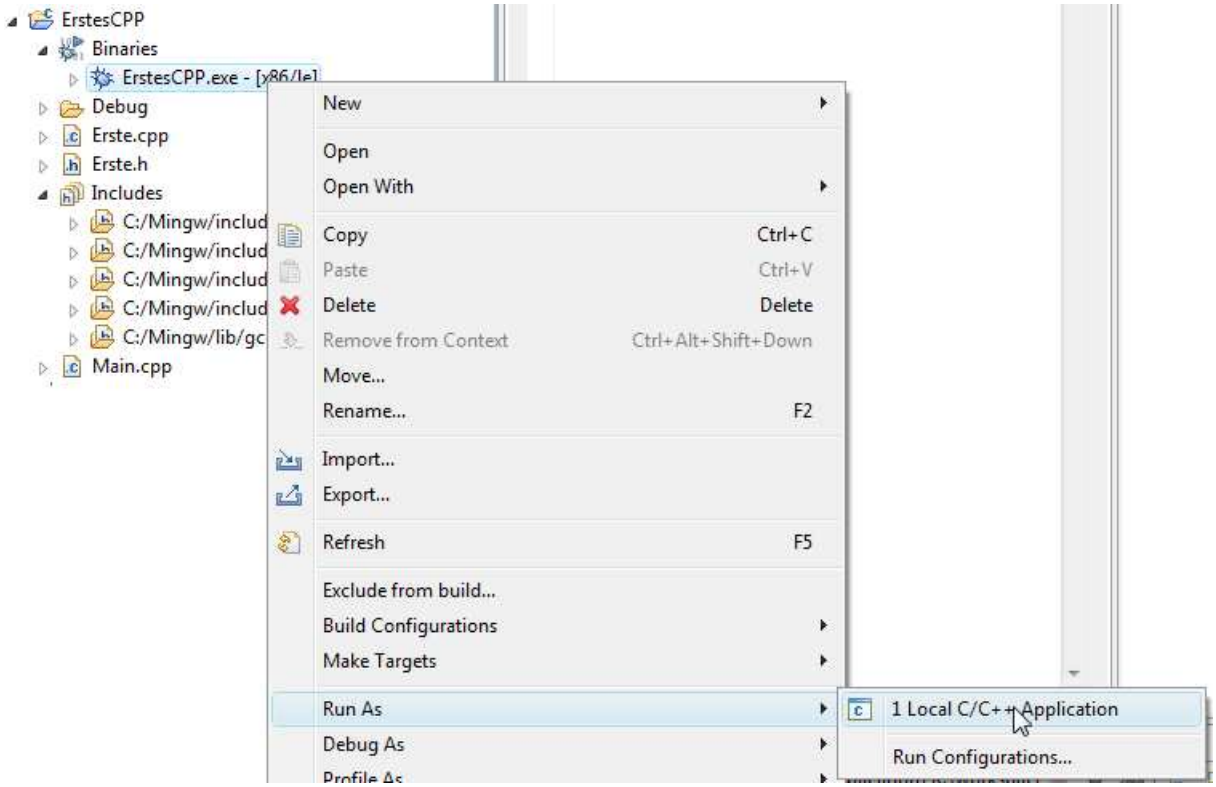




Zur Programmausführung gibt es die Möglichkeit, den kleinen Pfeil rechts neben dem Ausführungspfeil  zu nutzen. Dazu muss aber die Datei mit der main-Funktion um Editorfenster zur Bearbeitung selektiert sein, üblicherweise blau markiert. Dann kann das Programm beim ersten Mal wie folgt gestartet werden.

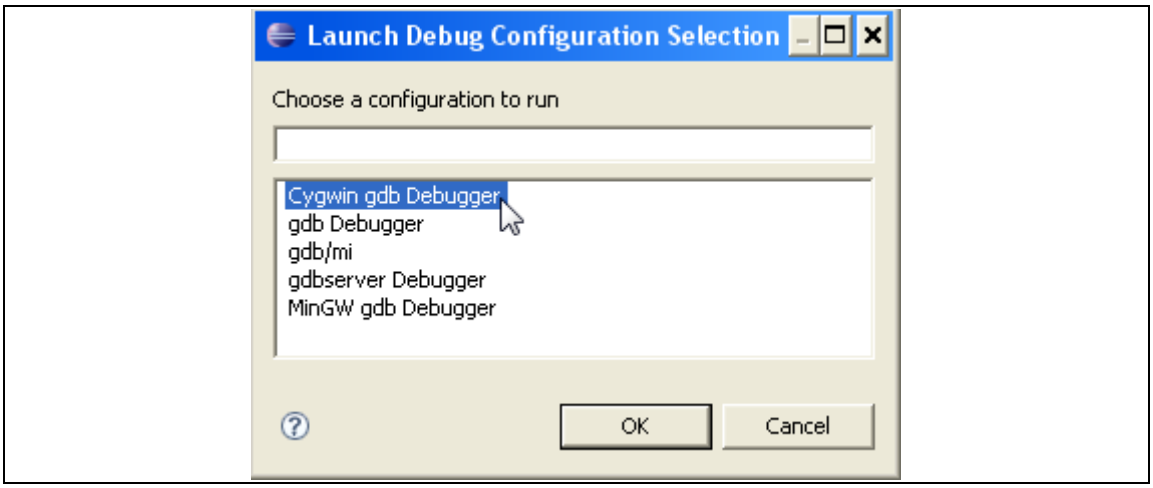


Als Variante kann man das Programm über einen Rechtsklick auf der ausführbaren Datei mit „Run As -> Local C/C++ Application“ starten.

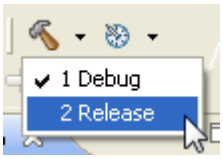


nur für ältere Eclipse-Varianten:

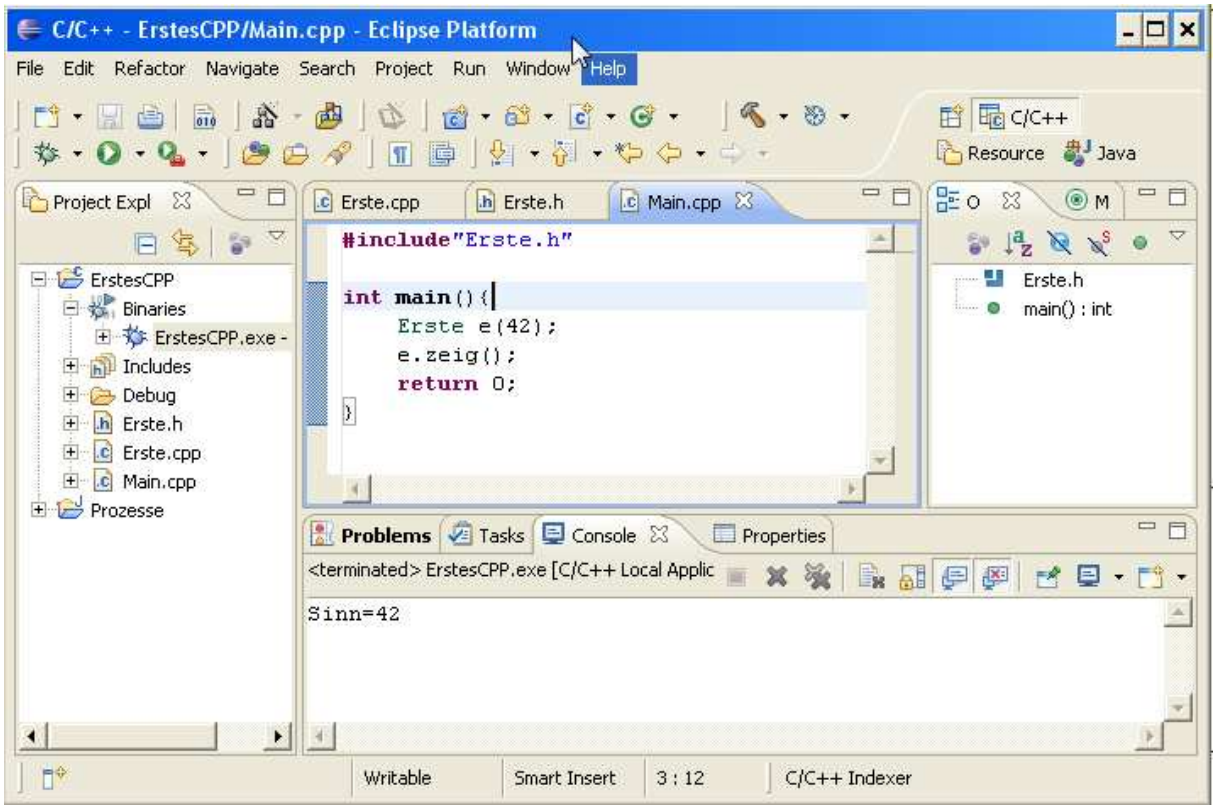
Da Projekte standardmäßig im Debug-Modus laufen, muss jetzt einmal ein Debugger für das Projekt gewählt werden. Da Cygwin genutzt wird, ist natürlich auch dieser Debugger zu nutzen.



Man kann auch auf den Debugger verzichten, wenn man standardmäßig kein Debug- sondern ein Release-Ergebnis erzeugt, was z. B. mit dem kleinen Pfeil nach unten neben folgendem Hammer-Icon umgestellt werden kann.

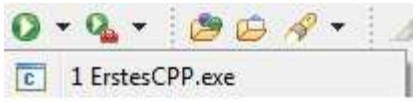


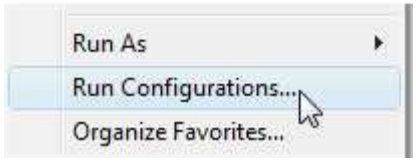
Die Ergebnisse der Programmausführung werden unten angezeigt.



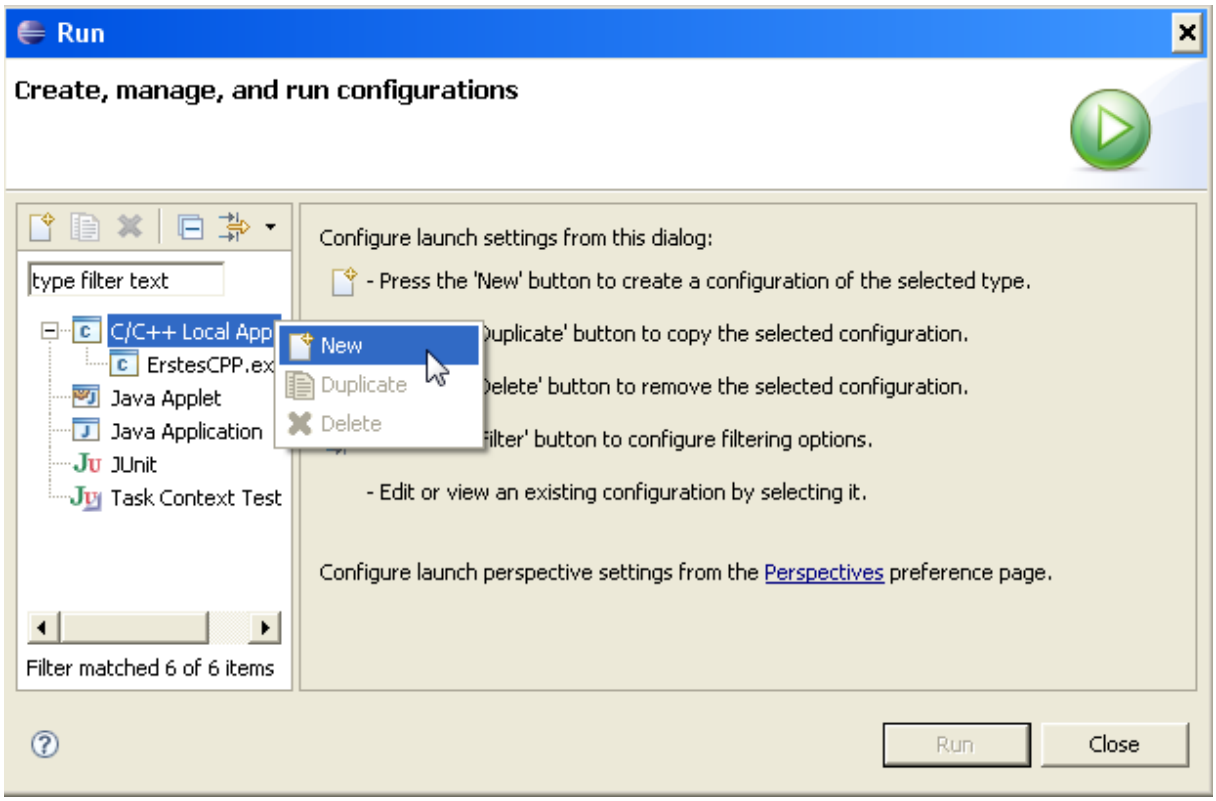
Zur erneuten Ausführung reicht es dann, direkt auf den Ausführungspfeil zu drücken.

Sind weitere Einstellungen zu machen oder treten Probleme auf, kann „Run Configurations...“ aus den Ausführungsmöglichkeiten genutzt werden.

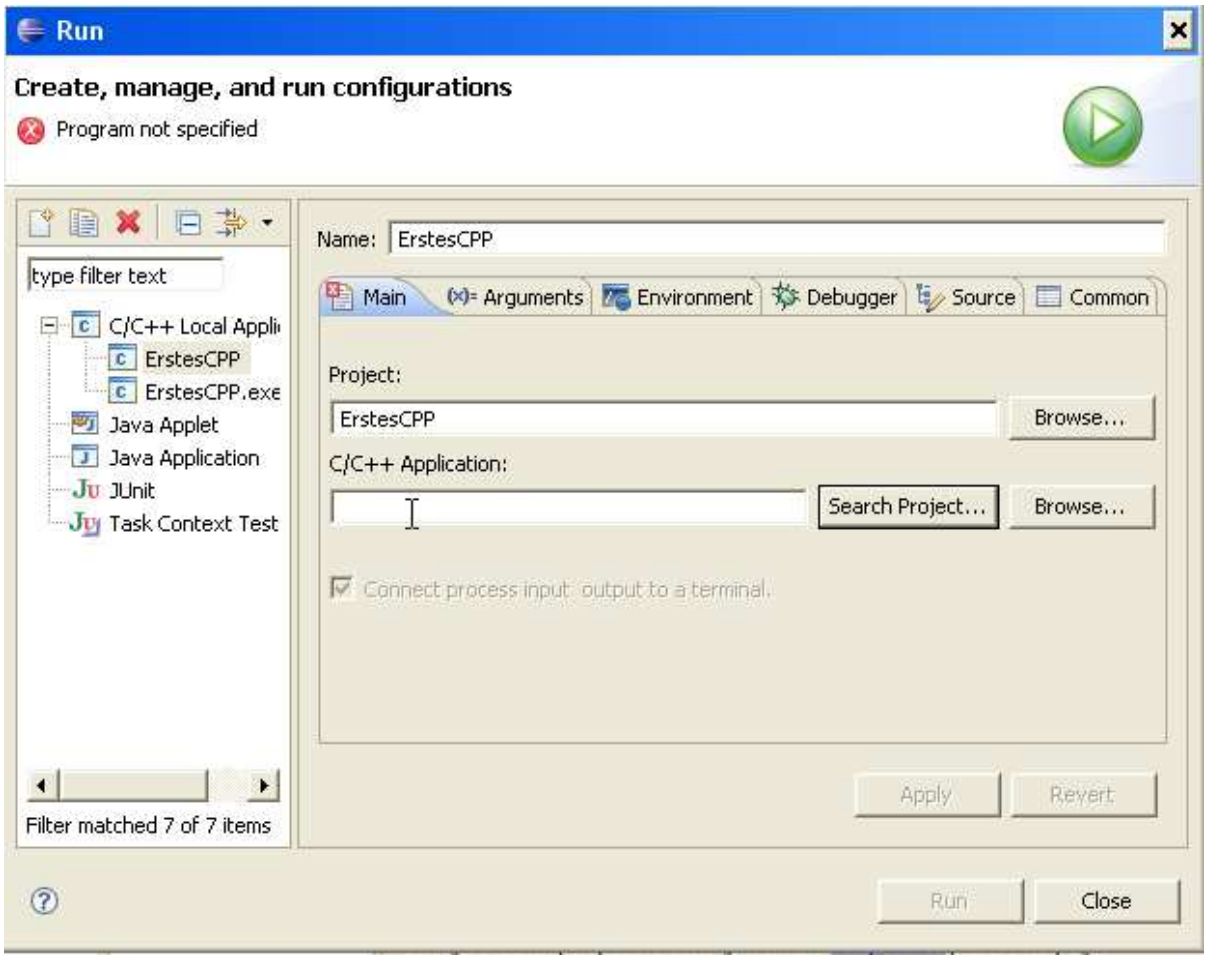




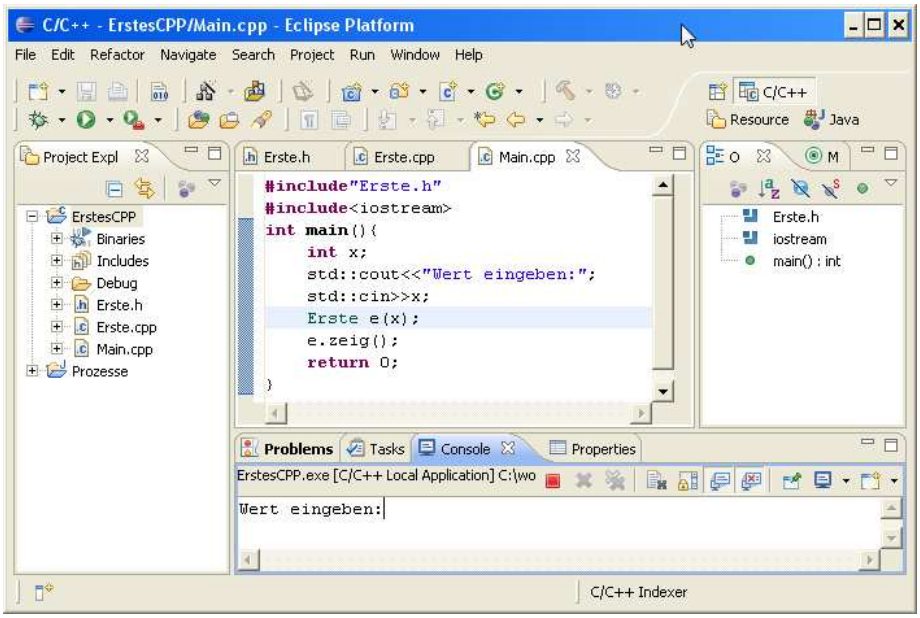
Eine neue Konfiguration wird dann durch einen Rechtsklick auf „C/C++ Local Application“ und „New“ erzeugt.

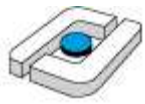


Hier kann u. a. der Name der Konfiguration angegeben werden. Weiterhin muss unter „C/C++ Application“ eine ausführbare Datei gesucht werden. Der erweiterte Modus ist gerade dann sinnvoll, wenn man Werte eingeben möchte, die eigentlich über die Kommandozeile eingegeben werden. Hierfür steht der Reiter „Arguments“ zur Verfügung.

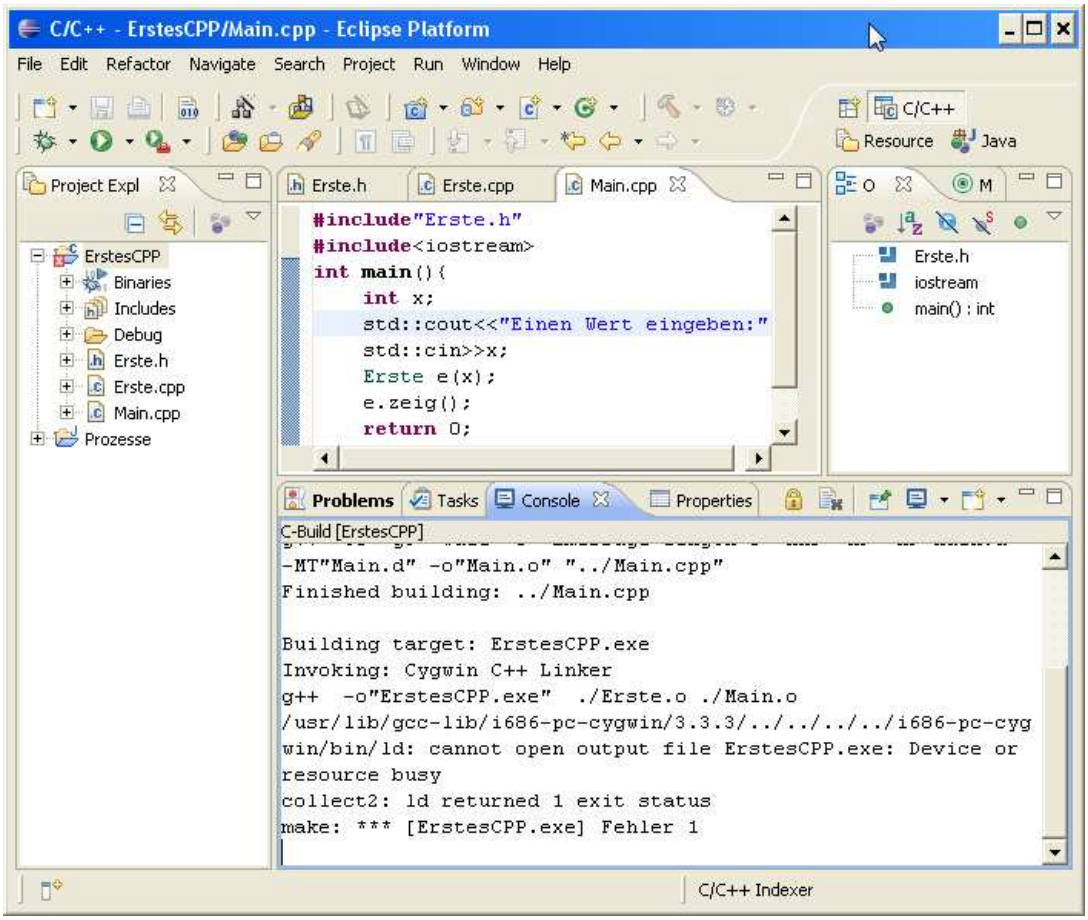


Anders als in Java kann es Probleme geben, wenn man vergisst, ein Programm zu terminieren und dieses erneut kompiliert. Als Beispiel wird folgende Variante der Main-Funktion genutzt, die zunächst eine Eingabe erwartet. Man erkennt, dass unten beim Programm kein <terminated> steht. Eingaben können in diesem Fenster vorgenommen werden, der Cursor wird automatisch an der passenden Stelle platziert.

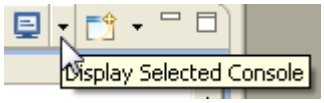




Jetzt kann man während das Programm läuft eine Datei ändern. Drückt man dann erneut auf den Kompilier-Button, erhält man folgendes Ergebnis.

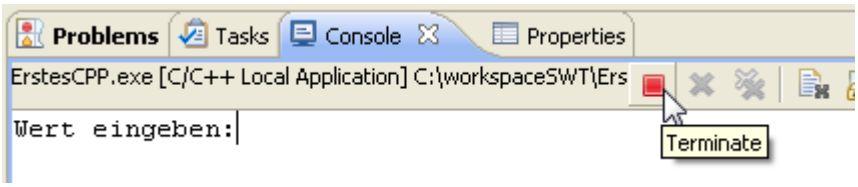


Die Ausgabe bedeutet, dass ErstesCPP.exe nicht erstellt werden kann, da eben dieses Programm noch läuft. Um sich über noch laufende Programme zu informieren und zu diesen zu wechseln, wird der kleine Pfeil in der Kopfzeile des unteren Fensters rechts neben dem Monitorsymbol gedrückt.

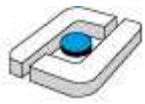


Dann wird zur laufenden Applikation gewechselt.

Diese kann dann u. a. über den roten Knopf im Programmreiter beendet werden.



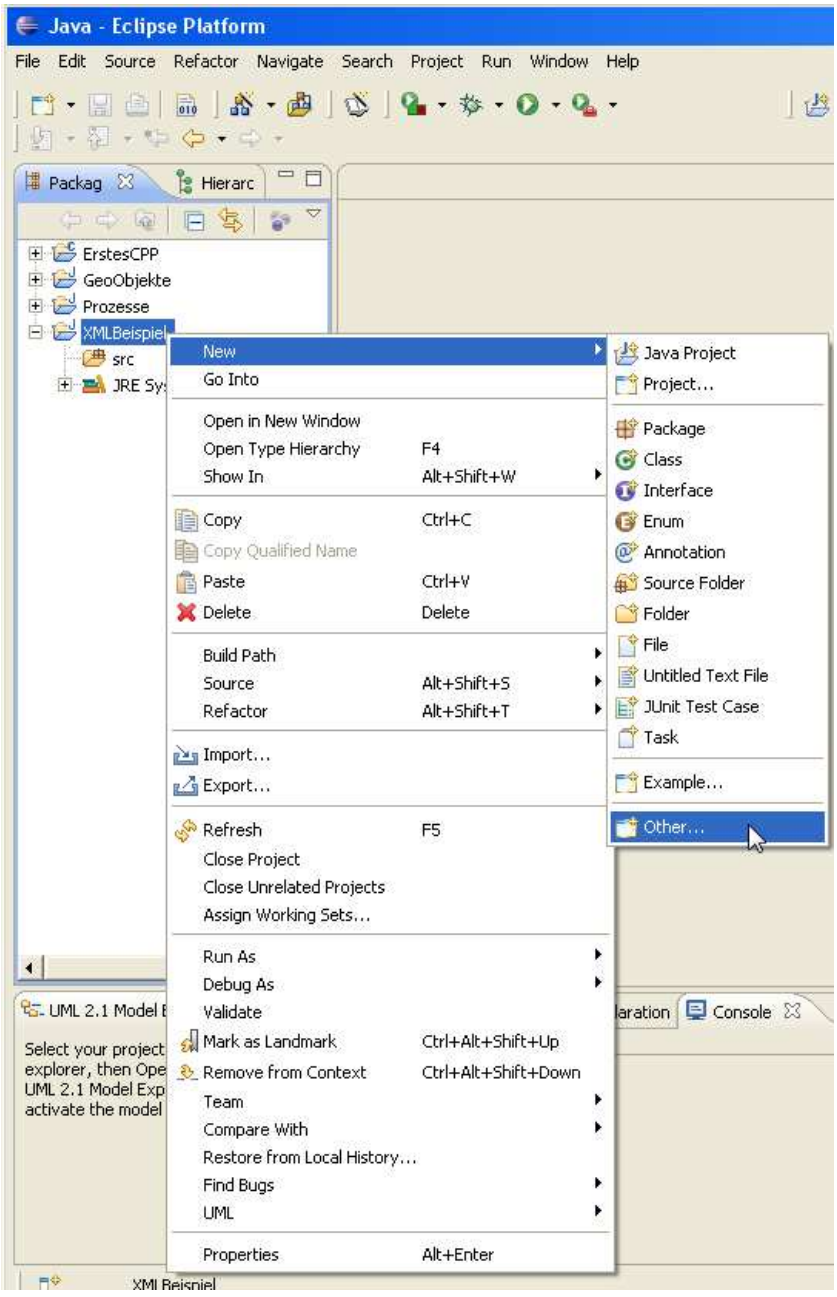
Danach kann erneut kompiliert werden.



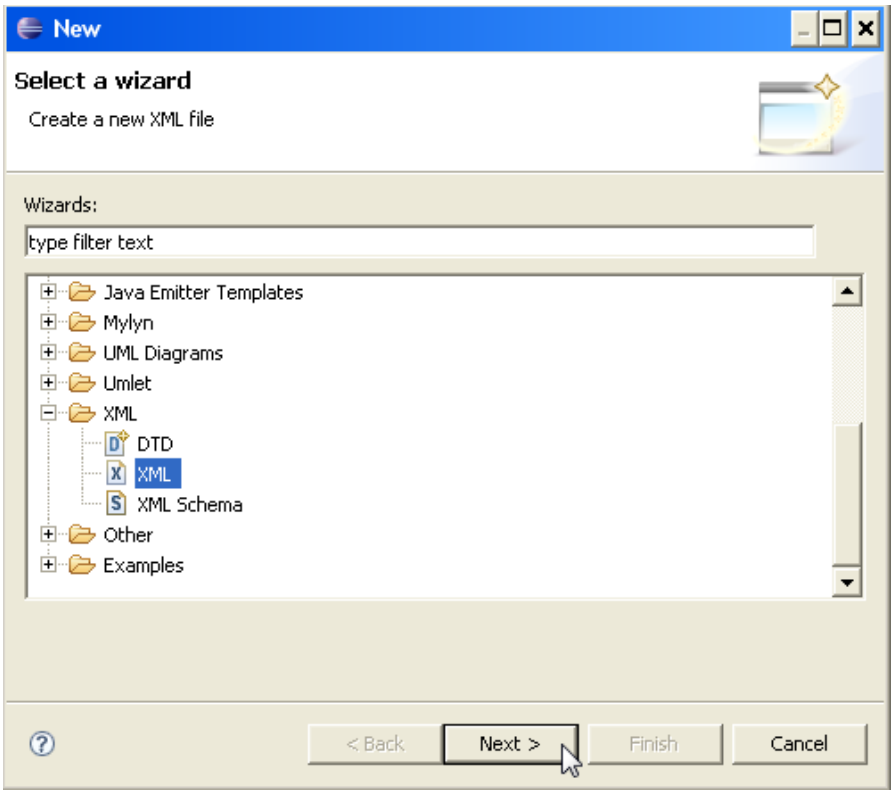
10 XML in Eclipse

Eclipse ermöglicht einen recht einfachen Umgang mit XML, was die Erstellung von XML- und XML-Schema-Dokumenten sowie deren Prüfung auf Wohlgeformtheit und Gültigkeit betrifft. Das Plugin von XML steht in der Java-Entwicklungsversion von Eclipse bereits zur Verfügung, alternativ muss das Plugin für Web-Dienste (sehr groß, wobei XML nur ein Bestandteil ist) installiert werden. Eine nutzbare Installation für Windows finden Sie im Dozentenverzeichnis.

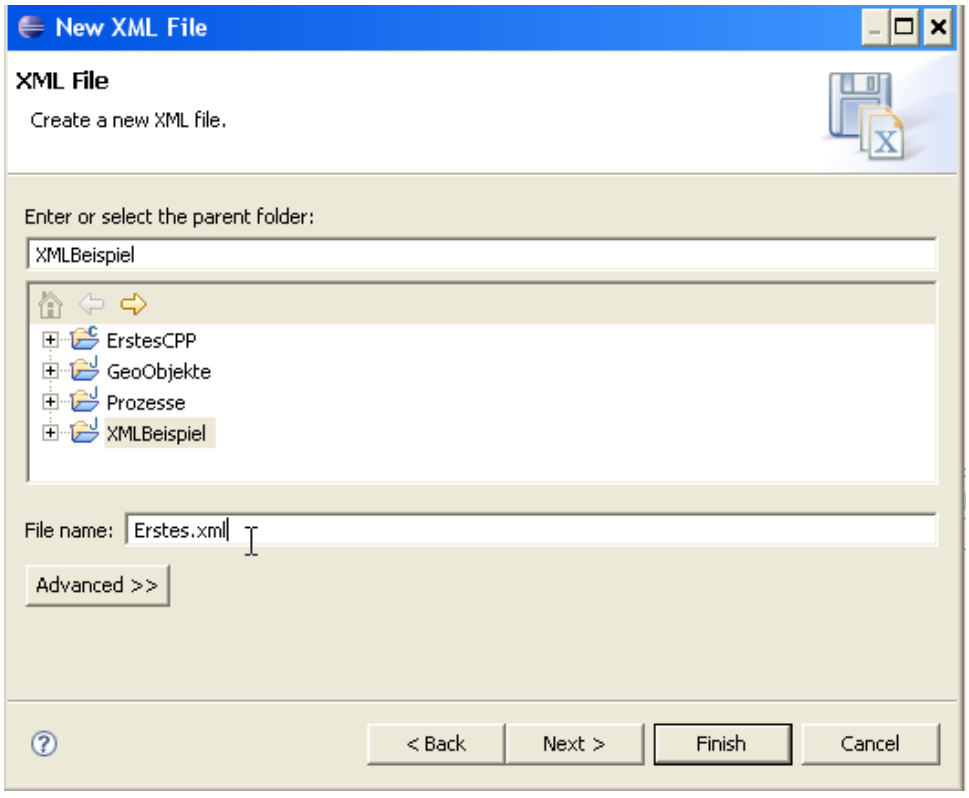
Um mit XML zu arbeiten, muss zunächst ein Java-Projekt angelegt werden (wie es im zugehörigen Kapitel beschrieben ist). In diesem Projekt können dann an beliebigen Stellen XML-Dokumente erstellt werden. Dazu führt man in dem Verzeichnis, das das neue Dokument enthalten soll, z. B. einen Rechtsklick aus, wählt „New“ und dann „Other“.



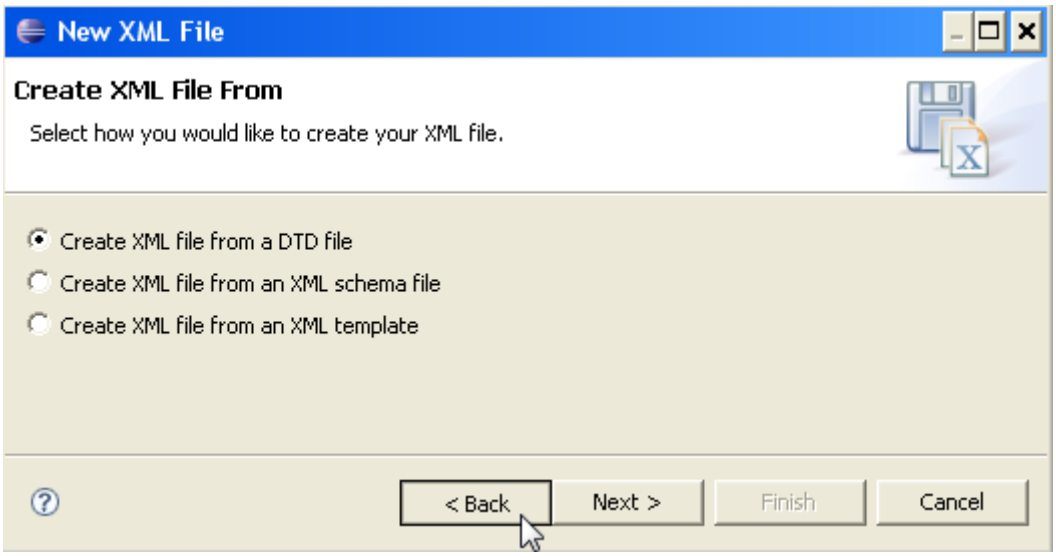
Im folgenden Menü, was in seiner Darstellung stark von den vorhandenen Plugins abhängig ist, wird der Eintrag XML gesucht und „Next>“ gedrückt.



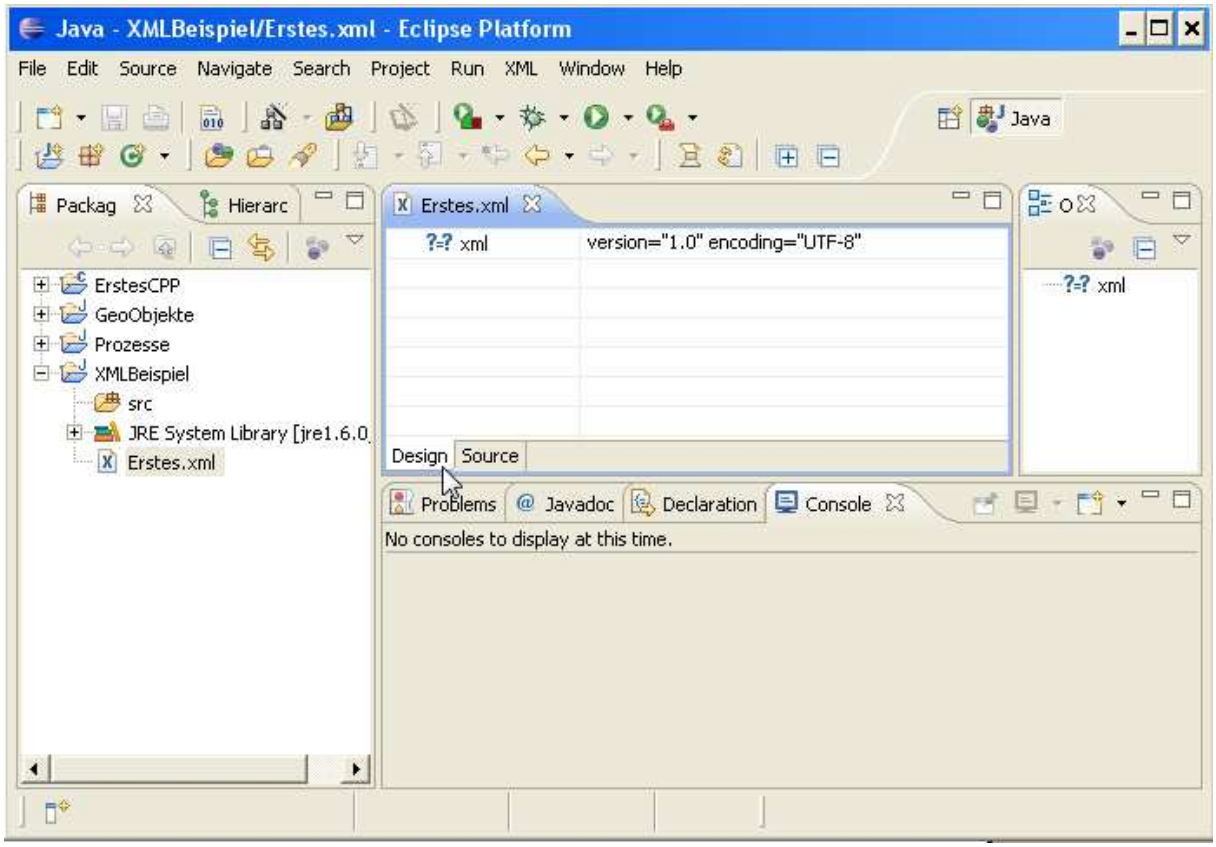
Danach muss der Dateinamen eingegeben werden. Will man nur eine neue XML-Datei erzeugen, muss man jetzt auf „Finish“ drücken. Um die weiteren Möglichkeiten kennen zu lernen, wird auf „Next“ gedrückt.

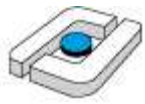


Man hat jetzt die Möglichkeit, festzulegen, auf welcher Basis das XML-Dokument entwickelt werden soll. Nachdem man diese festgelegt hat, kommt man mit „Next>“ zu einem individuellen Dialog, mit dem man die zu nutzende Datei angeben kann. Dieser Weg wird später betrachtet, im Beispiel kehren wir mit „<Back“ zum vorherigen Menü zurück und drücken „Finish“.

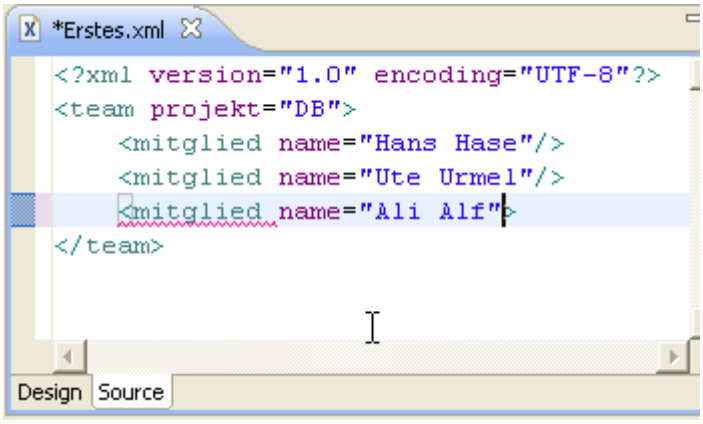


Der XML-Editor besteht aus zwei Varianten. Im Design-Modus wird die Struktur des XML-Dokuments angezeigt. Diese baumartige Anzeige kann u. a. mit Rechtsklicks an der richtigen Stelle bearbeitet werden. Gerade für Anfänger wird aber die einfache Source-Ansicht empfohlen, die hier als Einzige genauer betrachtet wird.

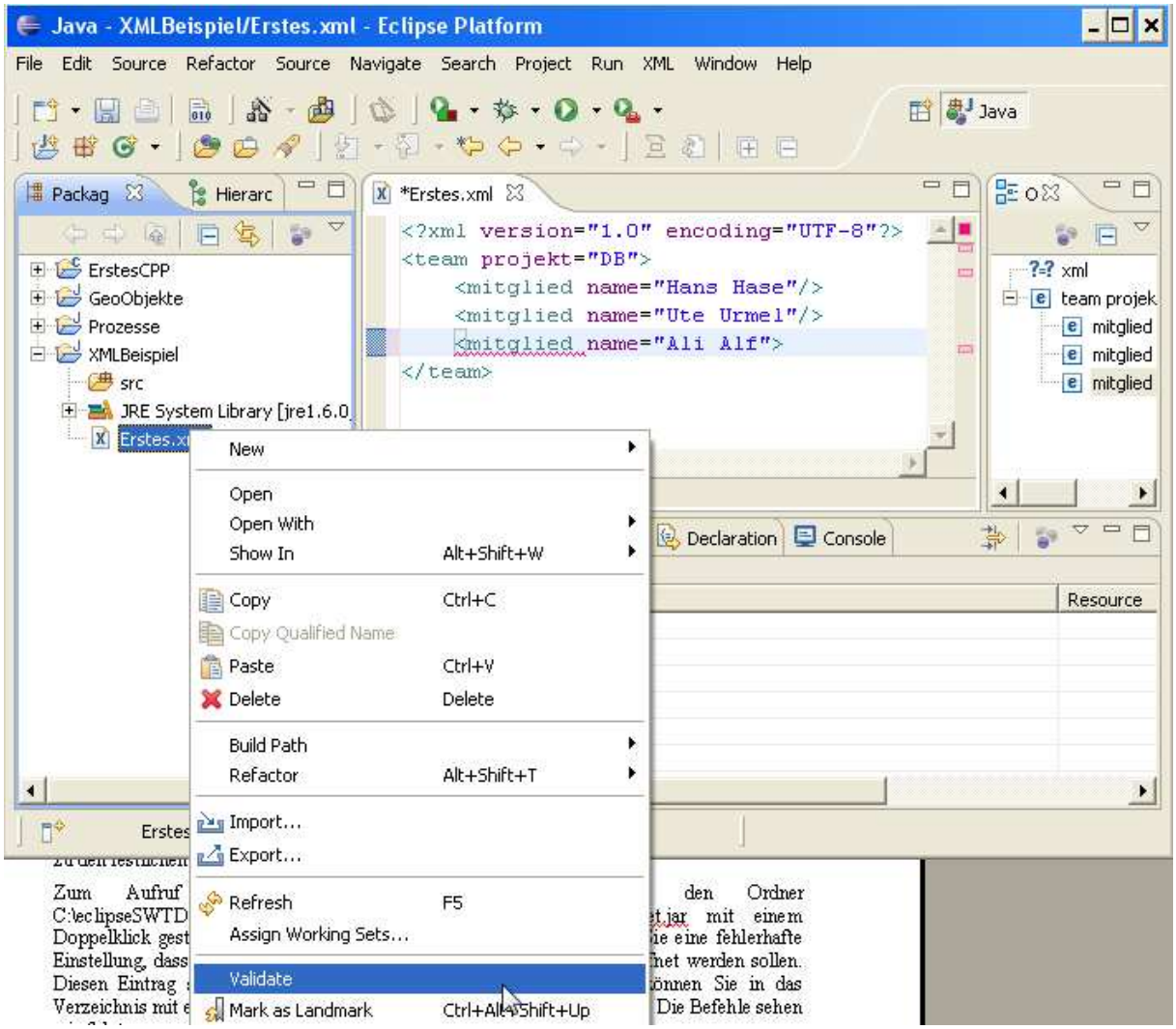




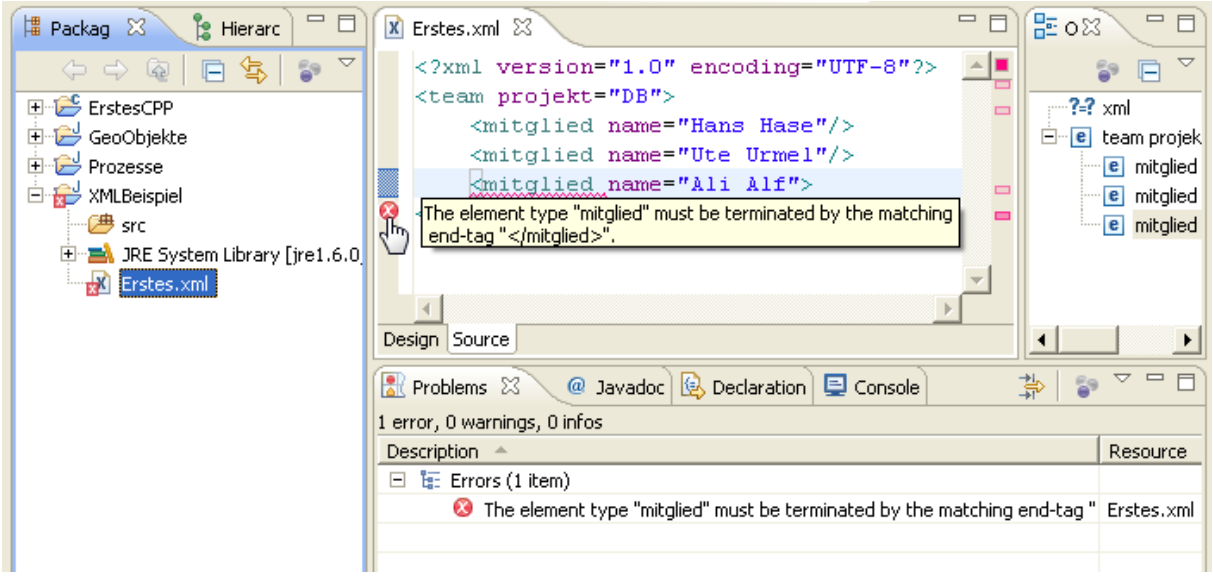
Im Source-Fenster kann dann ein XML-Dokument eingegeben werden. Die Wohlgeformtheit wird bei der Eingabe geprüft, kritische Bereiche rot unterschlängelt, was im folgenden Beispiel für das fehlende End-Tag gilt.



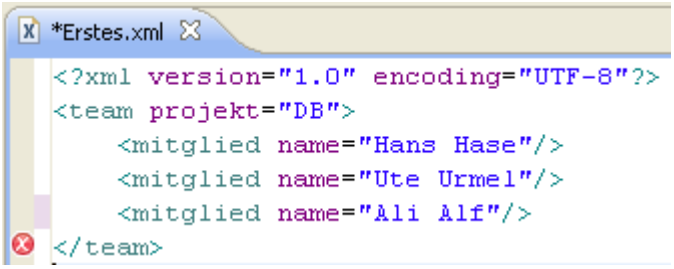
Achtung, nach dem Abspeichern werden Fehler nicht sofort gekennzeichnet. Dies passiert erst, wenn z. B. durch einen Rechtsklick auf das Dokument der Punkt „Validate“ ausgewählt wird. Eine Validierung ist auch für Ordner und ganze Projekte möglich.



Man erhält dann eine Fehleranzeige, schiebt man die Maus auf das weiße Kreuz im roten Kreis, wird der Fehlertext angezeigt, der auch unten im Reiter „Problems“ sichtbar ist.

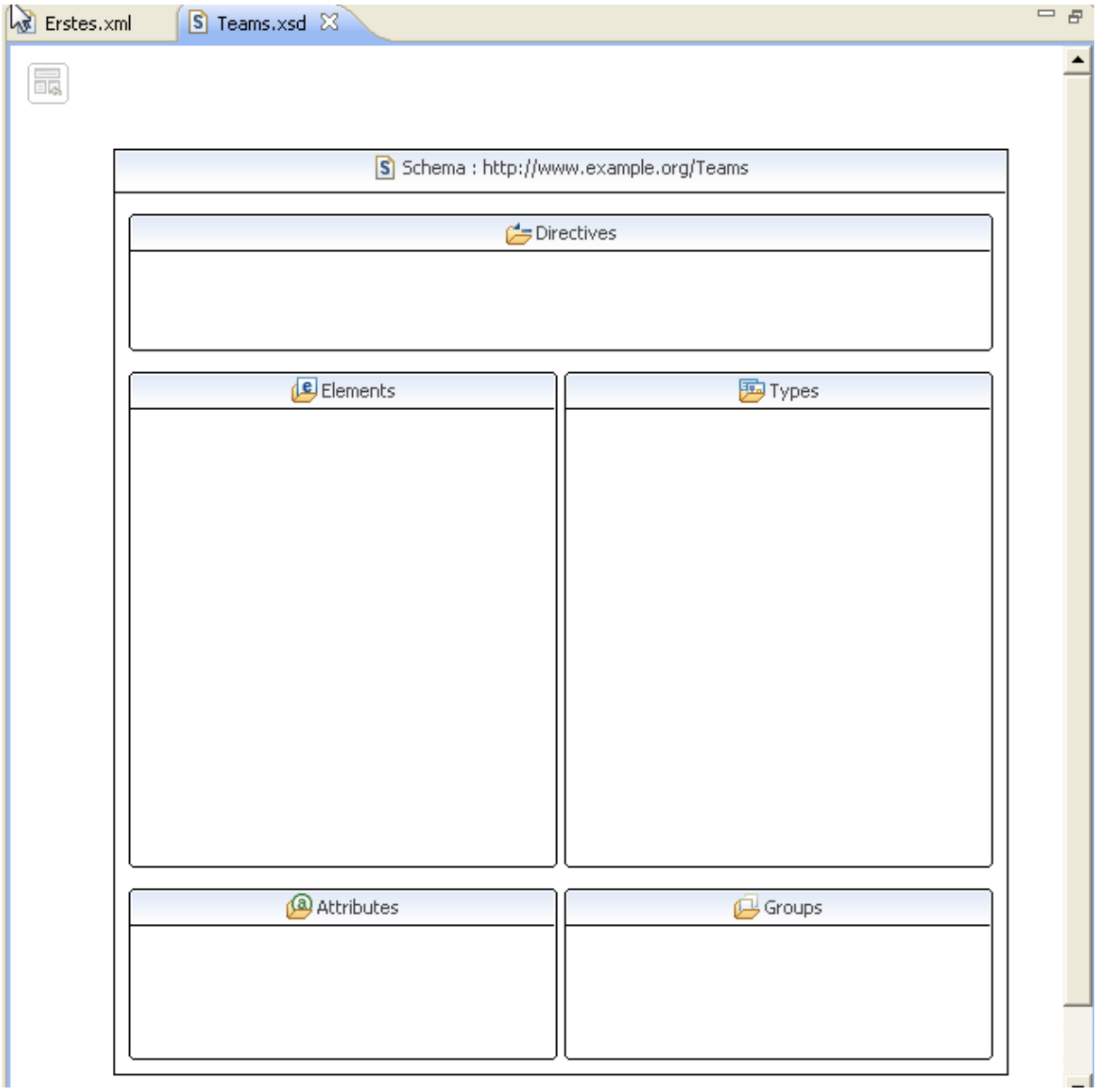


Wird der Fehler korrigiert, ändert sich die Anzeige am Rand nicht. Die Prüfung erfolgt erst, wenn erneut ein Validate ausgeführt wird.



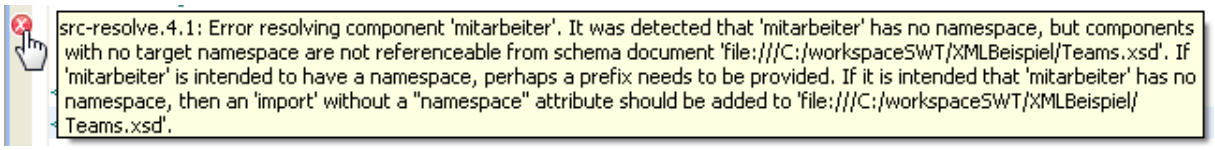
Ähnlich wie XML-Dokumente können auch XML-Schemata erstellt werden. Bei XML-Schemata wird aber durch das Validate geprüft, ob die Anforderungen an ein XML-Schema erfüllt sind, d.h. ob es gültig bezüglich der XML-Schema-Definition ist.

Für XML-Schemata gibt es eine etwas andere Design-Ansicht, in der ebenfalls gearbeitet werden kann, die hier aber nicht weiter betrachtet wird.



Das folgende Bild zeigt ein eingegebenes XML-Schema, wobei die Validierung zeigt, dass die geforderten Schema-Eigenschaften nicht erfüllt sind.

Die zugehörige leicht längliche Fehlermeldung lautet wie folgt.

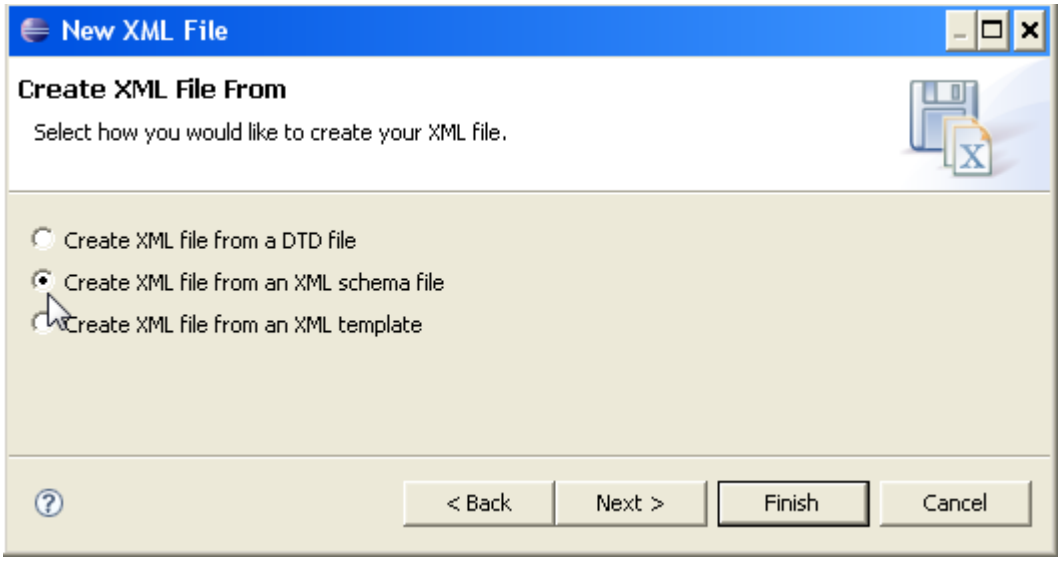


Das Schema wird jetzt wie folgt ergänzt und xmlns:xsi als Attribut von schema auf xmlns geändert (erleichtert etwas die Dokumentenerstellung).

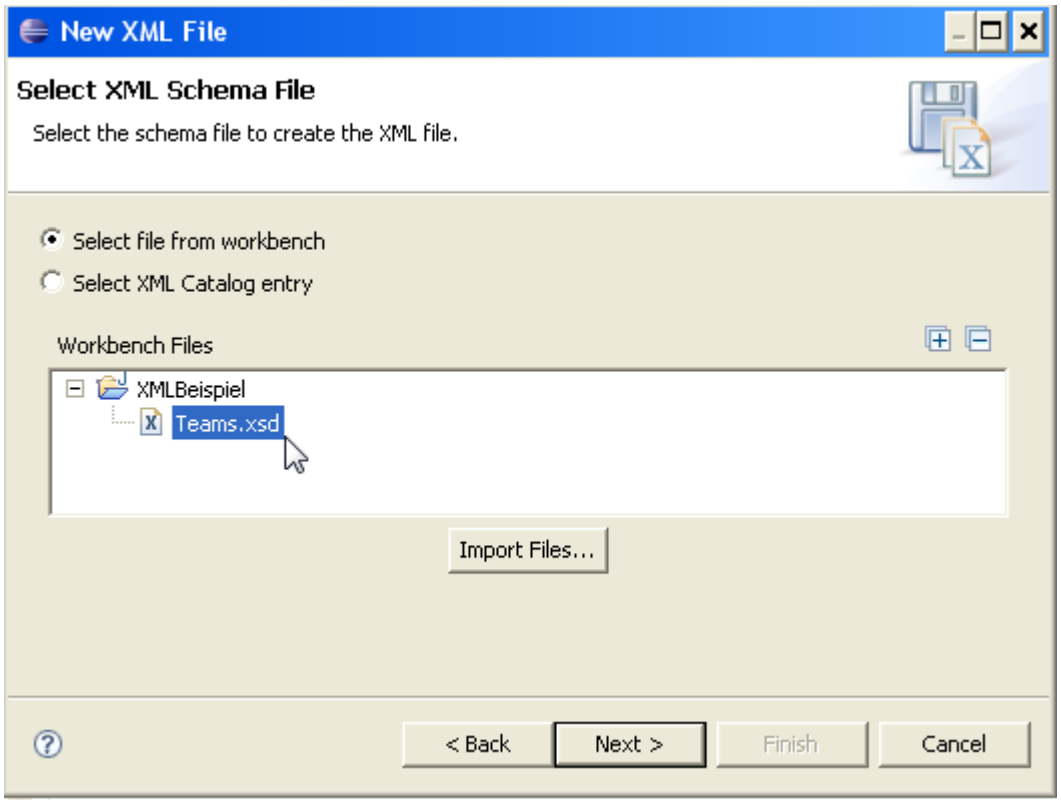
Erstes.xml | Teams.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/Teams"
    xmlns="http://www.example.org/Teams"
    elementFormDefault="qualified">

<xsd:element name="mitarbeiter" type="xsd:string"/>

<xsd:complexType name="TeamTyp">
 <xsd:sequence>
    <xsd:element ref="mitarbeiter" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="team" type="TeamTyp"/>

</xsd:schema>
```

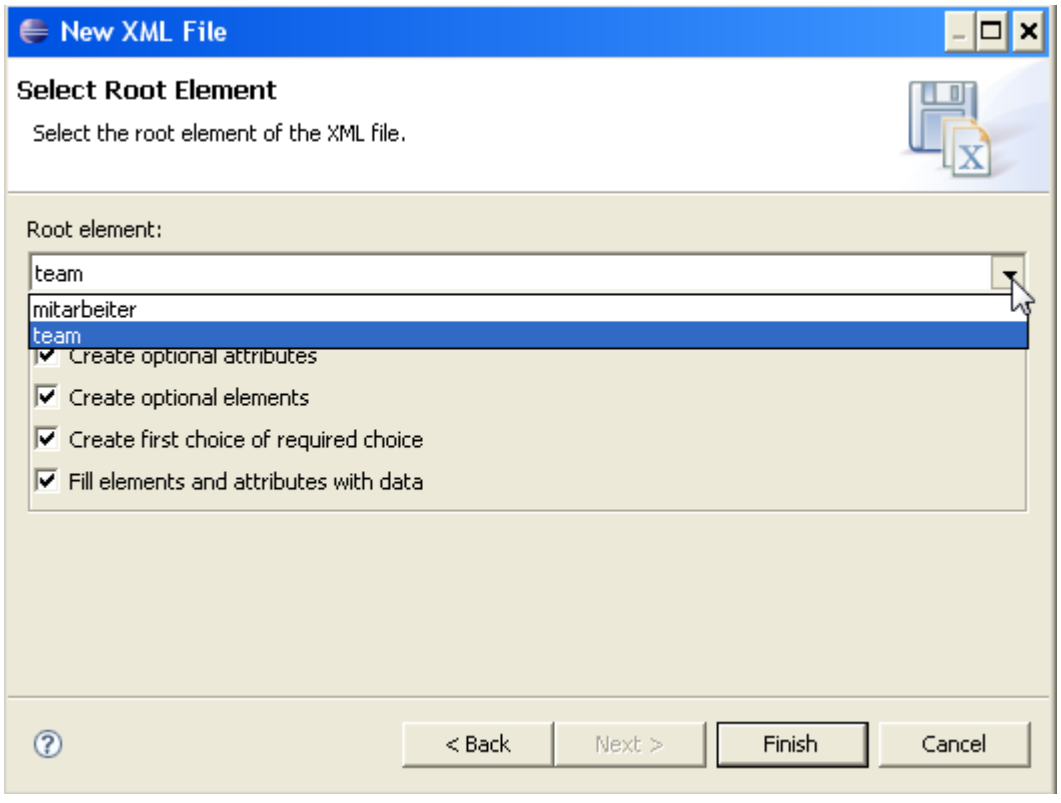
Es soll jetzt ein zum Schema passendes XML-Dokument angelegt werden. Dazu wird wie vorher beschrieben eine neue Datei angelegt und angegeben, dass diese zu einem existierenden XML-Schema passen soll.

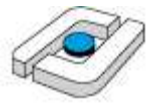


Danach kann das passende XML-Schema gesucht werden.



Danach wird das Wurzel-Element bestimmt und weitere Einstellungen können passend ausgesucht werden.





Es wird folgende Datei angelegt, wobei die Namensraumvergabe mit „p:“ wenn gewünscht, von Hand entfernt werden muss. (Von Hand bedeutet, dass es unter „Edit“ den passenden Menü-Punkt „Find/Replace...“ gibt.)

Erstes.xml | Teams.xsd | *Zwotes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<p:team xmlns:p="http://www.example.org/Teams"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/Teams Teams.xsd ">
  <p:mitarbeiter>p:mitarbeiter</p:mitarbeiter>
</p:team>
```

Bei der Eingabe wird die Gültigkeit und Wohlgeformtheit überprüft, so dass das folgende ungültige Element markiert wird.

Erstes.xml | Teams.xsd | *Zwotes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<team xmlns="http://www.example.org/Teams"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/Teams Teams.xsd ">
  <mitarbeiter>Kim Shi</mitarbeiter>
  <mitarbeiter>Omar Abdah </mitarbeiter>
  <mitarbeter />
</team>
```

Eine Prüfung auf Gültigkeit findet durch „Validate“ statt, so dass der Fehler konkret angezeigt wird. Die Platzierung der Fehlermeldung kann ab und zu irreführend sein, wenn der Fehler erst mit dem letzten schließenden Tag erkannt wird und so hier die Fehlermeldung z. B. bei der Prüfung von Abhängigkeiten steht.

Erstes.xml | Teams.xsd | Zwotes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<team xmlns="http://www.example.org/Teams"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/Teams Teams.xsd ">
  <mitarbeiter>Kim Shi</mitarbeiter>
  <mitarbeiter>Omar Abdah </mitarbeiter>
```

cvc-complex-type.2.4.a: Invalid content was found starting with element 'mitarbeter'. One of '{"http://www.example.org/Teams":mitarbeiter}' is expected.

Leider gibt es keine deutliche positive Meldung, dass eine Validierung erfolgreich war.

Erstes.xml | Teams.xsd | Zwotes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<team xmlns="http://www.example.org/Teams"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/Teams Teams.xsd ">
  <mitarbeiter>Kim Shi</mitarbeiter>
  <mitarbeiter>Omar Abdah </mitarbeiter>
  <mitarbeiter />
</team>
```

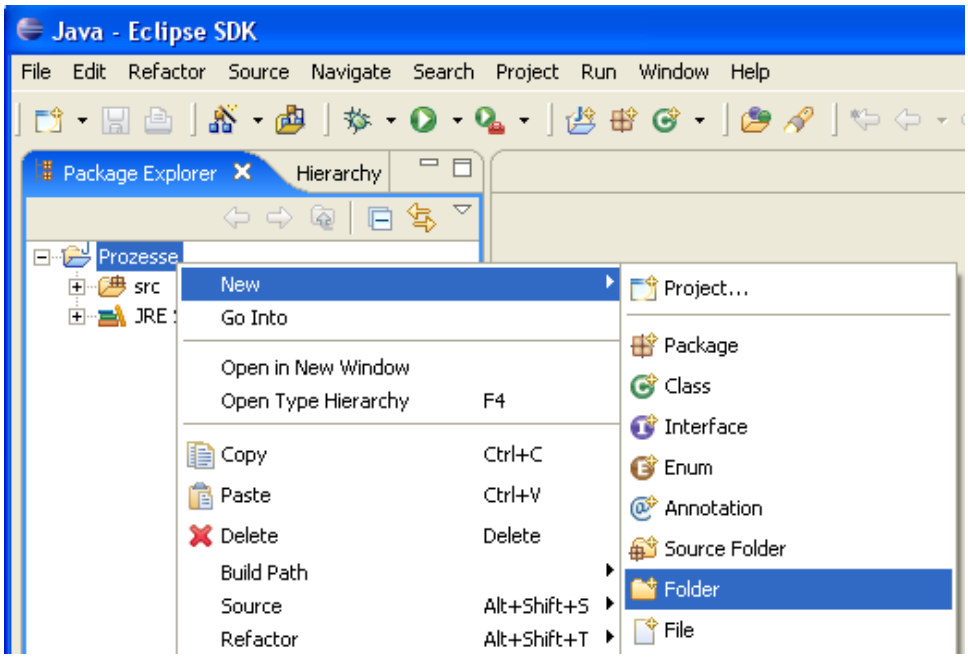
11 Erstellung von Analysemodellen mit UMLet

In der Vorlesung wird der pragmatische Ansatz verfolgt, dass nicht alle Dokumente eines Projekts bei jeder Änderung aktualisiert werden müssen. Analysemodelle spielen am Anfang die zentrale Rolle, da sie zum Verständnis der Aufgabenstellung dienen und den Übergang zur Realisierung ermöglichen. Nach einer vollständigen Analyse, die bei inkrementell vorgehenden Projekten immer wieder ergänzt wird, werden die Analysedokumente und –modelle eingefroren und nicht mehr aktualisiert.

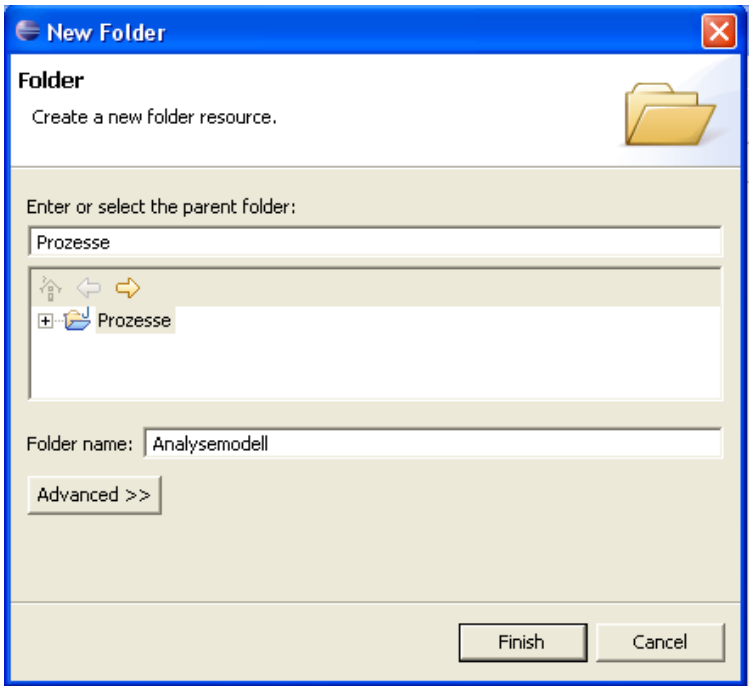
In der Veranstaltung wird das Werkzeug UMLet für die Erstellung von Skizzen genutzt. Das Werkzeug kann entweder ohne Eclipse oder mit Eclipse, dann aber mit geringer Einbindung zu den restlichen Eclipseteilen genutzt werden. Das Eclipse-Plugin kann von der Web-Seite http://www.umlet.com/ geladen werden. Die jar-Datei com.umlet.plugin_10.3.0.jar muss dann in den plugins-Ordner von Eclipse abgelegt werden.

Zum Aufruf des Werkzeugs ohne Eclipse wird die stand-alone-Variante von der Web-Seite geladen und umlet.jar mit einem Doppelklick gestartet. Falls dann ein Entpackprogramm aufgeht, haben Sie eine fehlerhafte Einstellung, dass Dateien mit der Endung .jar mit diesem Programm geöffnet werden sollen. Diesen Eintrag sollten Sie im Entpackprogramm löschen. Alternativ können Sie in das Verzeichnis mit einer „Dos-Box“ gehen und das jar-File von Hand mit `java -jar umlet.jar` starten.

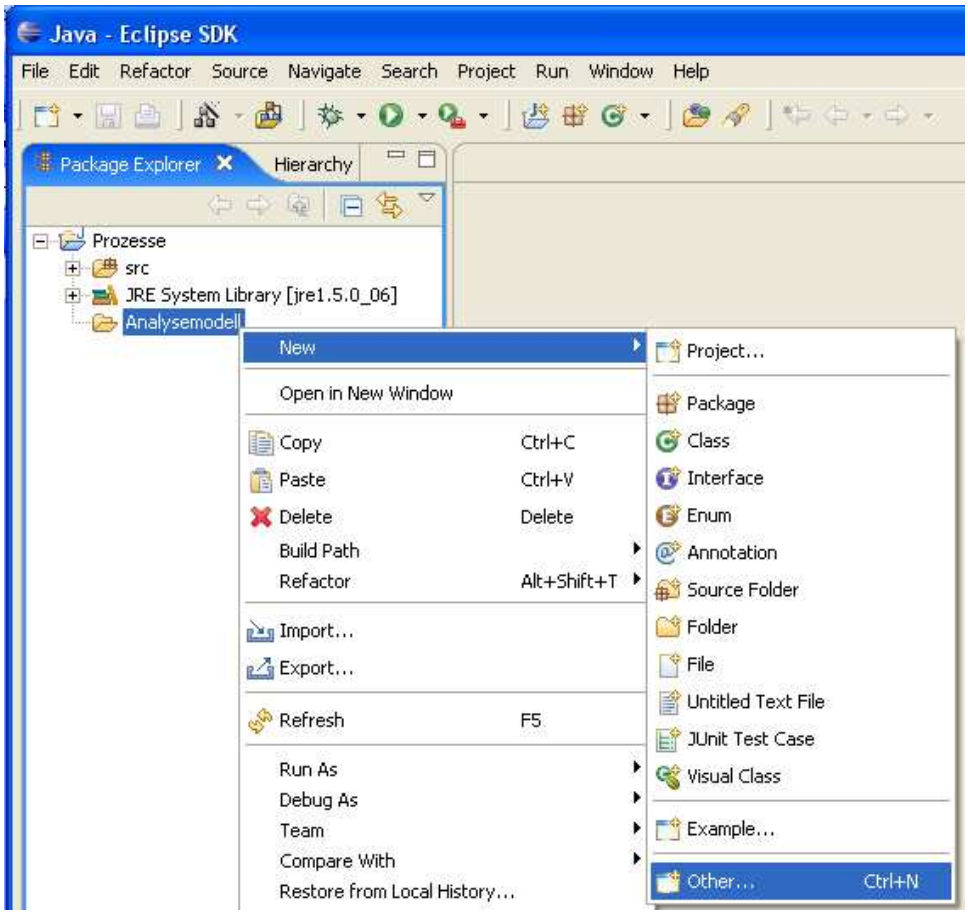
In Eclipse ist es sinnvoll, zunächst ein Projekt und in dem Projekt einen Ordner „Analysemodell“ anzulegen. Dies geschieht durch einen Rechtsklick auf dem Projekt und den Auswahlen „New -> Folder“.



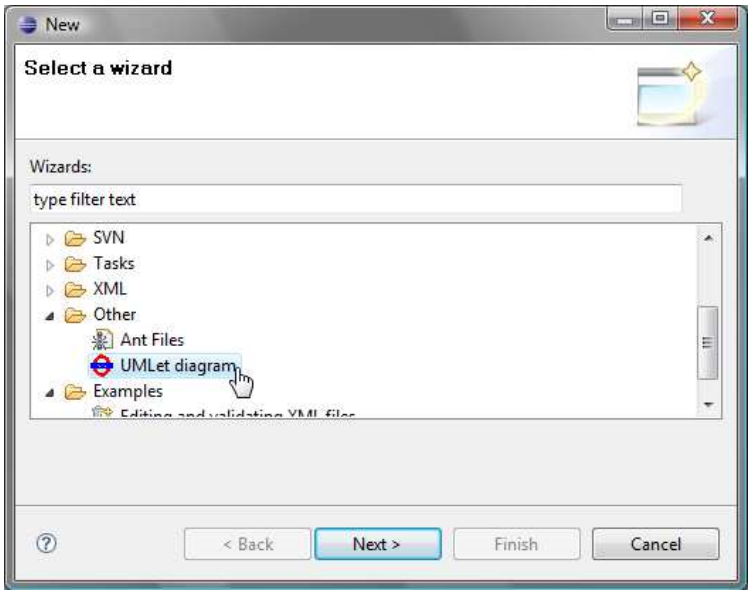
Hier muss nur noch der Name des Ordners eingegeben werden.



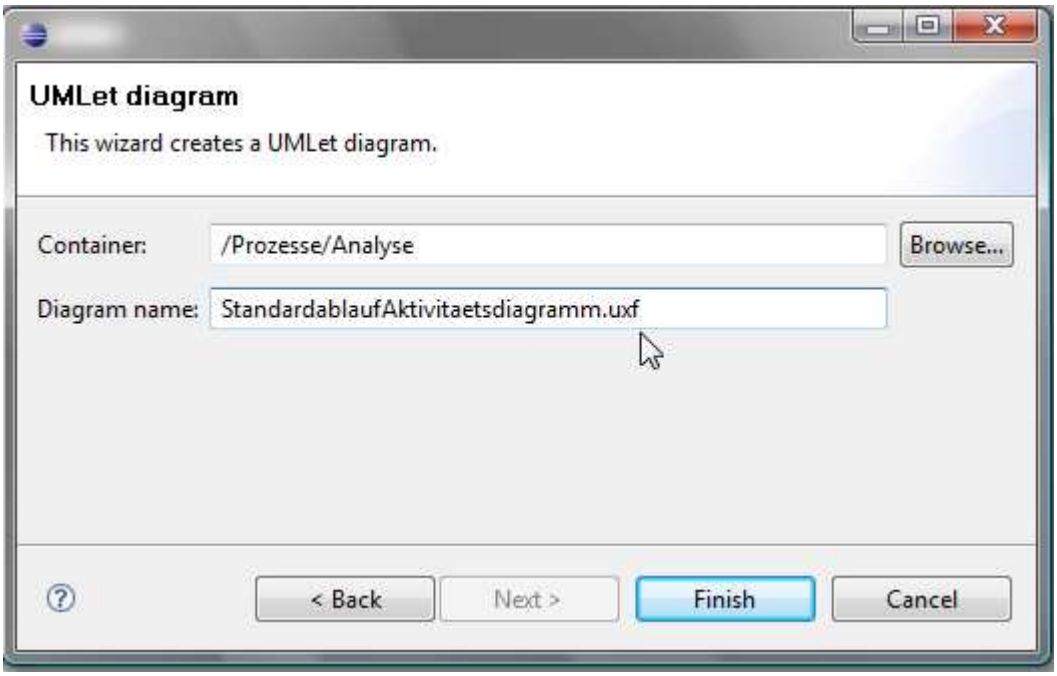
Zum Anlegen des Analysemodells wird nach einem Rechtsklick auf den Ordner Analysemodell „New->Other“ ausgewählt.



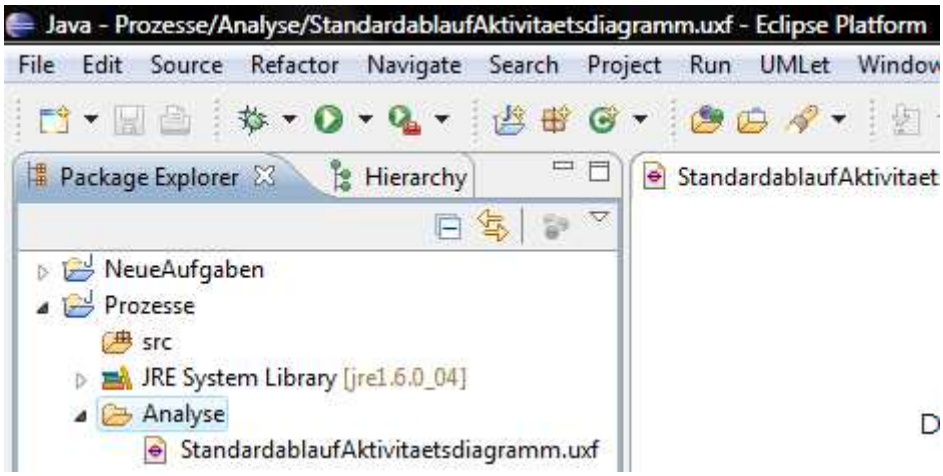
Hier wählt man den Ordner „Other“ und dann „UMLet diagram“. Bitte hier nicht den Ordner „UML Diagrams“ auswählen, da diese Diagramme zu einem anderen Werkzeug gehören.



Danach wird „Next>“ gewählt. Im folgenden Fenster kann dann der Name des Diagramms bei „Diagram name:“ angegeben werden. Die Art des Diagramms, also z. B. Use Case Diagram oder Aktivitätsdiagramm, wird erst später im Werkzeug eingestellt. Deshalb ist es sinnvoll, die Art des Diagramms im Dateinamen zu erwähnen. Der folgende Name steht also nur für ein Beispiel.

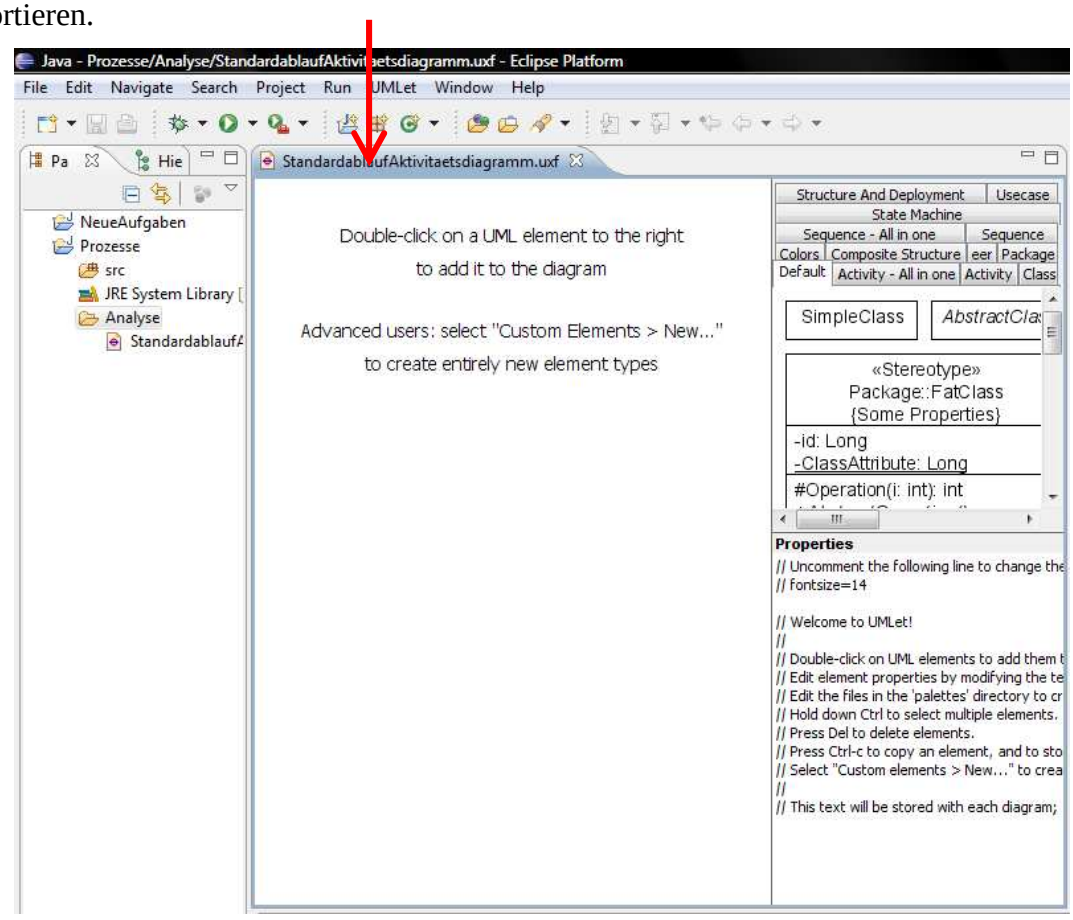


Danach befindet sich eine neue Datei mit der Endung uxf im Ordner Analysemodell des Eclipse-Projekts.



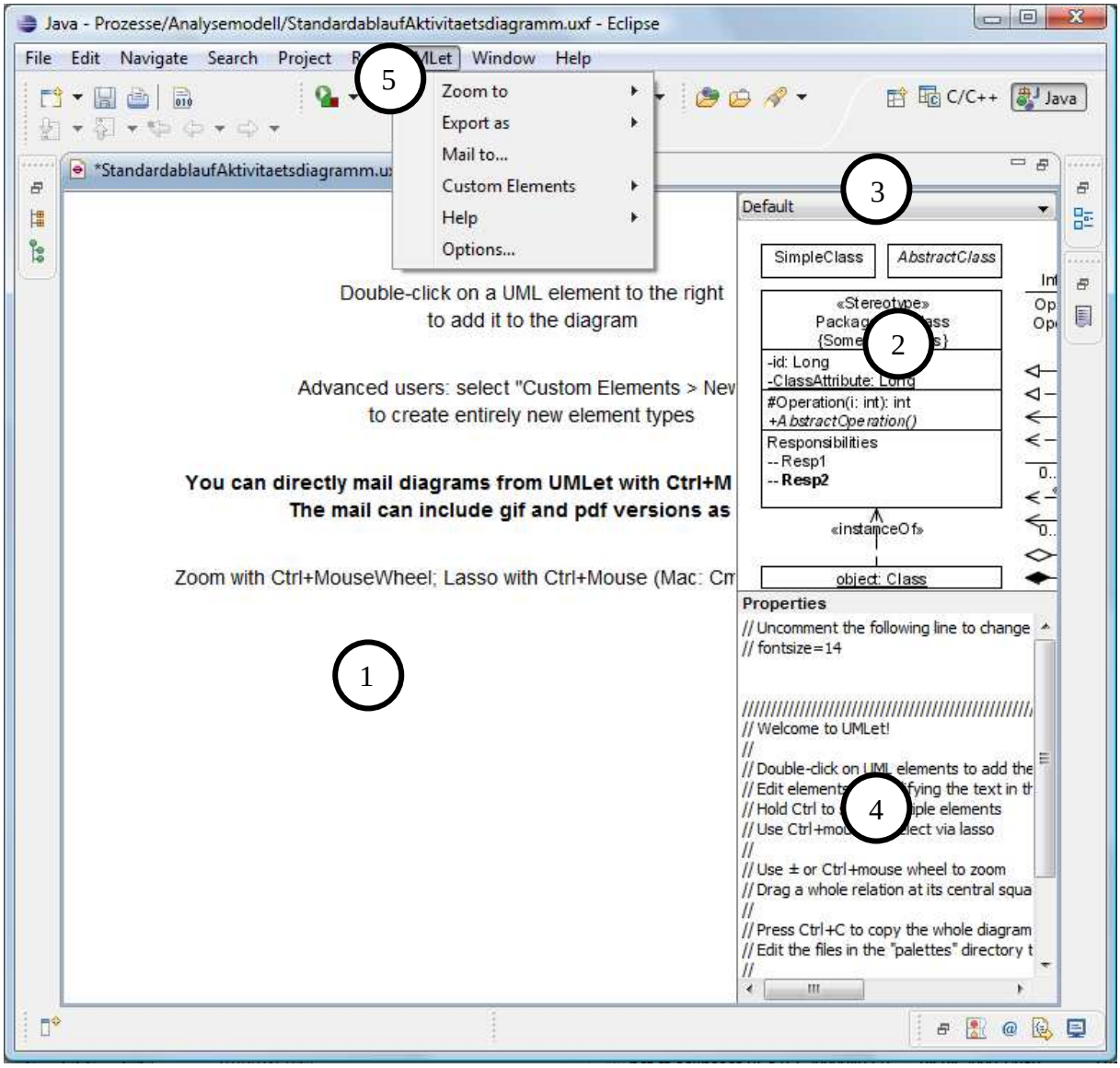
Neben dem Projekt-Browser wird das Diagramm bereits geöffnet. Später wird durch einen Doppelklick auf die uxf-Datei UMLet im zentralen Fenster von Eclipse geöffnet.

Falls dies nicht der Fall sein sollte, da das Zusammenspiel Java 6, Eclipse 3.4 und UMLet nicht klappt, kann zunächst folgender Trick genutzt werden. Eclipse wird mit der unvollständig geöffneten Datei geschlossen. Danach wird Eclipse erneut geöffnet und mit etwas Glück steht der UMLet-Diagrammeditor zur Verfügung. In einem letzten Versuch kann man ein schon existierendes Diagramm importieren und dann hoffentlich erfolgreich öffnen. Falls es nicht geht, kann man das Diagramm außerhalb von Eclipse erzeugen und später importieren.



Da das Fenster relativ klein zum Arbeiten ist, sollte dieses Fenster vergrößert werden. Generell geschieht dies durch einen Doppelklick auf den jeweiligen Reiter (mit Pfeil im vorherigen Bild markiert).

Die folgenden Nutzungshinweise gelten auch, wenn UMLet nicht in Eclipse genutzt wird.



Im Werkzeug gibt es neben der üblichen Dateibehandlung die vier markierten wichtigen Bereiche.

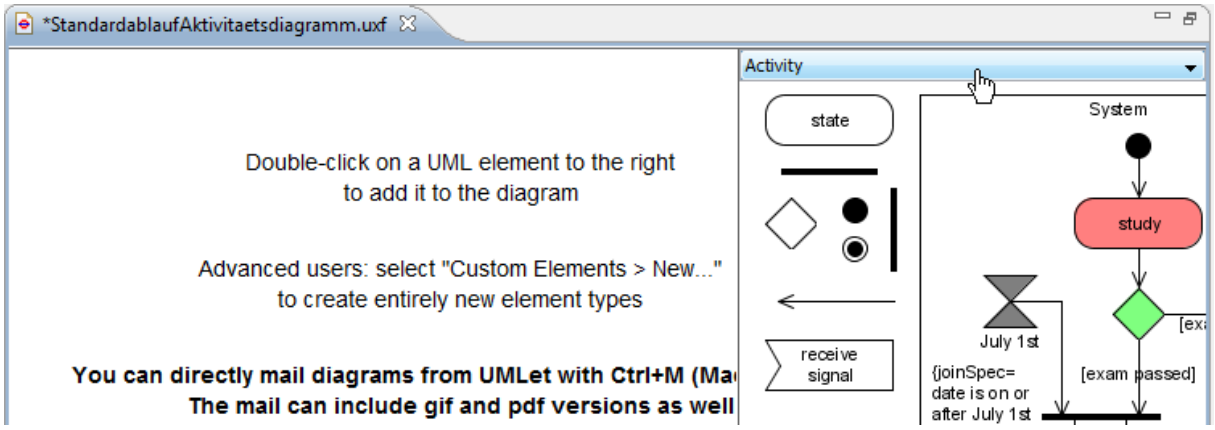
1. Dies ist der Zeichenbereich, in dem alle Elemente verknüpft und angezeigt werden.
2. In diesem Bereich wird eine Palette vom UML-Zeichenelementen angezeigt, die durch einen Doppelklick in den Zeichenbereich übernommen werden. (Diese Art der Steuerung ist etwas gewöhnungsbedürftig, aber recht effizient.)
3. In dieser Drop-Down-Box kann man unterschiedliche Paletten auswählen, die für unterschiedliche UML-Diagramme verschiedene Zeichenelemente anbieten.

4. Wenn man Beschriftungen z. B. von Aktivitäten oder Inhalte von Klassen ändern möchte, passiert dies immer in dieser Textbox.
5. Exportieren der Graphiken in verschiedenen Formen (eps, jpg, pdf, svg)

UMLet macht keine Syntaxprüfung, dass bedeutet, dass man beliebigen „Unsinn“ in die Diagramme zeichnen kann. Dies ist aber auch ein wichtiger Vorteil für die Analysephase, da man sich hier frei für eine Darstellungsform entscheiden kann. Wichtig ist nur, dass sie von jedem Projektbeteiligten gelesen werden kann und dass die Grundideen der Darstellungsweise dokumentiert sind.

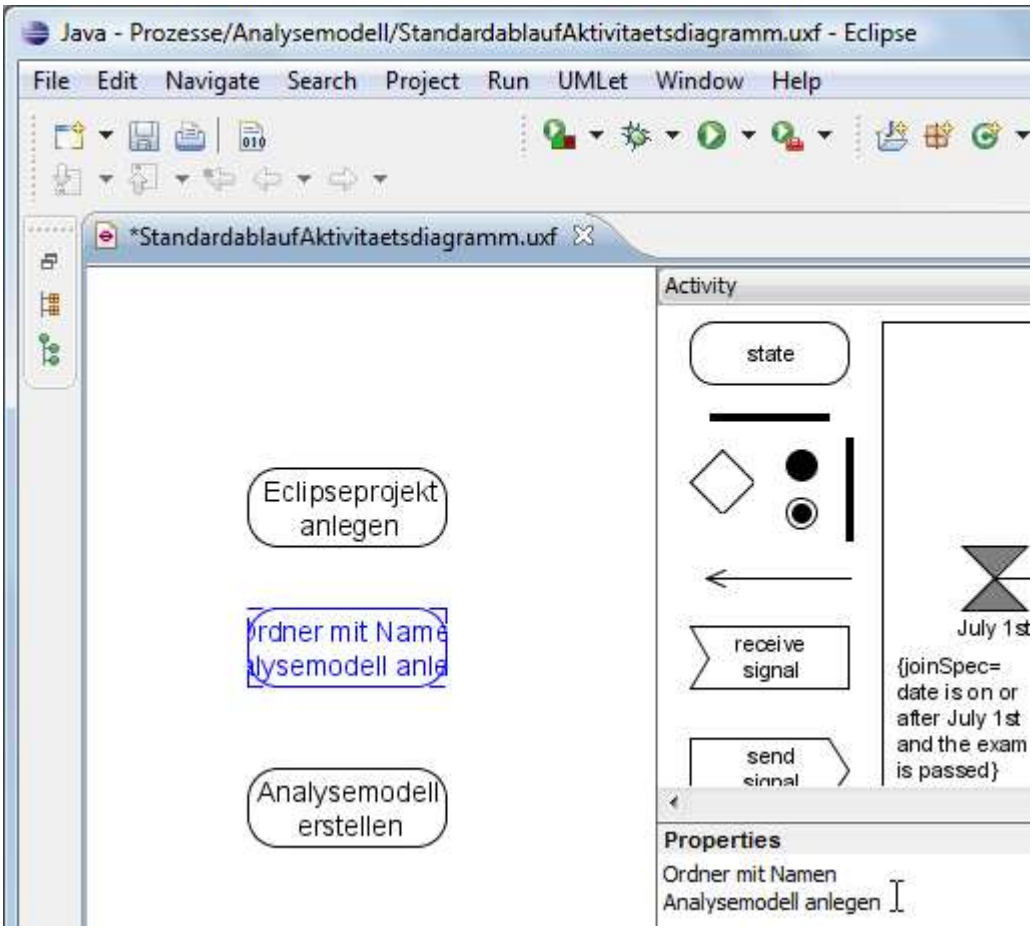
Beim Arbeiten mit UMLet ist zu beachten, dass man mit einem Doppelklick auf ein Zeichenelement, egal ob rechts in der Palette oder links in der Zeichenfläche, das ausgewählte Element verdoppelt. Dies ist sehr hilfreich, wenn man z. B. mehrere Aktivitäten oder auch Pfeile im Zeichenbereich organisieren möchte. Bei Anfängern führt dieser Ansatz aber ab und zu zu Problemen. Im Folgenden ist skizziert, wie man ein Aktivitätsdiagramm anlegen kann, dabei ist jeder Nutzer zum selbst Experimentieren aufgerufen.

Zunächst wird die zu Aktivitätsdiagrammen gehörende Palette ausgewählt.

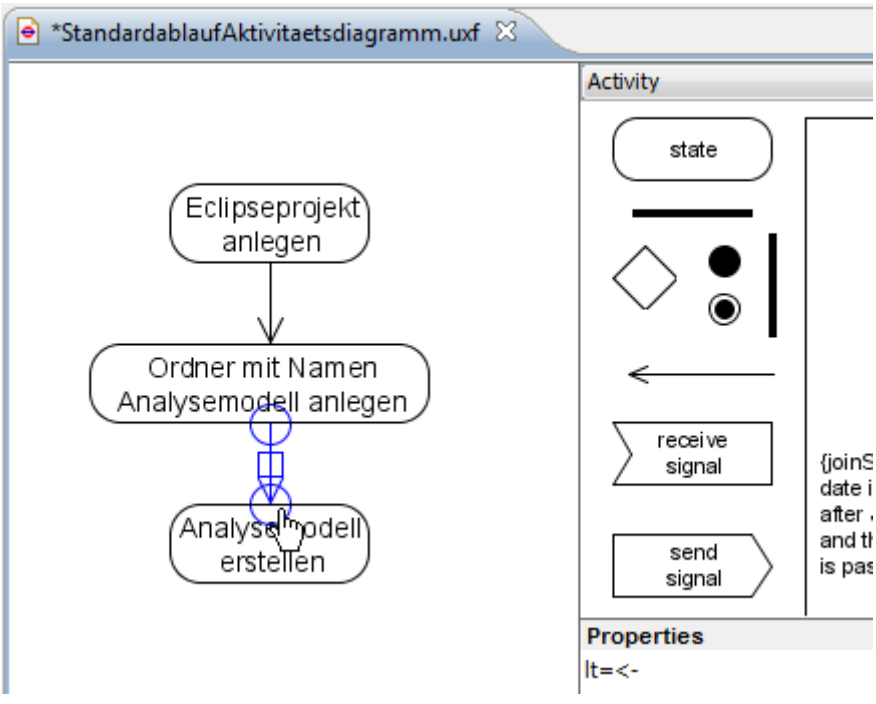


Danach wird auf der rechten Seite die Aktion, etwas unsauber als „state“ bezeichnet, doppelt angeklickt und so in den Zeichenbereich übernommen. Um mehrere Aktionen zu erzeugen, wird dann ein Doppelklick auf der Aktion im Zeichenbereich ausgeführt. Graphische Elemente können generell mit gedrückter linker Maustaste verschoben werden.

Die Beschriftung wird im erwähnten vierten Bereich geändert. Wenn man die Maus über ein Element schiebt, kann man dessen Größe anpassen.

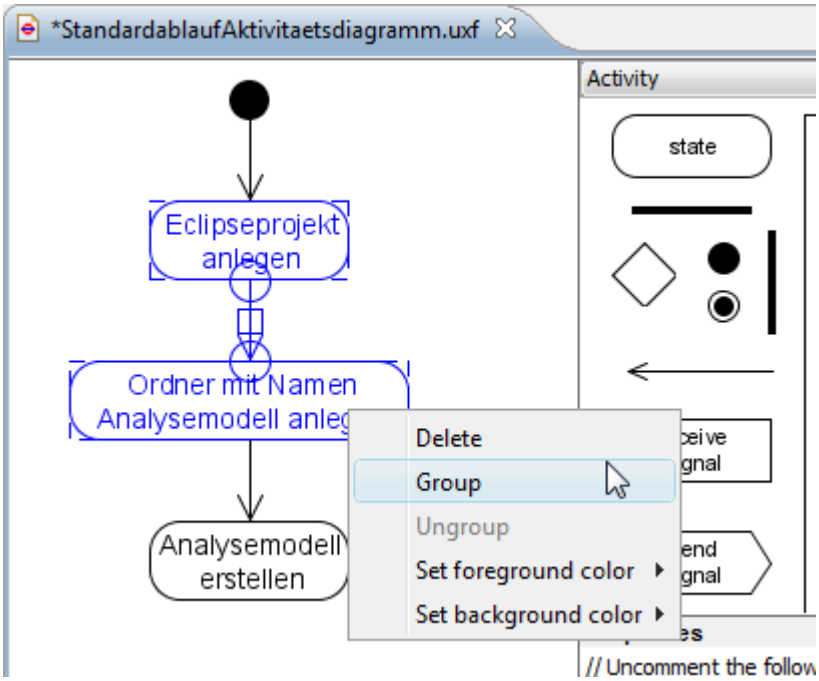


Danach werden die anderen Elemente eingefügt und miteinander verbunden, Pfeile werden ebenfalls mit einem Doppelklick kopiert.

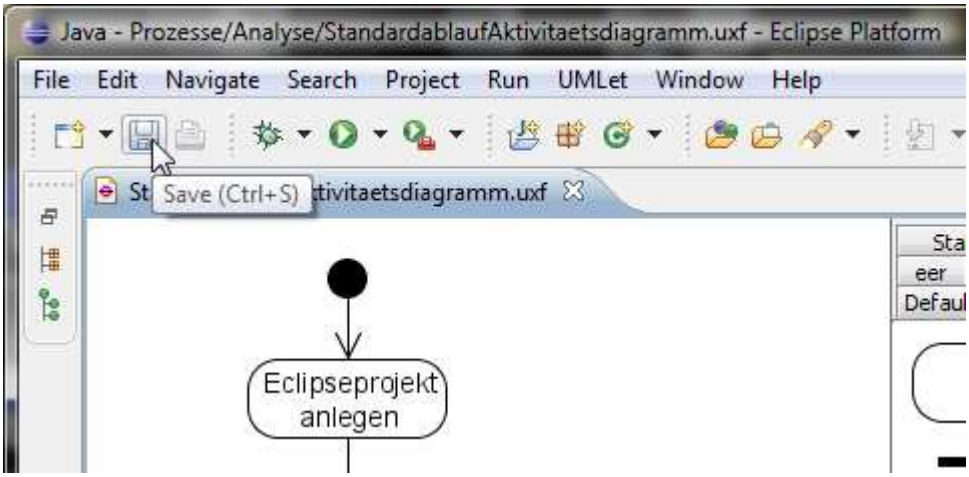


Klickt man neben das Diagramm und verschiebt bei gedrückter Linker Maustaste die Maus, so wird das gesamte Diagramm verschoben. Man kann mit gedrückter Strg-Taste mehrere

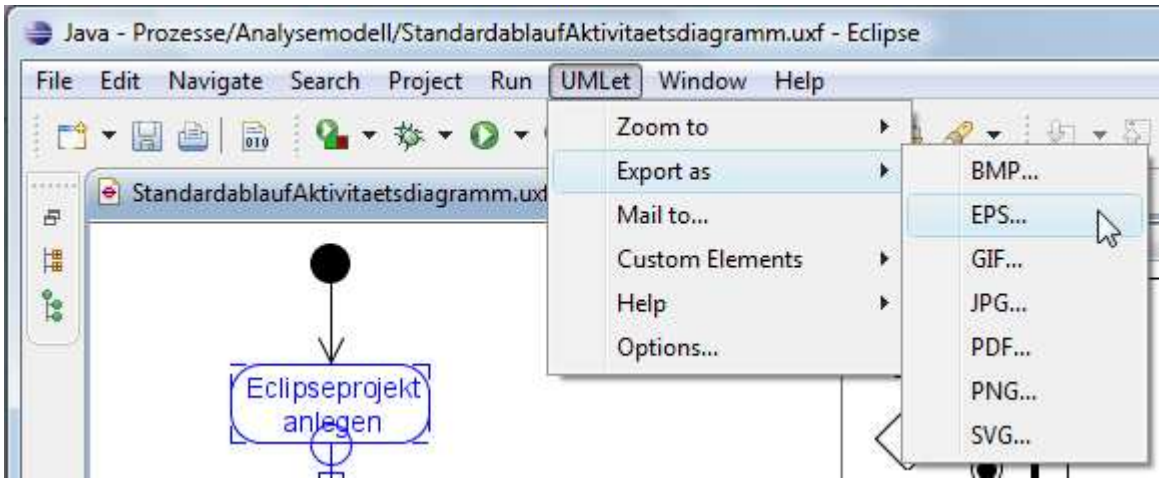
Elemente selektieren und diese dann mit einem Rechtsklick im dann sichtbaren Menü zu einer Gruppe zusammenfassen. Danach wird diese Gruppe immer zusammenbehandelt, was z. B. für das Verschieben einer Teilmenge der Diagrammelemente sehr hilfreich ist.



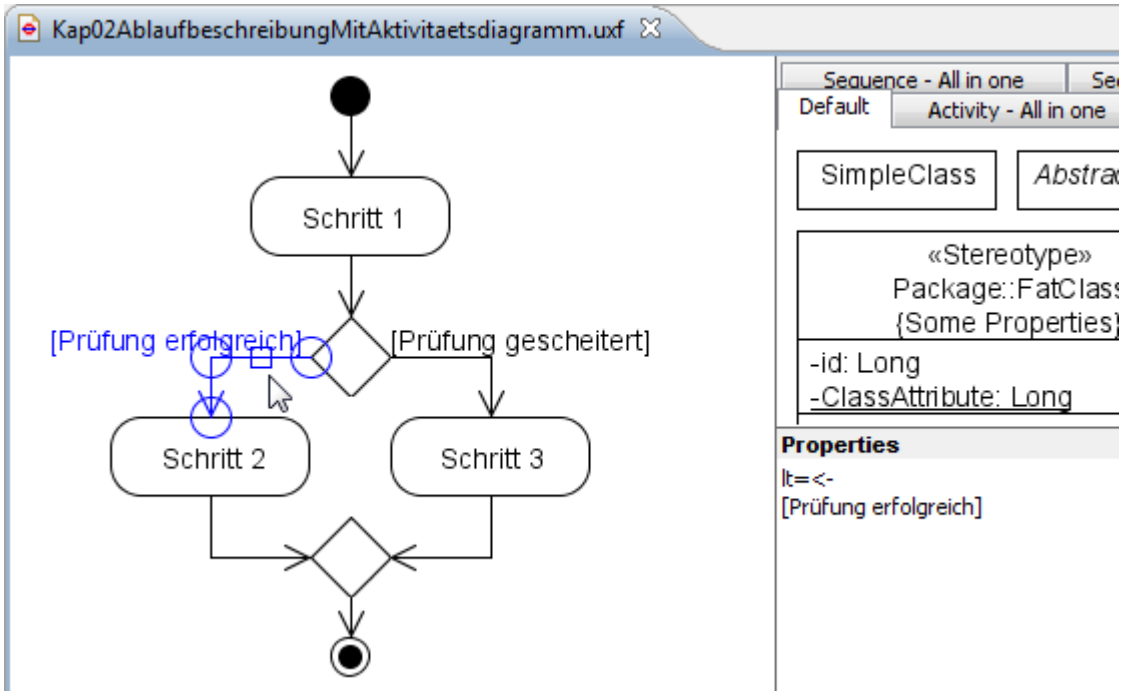
Nachdem ein Modell vollständig eingegeben wurde, muss es mit dem Punkt „Save“ im File-Menü der Eclipse-Umgebung gespeichert werden. Eine Umbenennung ist hier nicht möglich!



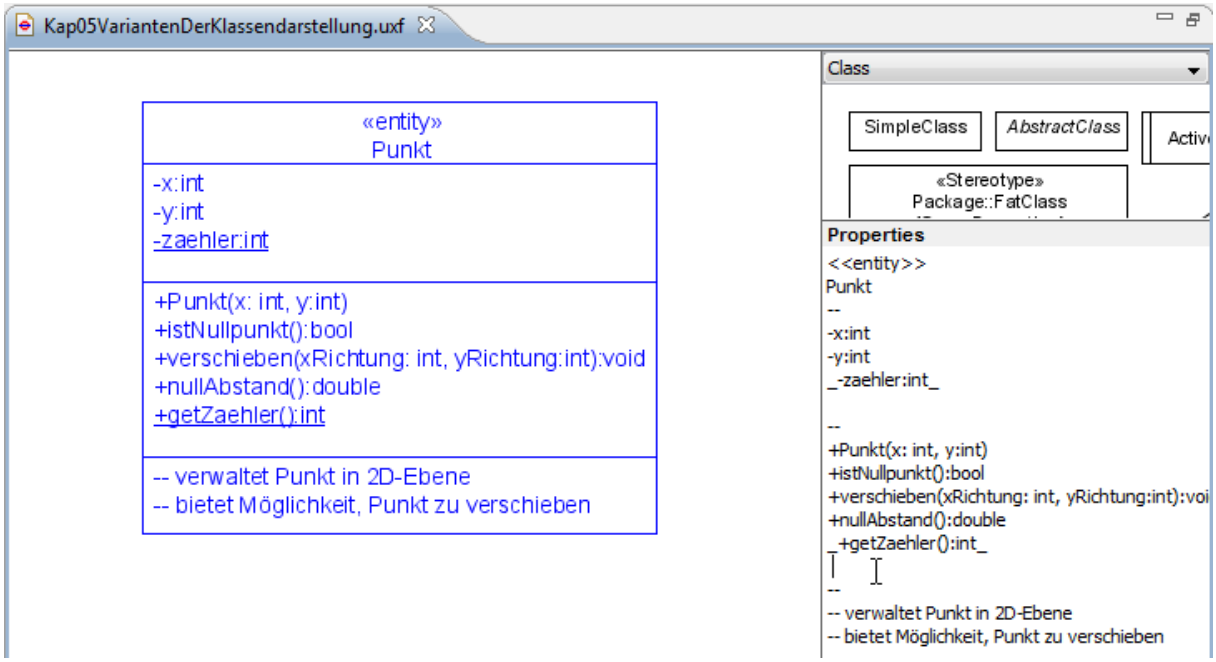
Im UMLet-Menü der Hauptzeile kann man das Bild in andere Formate exportieren, dabei können diese Formate nicht wieder eingelesen werden. Für Ausdrucke ist besonders Encapsulated Postscript zu empfehlen, dass zwar in der Bildschirmdarstellung in Word und Powerpoint sehr schlecht aussieht, aber in Ausdrucken und PDF-Exporten sehr gut aussieht. Für Powerpoint-Präsentationen ist JPG geeignet. Bei EPS ist allerdings zu beachten, dass Bilder, in denen große Kästen genutzt werden, die weitere Diagramminhalte enthalten, z. B. Pakete oder Zustände mit Teilzuständen, eventuell nicht vollständig dargestellt werden. Der Trick ist dann, diese Kästen zuletzt zu zeichnen (am Ende kopieren, Originalkasten löschen, Kopie an richtige Stelle setzen).



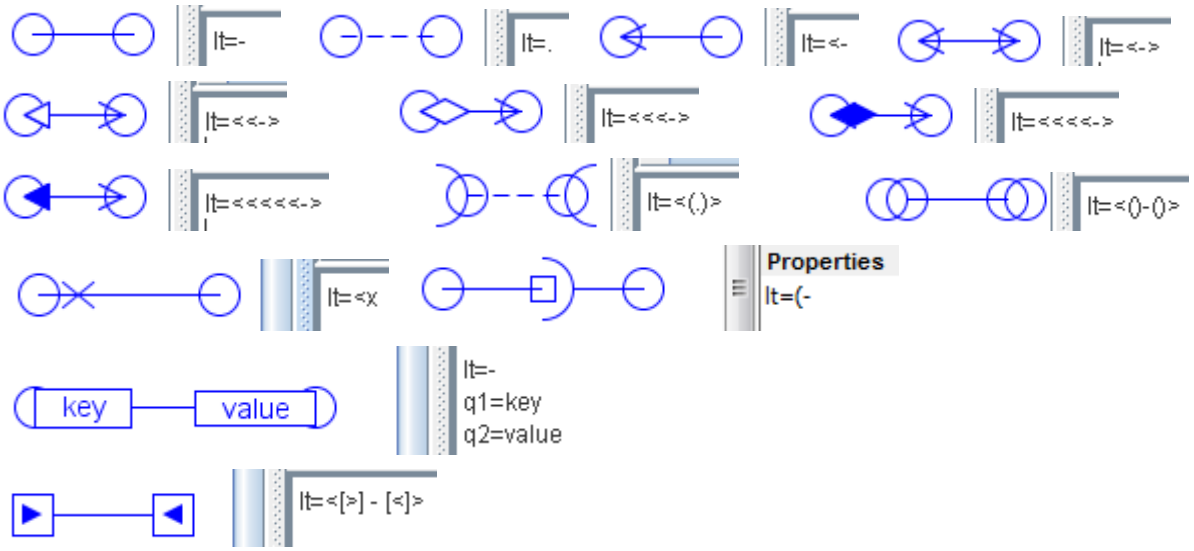
Möchte man Kanten z. B. bei Entscheidungen beschriften, ist zu beachten, dass die erste Zeile in der Beschriftungszeile gleich bleibt, da mit ihr angegeben wird, an welchen Enden Pfeilspitzen stehen sollen. Die folgende Abbildung zeigt links eine selektierte Kante und rechts das Textfeld, in dem die Beschriftung eingetragen wird. Knicke in Kanten erzeugt man, wenn man auf einer selektierten Kante an dem Punkt, wo ein neuer Knickpunkt entstehen soll, an der Kante mit gedrückter linker Maustaste zieht. Der markierte Mittelpunkt der Kante dient dazu, die gesamte Kante zu verschieben.



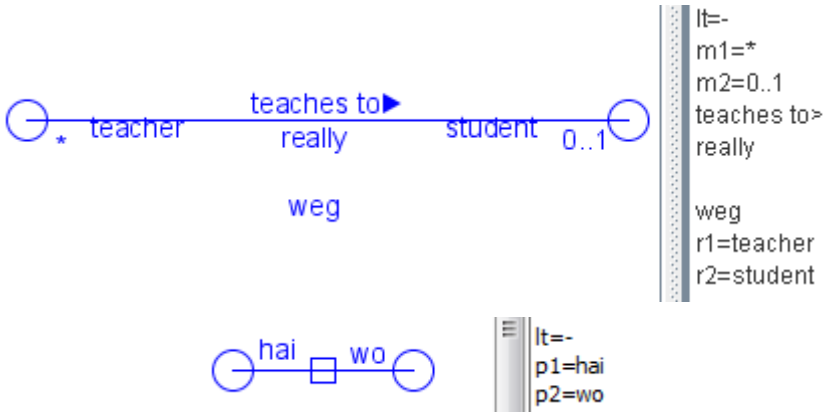
Bei der Bearbeitung von Klassen ist noch zu beachten, dass bei Texteingaben eine leere Zeile mit zwei Minuszeichen dazu führt, dass im Klassendiagramm ein Strich gemalt wird. Möchte man Leerzeilen einfügen, müssen diese mindestens ein Leerzeichen enthalten. Klassenvariablen und Klassenmethoden werden in der UML unterstrichen, dies ist durch das Voranstellen und Abschließen mit einem Unterstrich möglich. Es gibt einige weitere Steuerungsbefehle, von denen einige im Folgenden betrachtet werden.



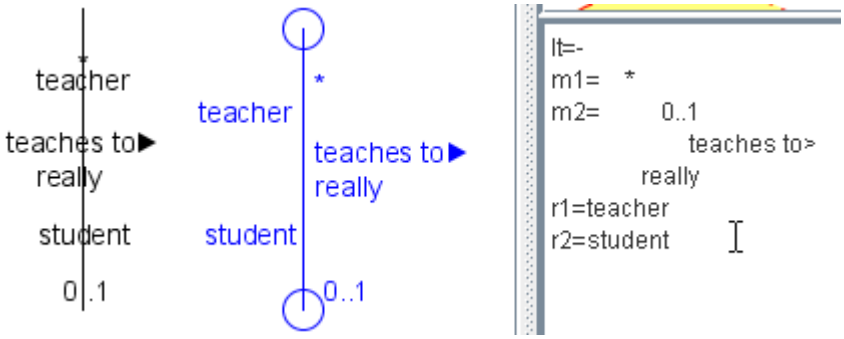
Beschäftigt man sich etwas genauer mit UMLet, gibt es noch einige wenige Steuerbefehle, die in der Beschriftungsbox eingegeben werden können. Die folgenden Bilder zeigen zunächst links die Linienart und rechts den eingegebenen Text, die Kreise zeigen nur, dass die Linie selektiert wurde.



Weiterhin gibt es verschiedene Plätze, an denen Eigenschaften an Linien notiert werden können. Die Leerzeile im folgenden Bild entsteht nur, da in dieser Zeile im Texteditor ein Leerzeichen steht.

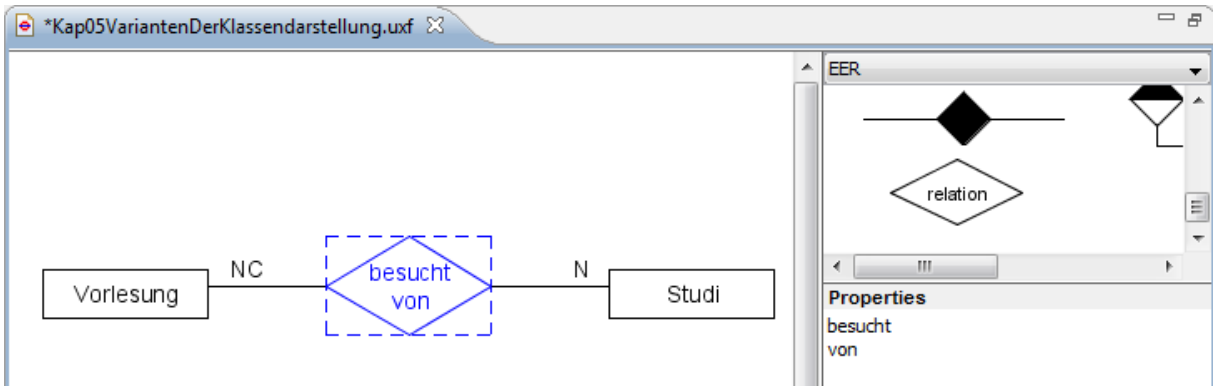


Da die Platzierung nicht immer optimal ist, kann man die Texte etwas mit Leerzeichen verschieben.

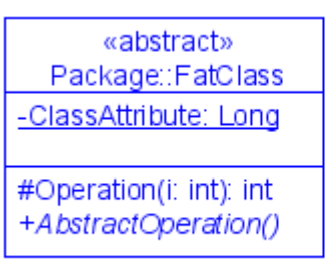


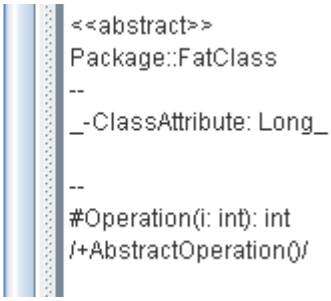
Ausgefüllte Elemente können eine Farbe haben.

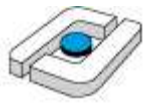
Beim Reiter „EER“ gibt es eine beschriftbare Relation, die man auch zur ER-Modellierung nutzen kann.



Für abstrakte Methoden müssen Schrägstriche um den Namen herum stehen, damit dieser kursiv dargestellt wird.


«abstract»
Package::FatClass
-ClassAttribute: Long
#Operation(i: int): int
+AbstractOperation()


<<abstract>>
Package::FatClass
--
-ClassAttribute: Long
--
#Operation(i: int): int
/+AbstractOperation()/



12 Nutzung von EclipseUML (Omondo)

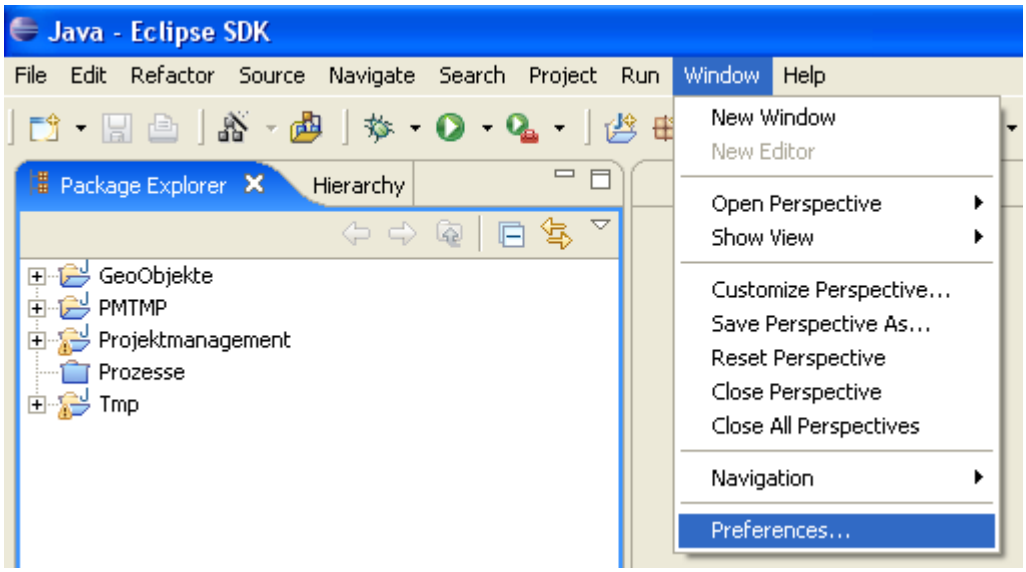
Omondo ist ein UML-Werkzeug, das eng in die Java-Entwicklung eingebunden ist. Hier wird die Omondo-Version genutzt, die zum Lernen zur freien Verfügung steht. Wie für jedes UML-Werkzeug gilt, dass man sich mit der Arbeitsweise des Werkzeugs vertraut machen muss. Da nicht immer alle UML-Notationsmöglichkeiten unterstützt werden, muss man die Möglichkeiten des Werkzeugs mit dem gewünschten Vorgehen verknüpfen, was leider dazu führt, dass man einen Arbeitsprozess an ein Werkzeug anpassen muss, was eigentlich dem Sinn von Software widerspricht.

Da es eine kommerzielle Version von Omondo gibt, stehen nicht alle Möglichkeiten in der freien Variante zur Verfügung, wobei die einfache Entwicklung von Java-Programmen vollständig unterstützt wird. Nervig bleiben trotzdem anwählbare Menü-Punkte, die dann nur zu einer Werbung für das kommerzielle Produkt führen.

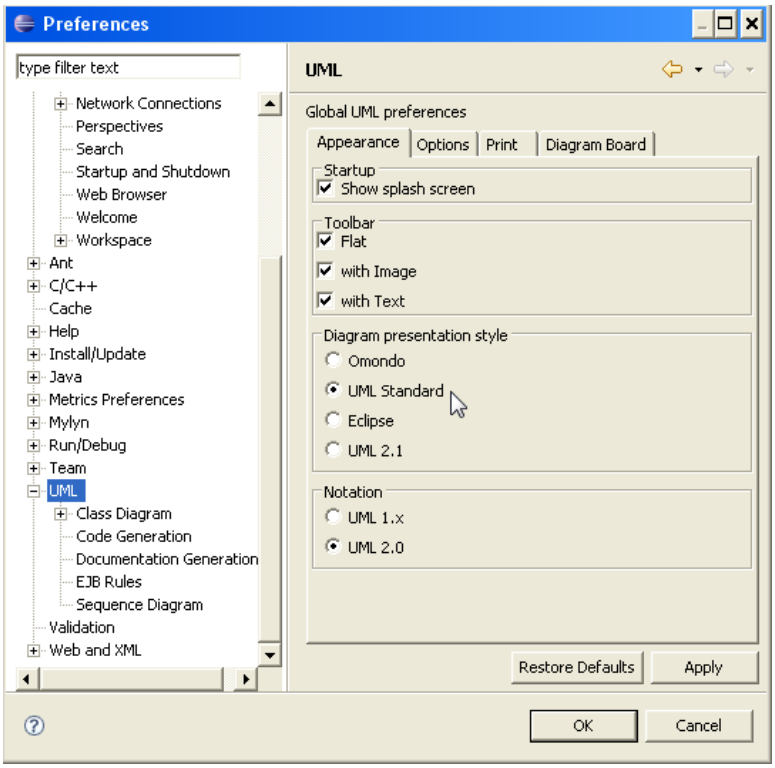
In dieser Notiz wird ein Klassendiagramm mit den drei Klassen Punkte, Linie und Polygon erstellt.

Nachdem man in Eclipse ein neues Java-Projekt angelegt hat, wird die weitere Entwicklung von Omondo aus gesteuert. Omondo unterstützt dabei auch das Round-Trip-Engineering, bei dem teilweise im Klassendiagramm und teilweise im Quellcode gearbeitet wird. Änderungen im Klassendiagramm oder im Quellcode werden im jeweils anderen Teil übernommen. Es ist allerdings ein sauberer Stil, die Erzeugung von Klassen, Exemplarvariablen und Methoden ausschließlich aus Omondo heraus zu steuern, da es beim Round-Trip doch leicht zu Inkonsistenzen führen kann. Die Probleme können in der Programmformatierung liegen, weiterhin ergänzt Omondo spezielle Kommentare, die nicht gelöscht werden dürfen.

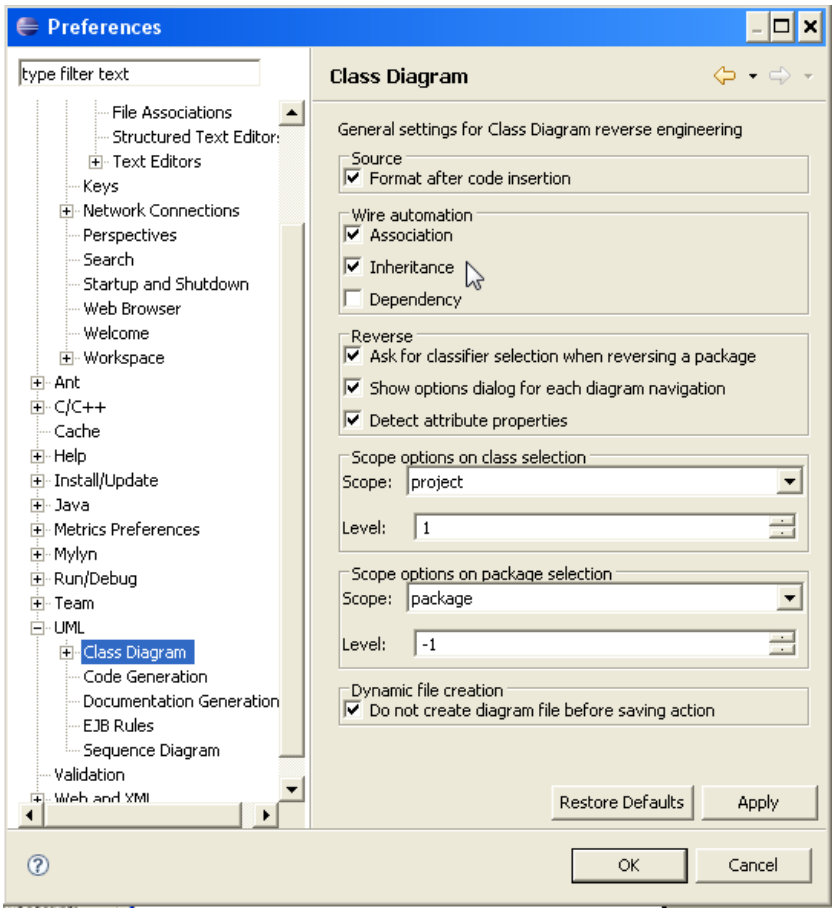
Vor der eigentlichen Nutzung von Omondo sollten einige Eigenschaften eingestellt werden. Wie alle generellen Eclipse-Eigenschaften werden diese unter „Window“ und dann „Preferences“ eingestellt.



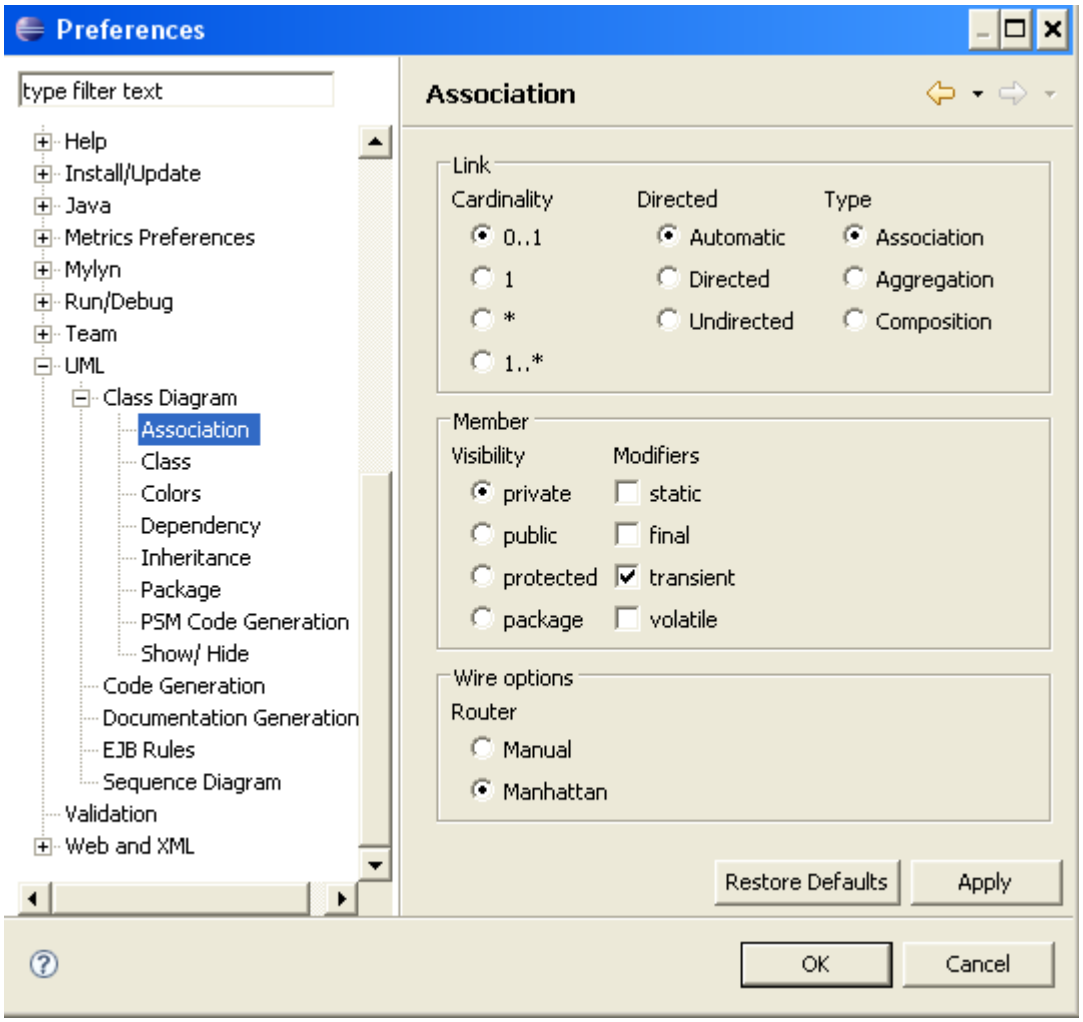
Die Einstellungen zu Omondo finden sich unter dem allgemeinen Punkt „UML“, hier sollten folgende Einstellungen unter dem Reiter „Appearance“ vorgenommen werden. Die Einstellungen bei den anderen Reitern können so gelassen werden.



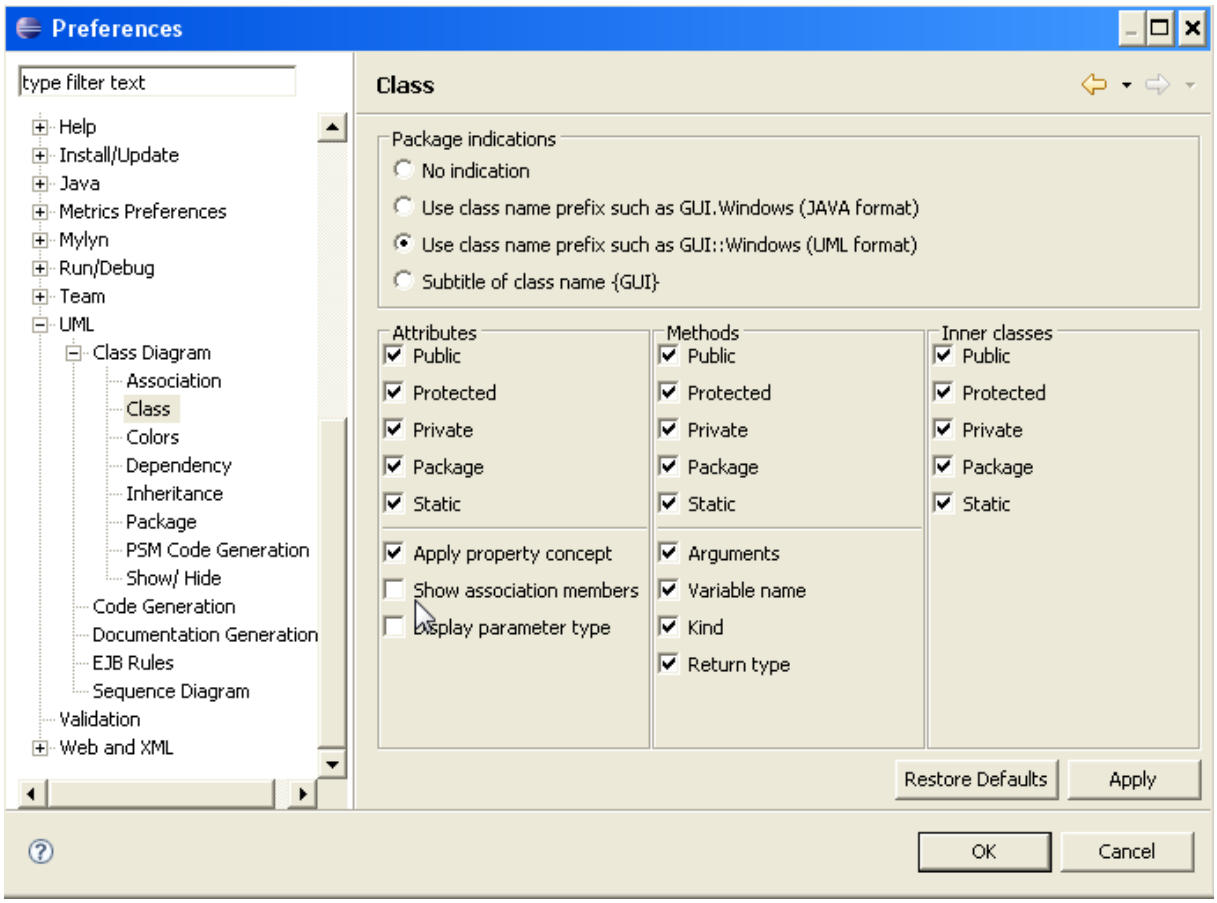
Die Einstellungen zum Unterpunkt “Class Diagramm“ können dem folgenden Bild entnommen werden.



Unter Association kann unter dem Punkt “Wire options“ eingestellt werden, ob man selbst das Layout übernehmen will, oder ob ein bestimmter Manhattan-Layout-Algorithmus zur Darstellung der Assoziationen genutzt werden soll. Ein Ansatz ist es, mit dem vorgegebenen Verfahren anzufangen und dies eventuell später zu ändern. Die benutzte Darstellungsart kann sogar individuell für Assoziationen geändert werden.

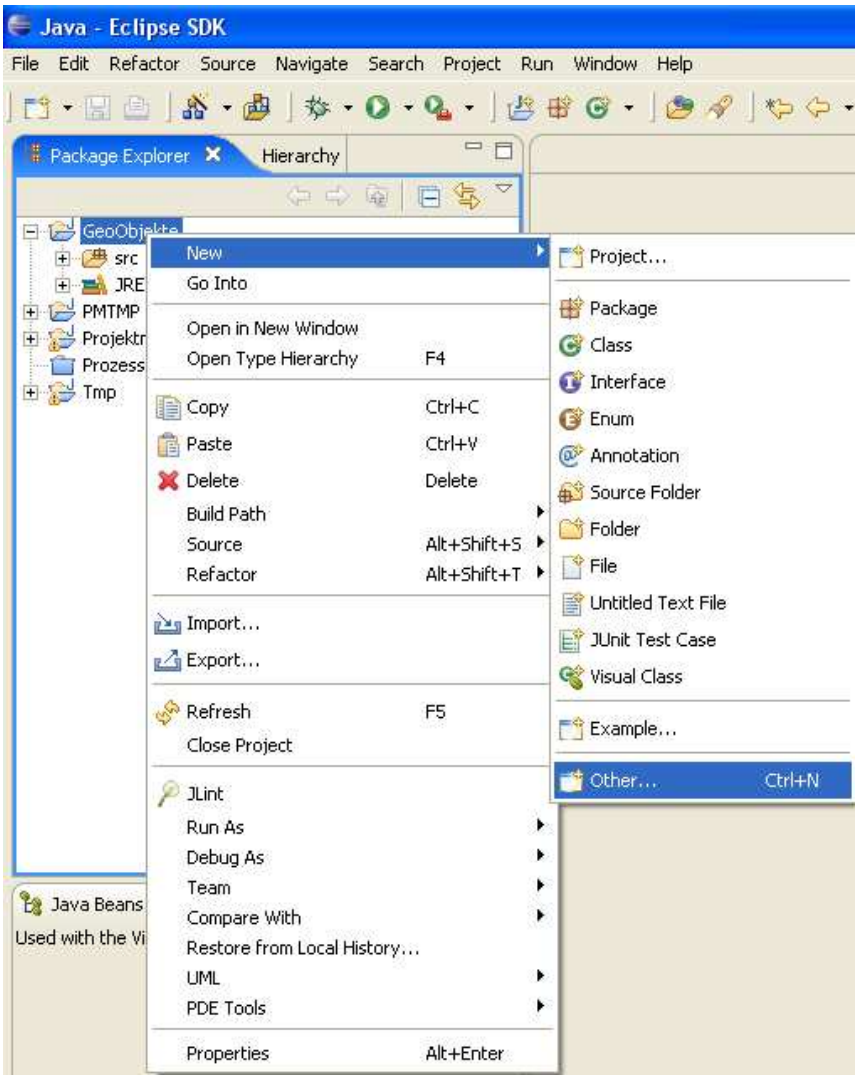


Beim Punkt Class müssen einige Haken gesetzt werden, die später individuell zur Darstellung von Klassen verändert werden können. Das so genannte „property concept“ sorgt dafür, dass get- und set-Methoden nicht explizit im Diagramm erscheinen. Dies hat Vor- und Nachteile abhängig davon, wie das Diagramm eingesetzt werden soll.

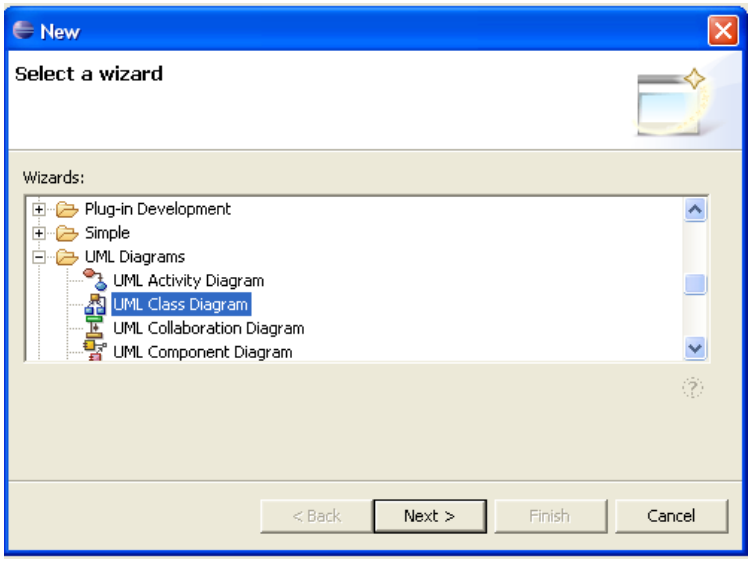


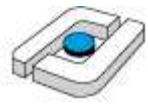
Bei den Punkten Dependency und Inheritance kann wieder das Layout-Verfahren angepasst werden.

Für die Spezifikation wird im neuen Java-Projekt GeoObjekte ein Klassendiagramm angelegt. Dazu wird nach einem Rechtsklick auf das Projekt im Projekt-Browser „New“ und dann „Other“ ausgewählt.

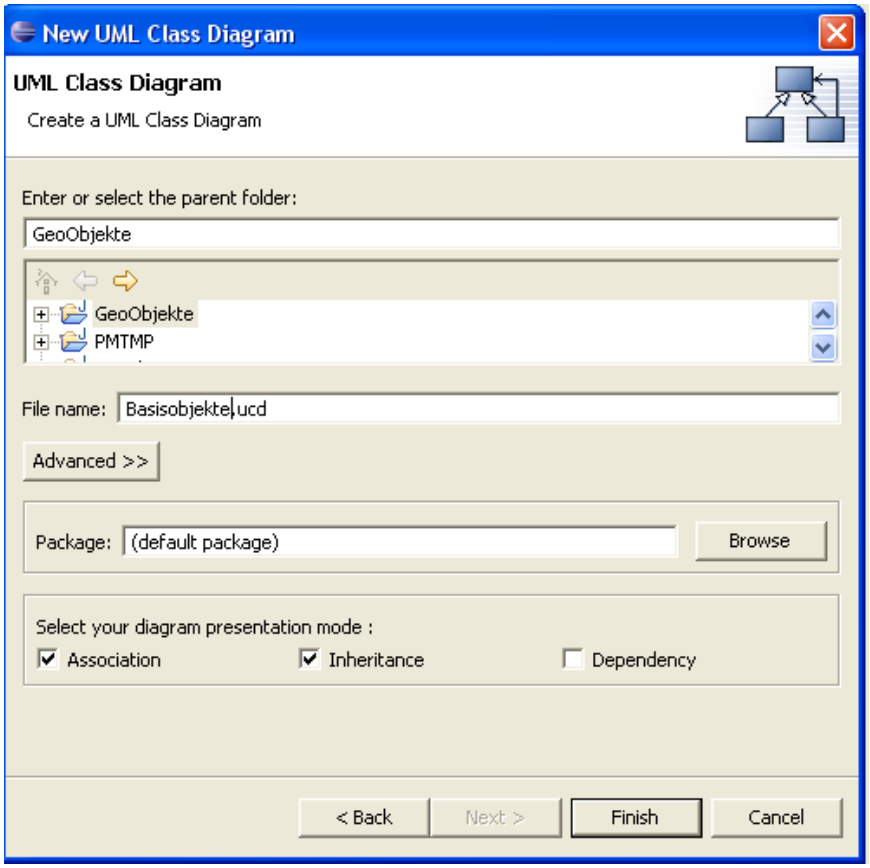


Im aufgehenden Fenster wird der Ordner „UML Diagrams“ geöffnet, „UML Class Diagramm“ ausgewählt und „Next“ gedrückt.

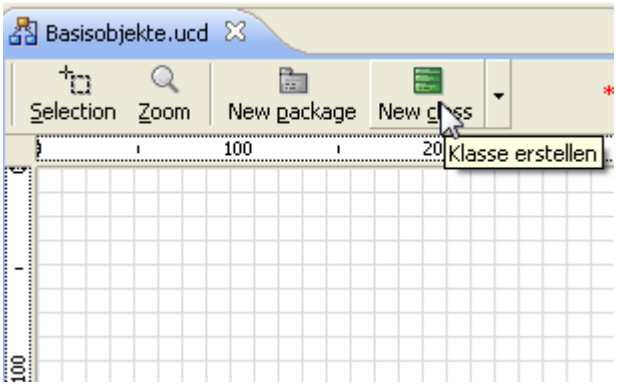


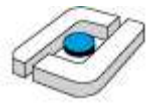


Im aufgehenden Fenster reicht es aus, bei „File Name“ einen sinnvollen Dateinamen einzugeben und die Haken in der letzten Zeile zu prüfen. Grundsätzlich besteht die Möglichkeit, dass Diagramme zu verschiedenen Paketen in den zugehörigen Ordnern entwickelt werden. Diese Möglichkeit sollen erfahrene Nutzer später anwenden. Die Diagramme sind, wie fast alles in Eclipse, später mit Drag and Drop verschiebbar. Die Eingabe wird mit „Finish“ beendet.

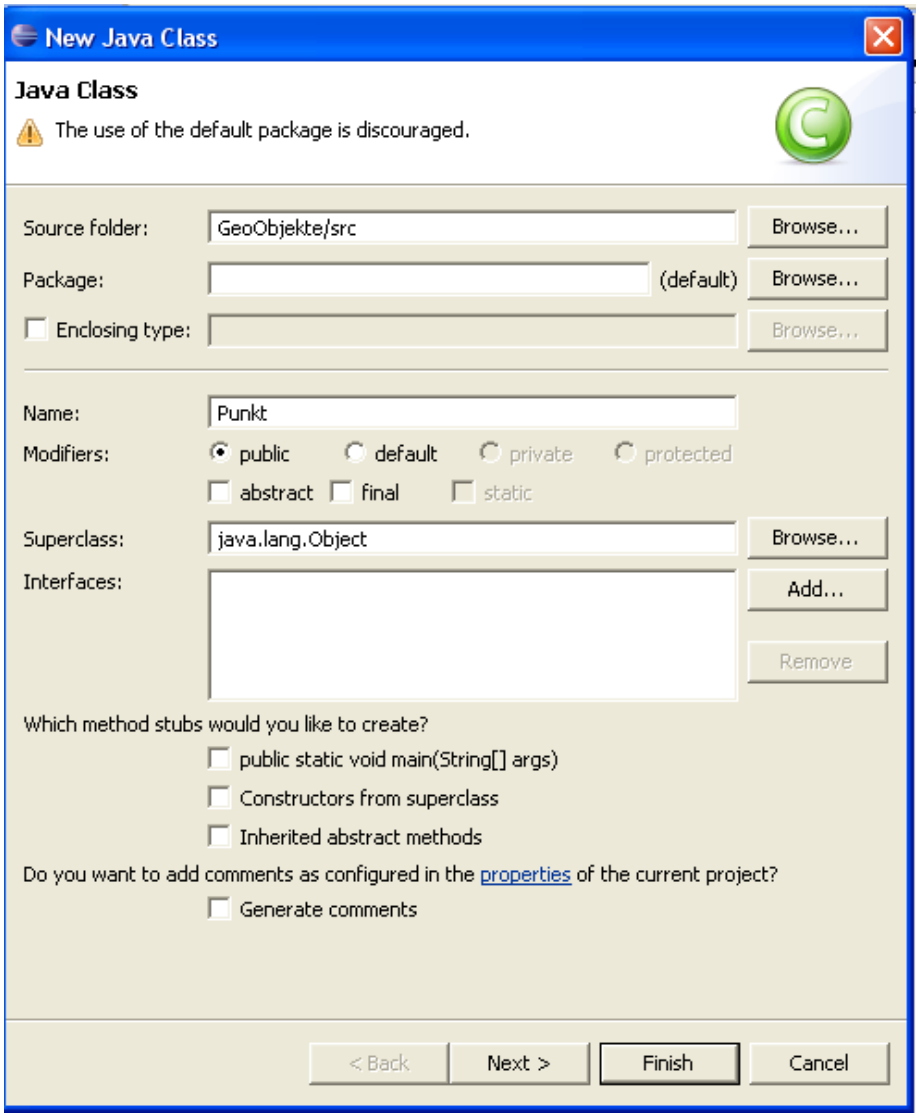


Die grundsätzliche Bedienung des Werkzeugs sieht für die Erstellung neuer Elemente so aus, dass die Art des neuen Elements in der Kopfzeile angeklickt wird und dass man dann durch einen Klick mit der Maus in der Arbeitsfläche das neue Element platziert. Für dieses Element wird dann ein Dialog zur Eingabe der Eigenschaften geöffnet. Im Beispiel wird „New Class“ gewählt. Da hier beim ersten Mal einiger Code geladen wird, kann dies etwas dauern.

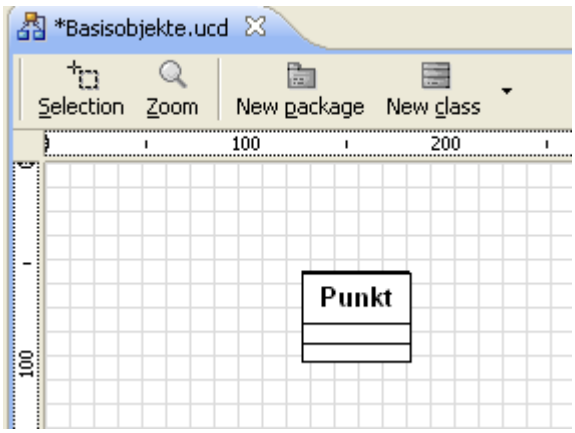


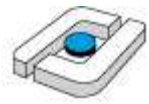


Danach erscheint der Eclipse-Standarddialog zur Erstellung einer Java-Klasse. Im Beispiel wird nur ein Klassenname eingegeben. Erfahrenere Java-Programmierer werden die Paketstruktur nutzen. Die Eingabe wird mit „Finish“ abgeschlossen.

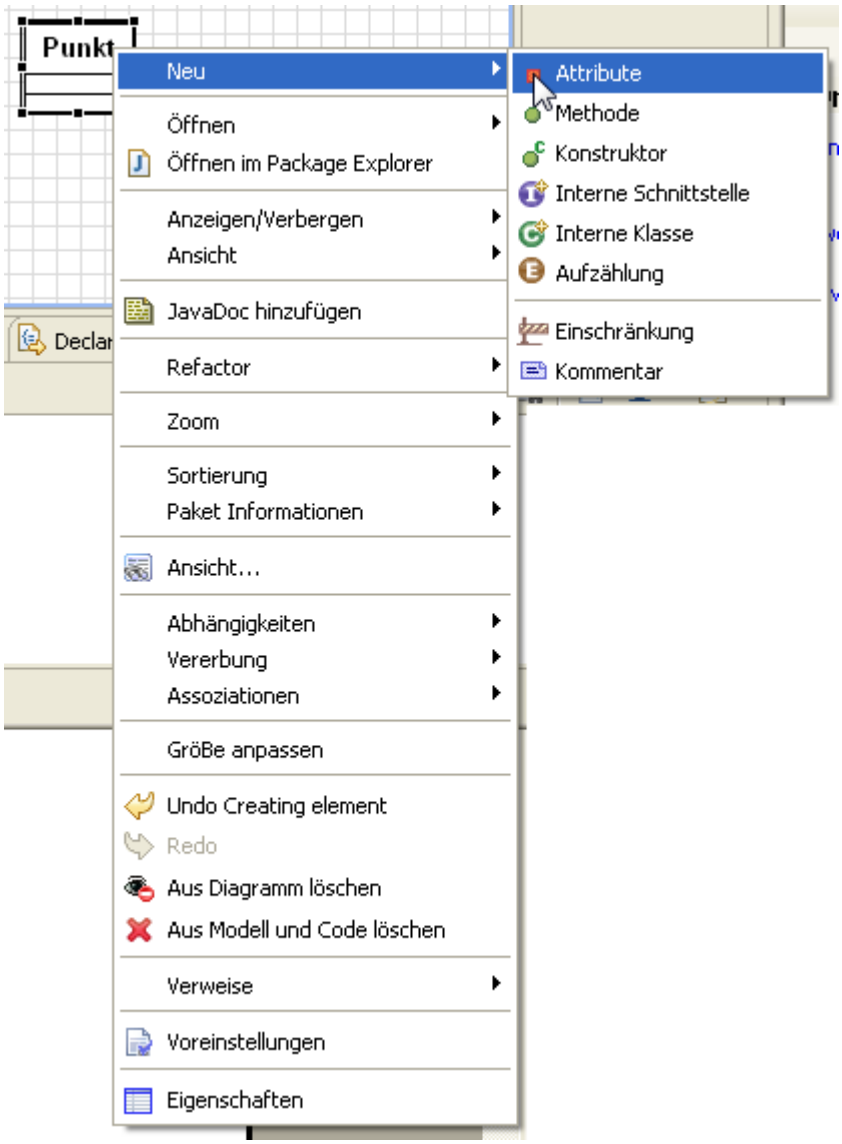


Man erhält eine einfache Klasse im Klassendiagramm.



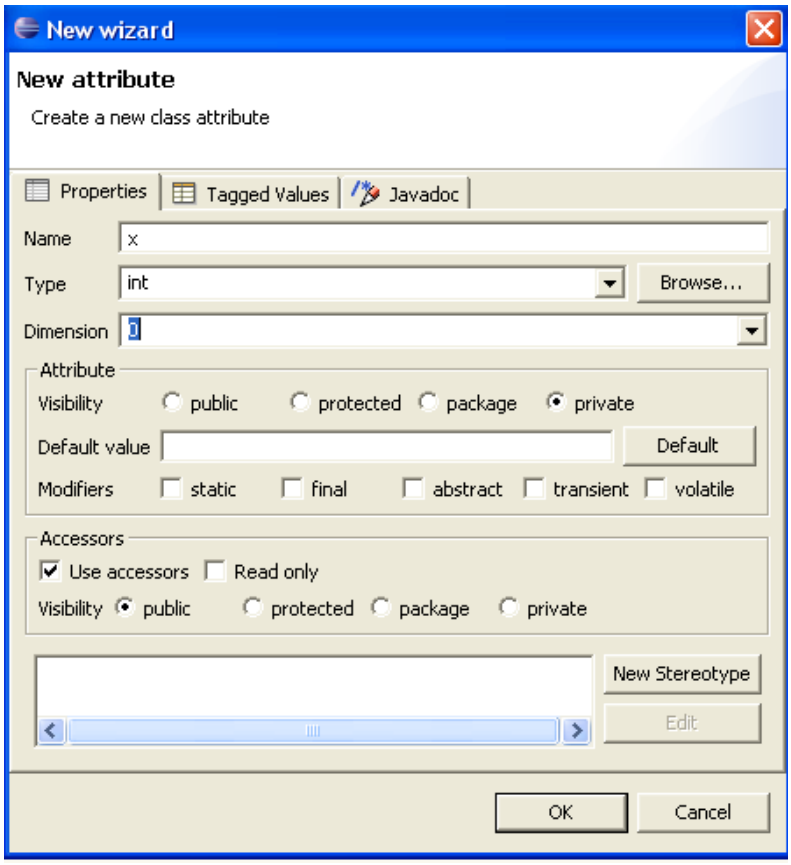


Durch einen Rechtsklick auf die Klasse werden die vielfältigen Bearbeitungsmöglichkeiten für Klassen in Omondo angezeigt, die gerade bei der Analyse komplexer Systeme, siehe z. B. „Dependencies“ (oder „Abhängigkeiten“ abhängig von der installierten Variante), sehr hilfreich sein können. Im konkreten Fall soll eine Exemplarvariable, die hier Attribut genannt wird, angelegt werden. Dazu wird der Punkt „Neu“ und dann „Attribute“ ausgewählt.

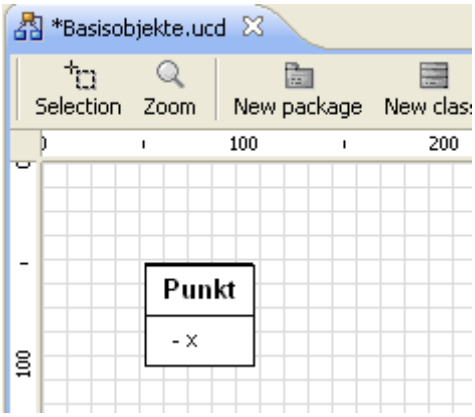


Im aufgehenden Fenster muss der Name und der Typ (Type) der Exemplarvariable angegeben werden. Im Abschnitt Accessors wird eingestellt, welche get- und set-Methoden automatisch erzeugt werden sollen. Mit „Use accessors“ wird eingestellt, dass es get- oder/und set-Methoden geben soll. Mit „Read only“ wird nur die get-Methode erzeugt.

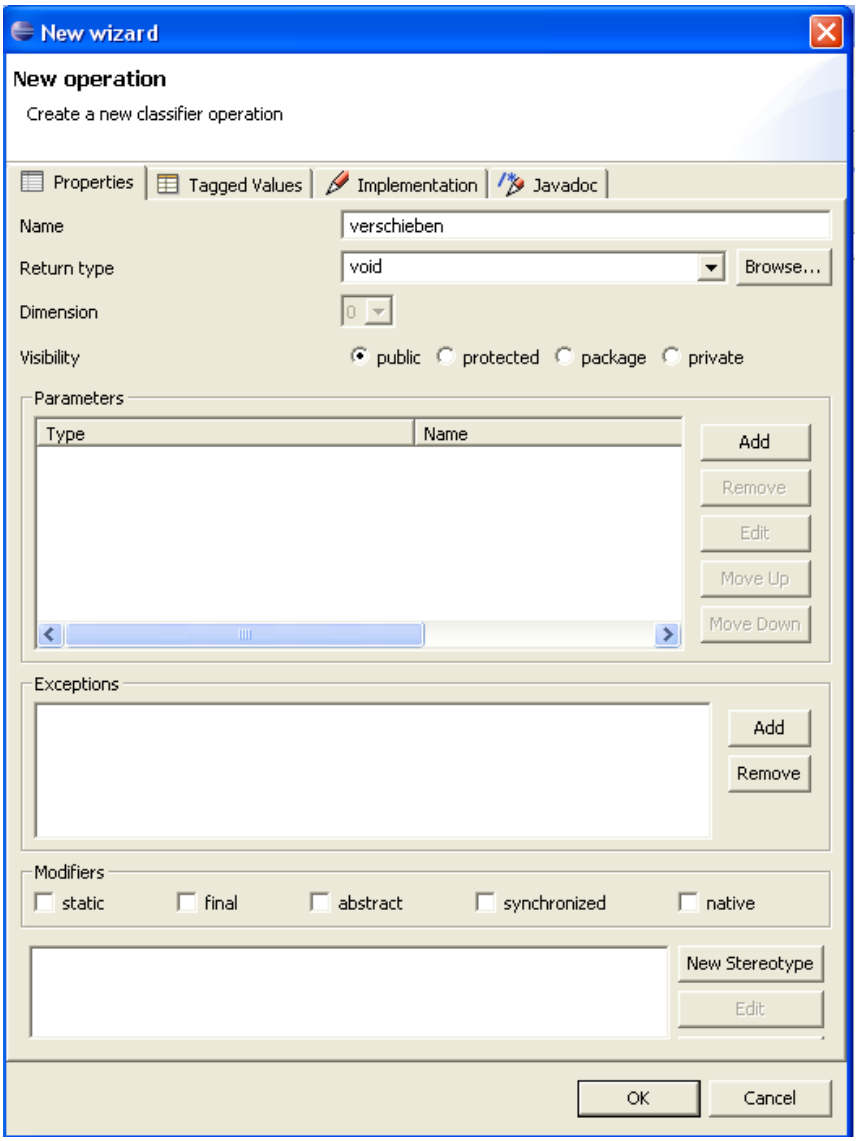
Mit Dimension wird festgelegt,ob es sich um einen Array handeln soll, dabei steht der Wert eins für einen eindimensionalen Array x[] und ein Wert drei für einen dreidimensionalen Array x[][][]. Die weiteren Einstellungsmöglichkeiten sollten durch die Java-Vorlesung inhaltlich bekannt sein. Unter dem Reiter „Javadoc“ kann ein Kommentar zur Exemplarvariable stehen, was bei großen Projekten immer erforderlich ist.



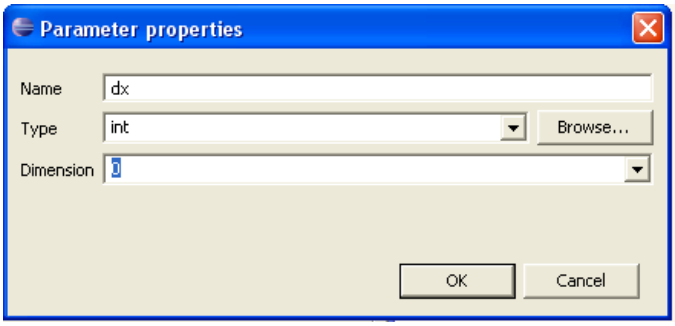
Im Klassendiagramm wird die neue Exemplarvariable sichtbar, wobei auf die Angabe von generierten get- und set-Methoden verzichtet wird. (Nimmt man Java-Quellcode und lässt daraus ein Klassendiagramm erstellen, muss dies leider nicht der Fall sein, weiterhin sollte der Typ sichtbar sein, ist es aber anscheinend nicht)



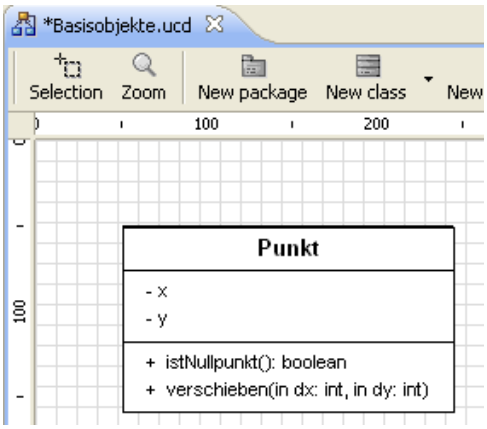
Im nächsten Schritt wird eine neue Methode ergänzt. Dazu wird im Bearbeitungsmenü „Neu“ und dann „Methode“ ausgewählt. Im dann aufgehenden Fenster muss der Methodenname (Name) und der Rückgabetyp (Return type) angegeben werden.



Um Parameter für die Methode zu ergänzen, muss im Abschnitt „Parameters“ der „Add“-Knopf gedrückt werden. Dabei öffnet sich das folgende Fenster, in dem der Parametername und sein Typ angegeben werden. Die Eingabe endet mit „Ok“.

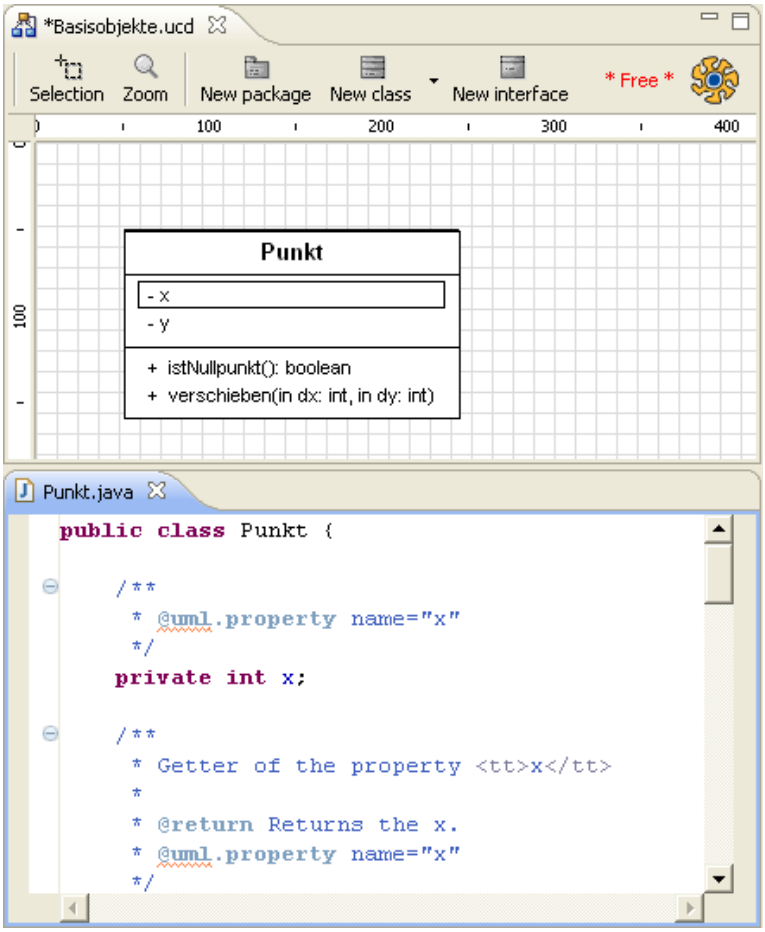


Nachdem alle Parameter und weitere Methodeneigenschaften eingegeben wurden, wird die Methodenerstellung mit „Ok“ abgeschlossen. Im folgenden Klassendiagramm wurde eine weitere Methode ergänzt.



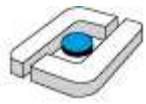
Generell müssen Diagrammänderungen gespeichert werden. Wie in Eclipse üblich, zeigt ein kleiner Stern links oben beim Dateinamen, dass eine Datei noch nicht gespeichert wurde.

Macht man einen Doppelklick auf die Klasse, eine Exemplarvariable oder Methode, so öffnet sich der zugehörige Quellcode. Dabei ist es die Standardeinstellung, dass das Fenster halbiert wird.

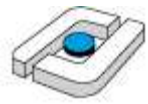


Generell gilt, dass Omondo das Programmgerüst erstellt und der Entwickler dann in den vorgesehenen Abschnitten den Programmcode ergänzt. Der von Omondo im Beispiel generierte Programmcode sieht wie folgt aus:

```
public class Punkt {
    /**
     * @uml.property name="x"
```

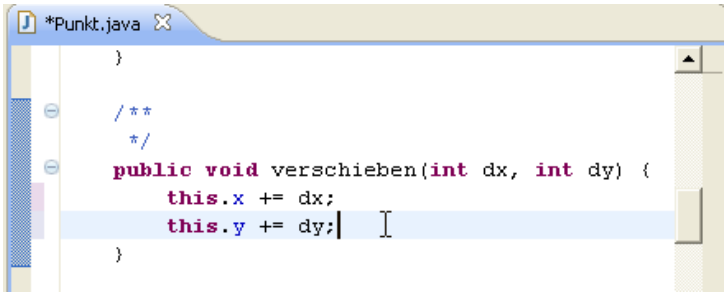


```
 */
private int x;

/**
 * Getter of the property <tt>x</tt>
 *
 * @return Returns the x.
 * @uml.property name="x"
 */
public int getX() {
	return x;
}

/**
 * Setter of the property <tt>x</tt>
 *
 * @param x
 *            The x to set.
 * @uml.property name="x"
 */
public void setX(int x) {
	this.x = x;
}

/**
 */
public void verschieben(int dx, int dy) {

}

/**
 * @uml.property name="y"
 */
private int y;

/**
 * Getter of the property <tt>y</tt>
 *
 * @return Returns the y.
 * @uml.property name="y"
 */
public int getY() {
	return y;
}

/**
 * Setter of the property <tt>y</tt>
 *
 * @param y
 *            The y to set.
 * @uml.property name="y"
 */
public void setY(int y) {
```

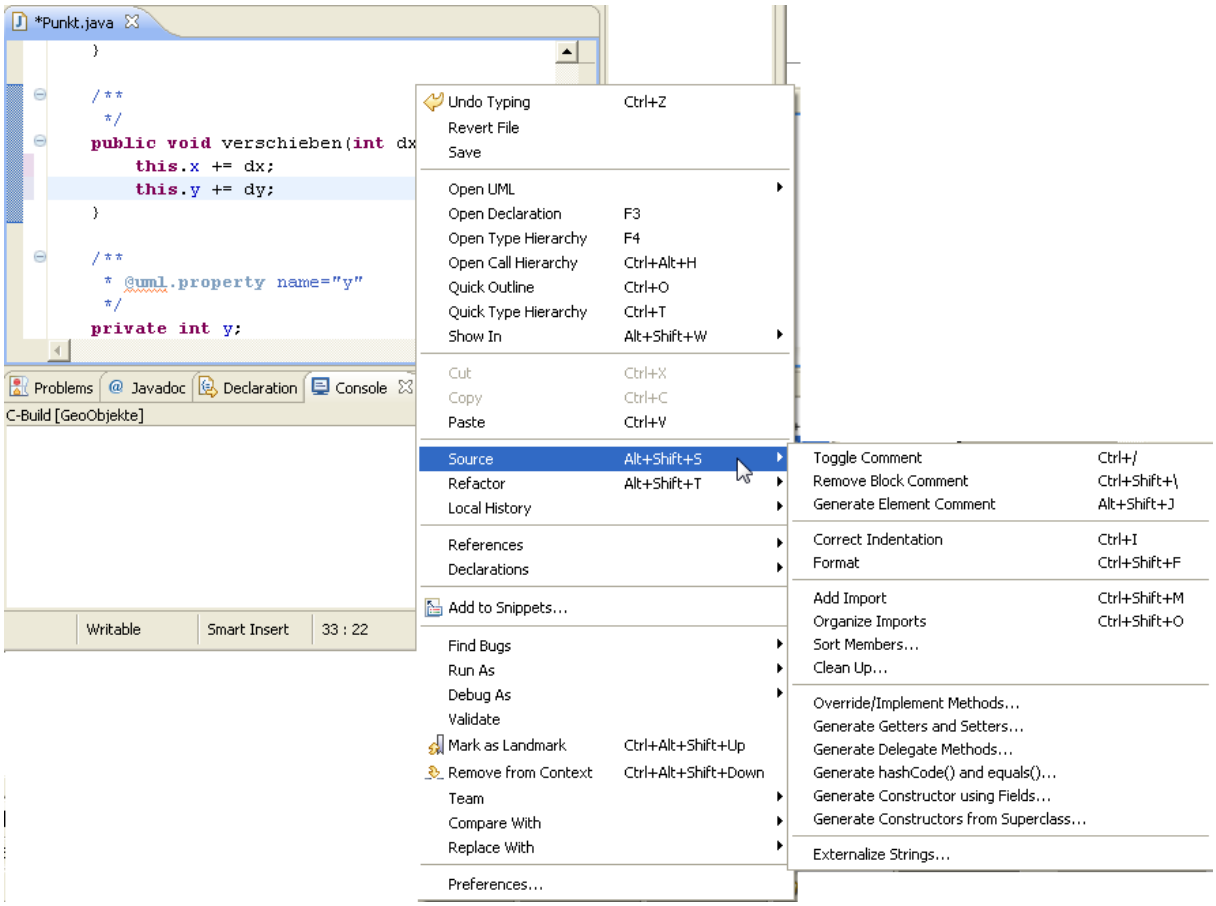


```
            this.y = y;
        }

        /**
         */
        public boolean istNullpunkt() {
            return false;
        }
    }
```

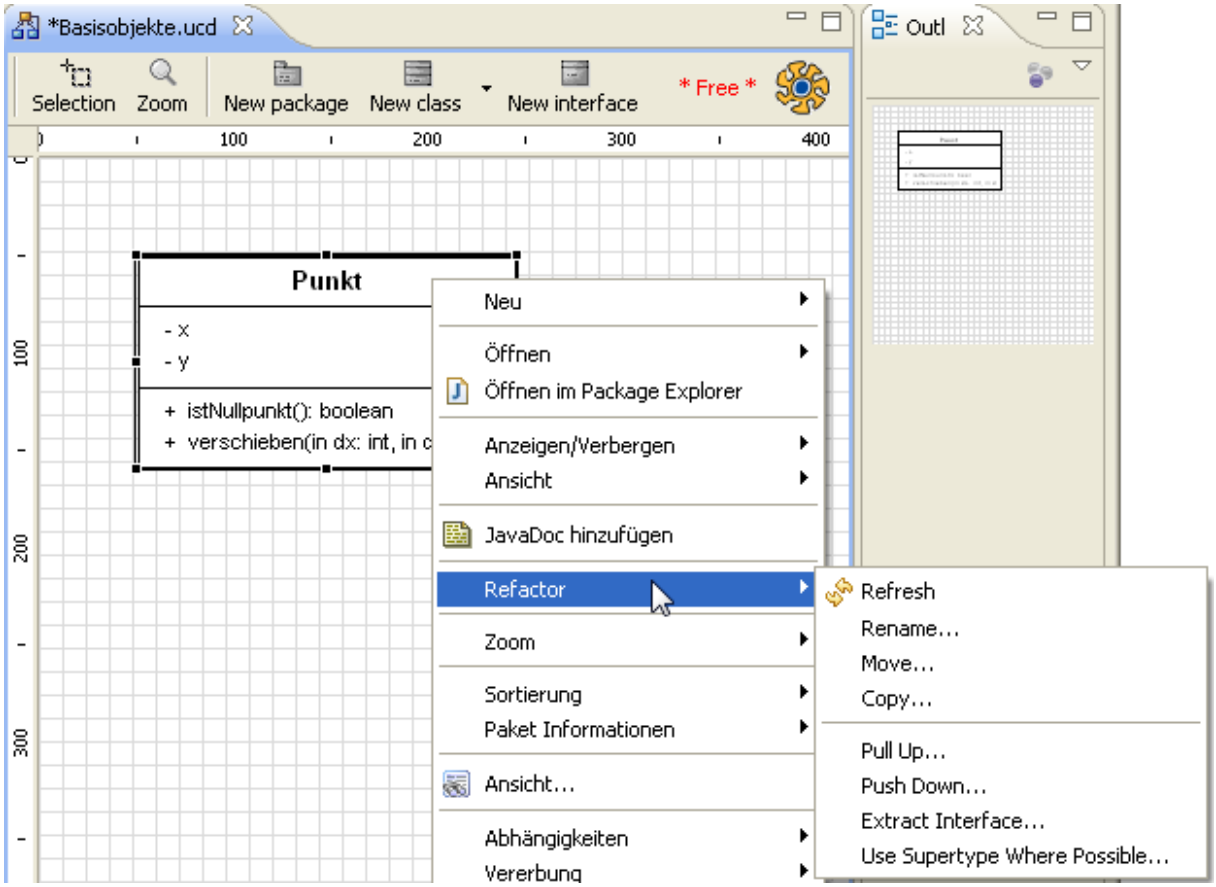
Wichtig ist, dass die so genannten Tags in den Kommentaren, z. B. „@uml.property“ nicht geändert werden. Der Entwickler ergänzt dann z. B. folgenden Programmcode.



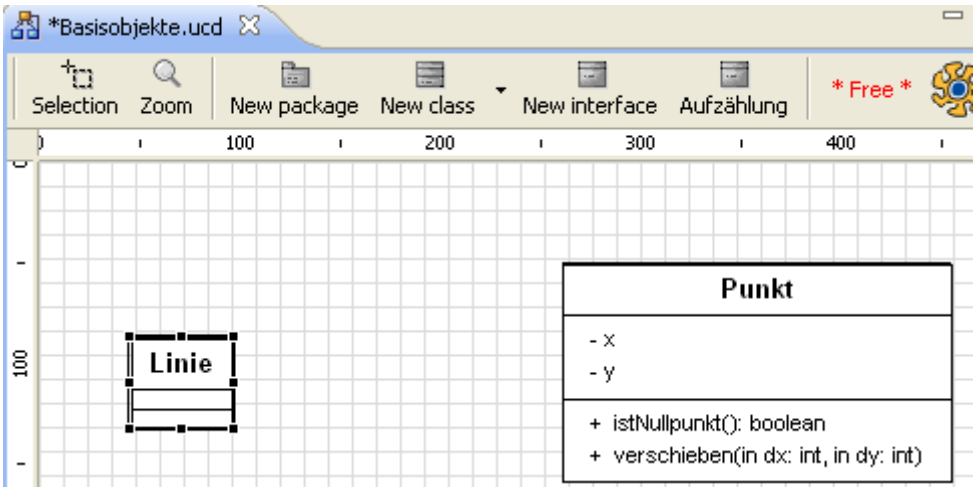
Da Omondo ab und zu einige unglückliche Einrückungen macht, kann auch die Code-Formatierung von Eclipse genutzt werden. Dazu wird im Editor ein Rechtsklick ausgeführt, dann „Source“ und „Format“ ausgewählt. Für Eclipse-Neulinge sei auf die vielfältigen Möglichkeiten hingewiesen, die noch in dem „Source“-Menü enthalten sind.



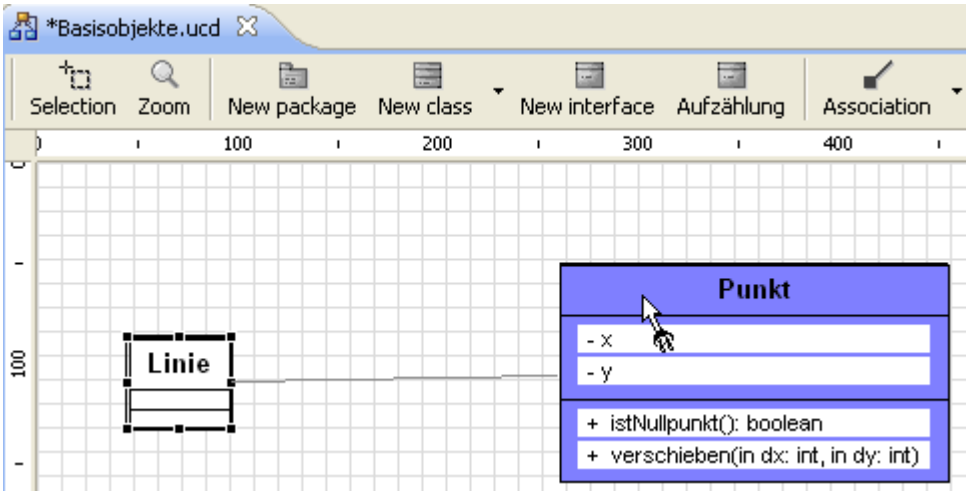
Möchte man die Eigenschaft einer Klasse, einer Exemplarvariable oder einer Methode ändern, so wird diese im Klassendiagramm ausgewählt und ein Menü über Rechtsklick geöffnet. In diesen individuellen Menüs gibt es immer einen Punkt „Refactor“, der, genau wie generell in Eclipse, die Änderung von Eigenschaften ermöglicht.



Im nächsten Schritt wird eine Klasse Linie implementiert, wobei eine Linie durch einen Start- und einen Endpunkt beschrieben wird. Grundsätzlich könnte man den Startpunkt genau wie vorher als Exemplarvariable definieren. Leider wird dann in EclipseUML keine Assoziation zur Klasse Punkt angezeigt. (Lässt man ein neues Diagramm generieren oder wählt unter „Assoziationen“ dann „show association“, ist diese Anzeige möglich.)

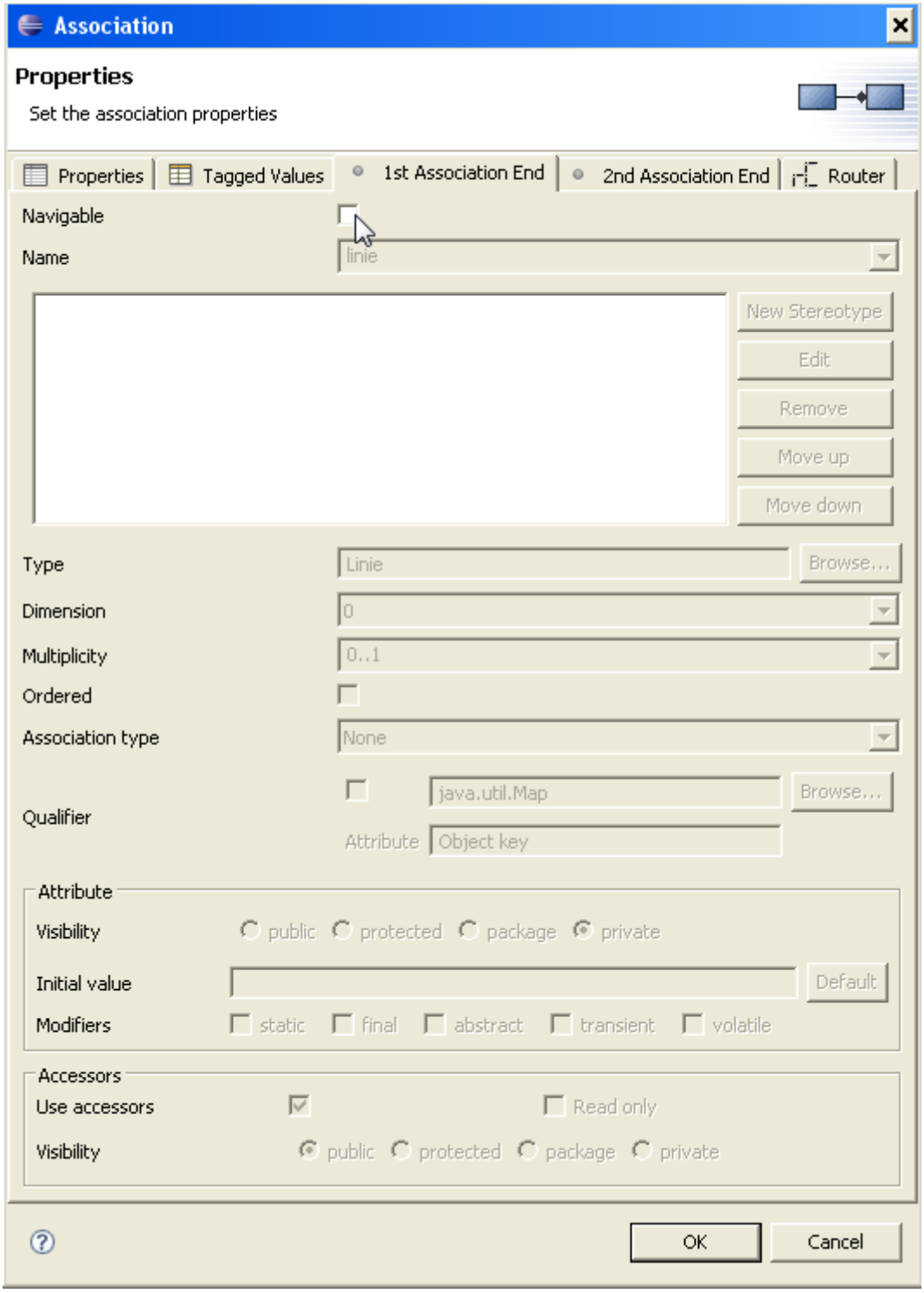


Aus diesem Grund wird der Startpunkt als Assoziation eingetragen. Dazu wird oben „Assoziation“ ausgewählt und dann auf die Startklasse, genauer den Klassennamen und kein anderes Teil der ausgehenden Klasse Linie und dann auf die Zielklasse Punkt, genauer den Klassennamen, geklickt.

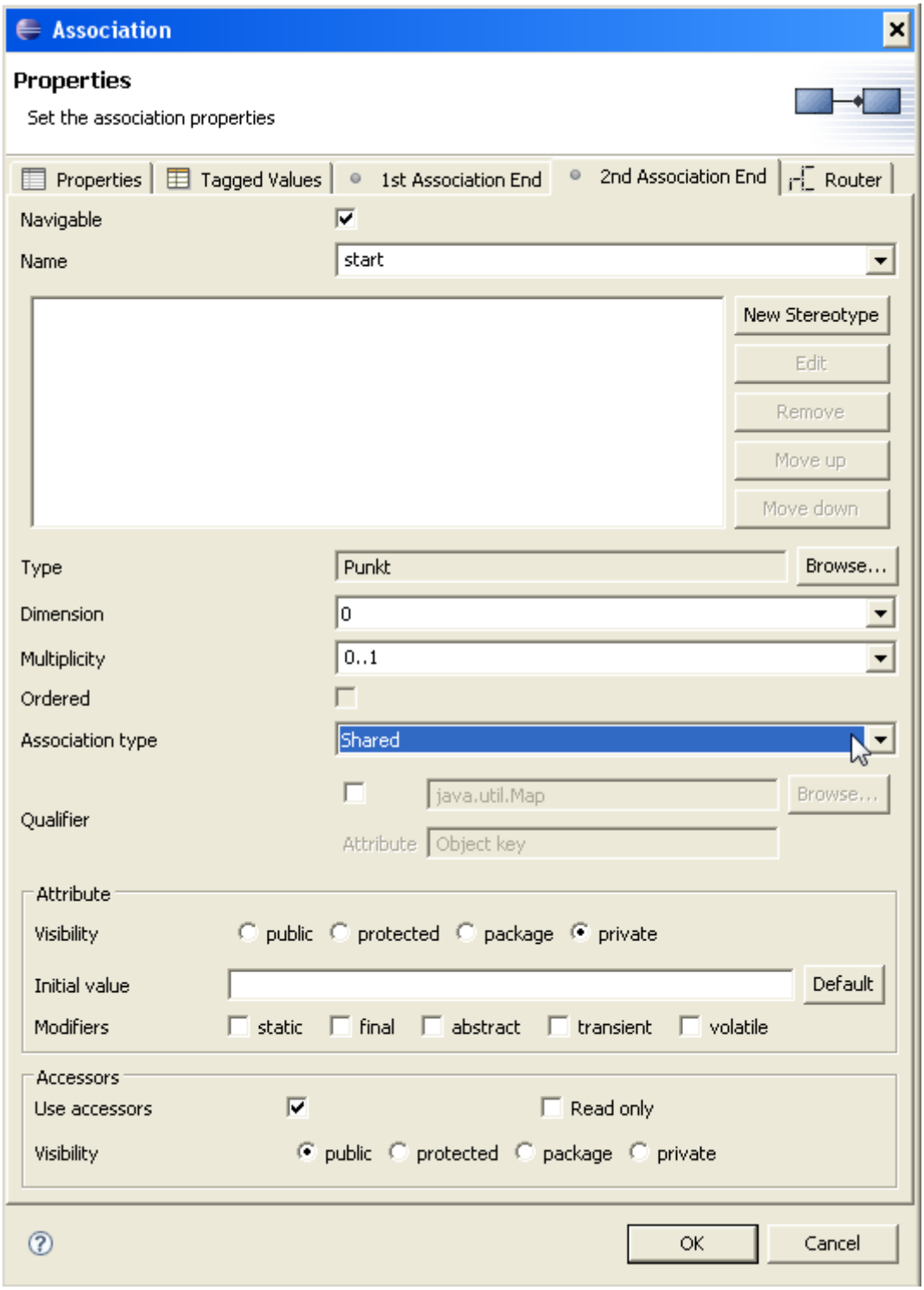


Es öffnet sich ein Fenster, in dem die Eigenschaften der Assoziation eingegeben werden sollen. Wie für Java typisch, soll die Linie eine Referenz auf den Punkt erhalten, wobei der Punkt nicht ausschließlich zur Linie gehören muss. Man spricht in der UML von einer Aggregation.

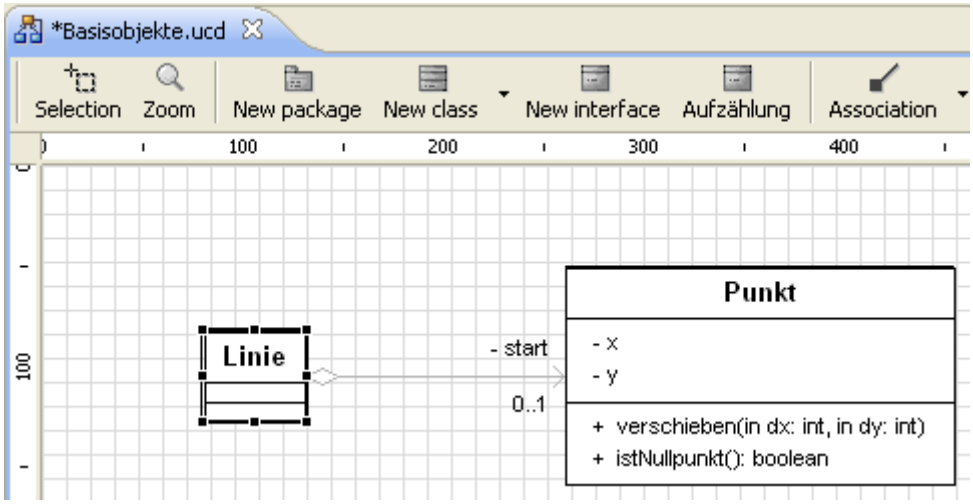
Interessant ist zunächst der Reiter „1st Association End“, dabei geht es um die Rolle der Linie in der Assoziation aus Sicht des Punktes. Da der Punkt nicht wissen soll, zu welcher Linie er gehört, wird einfach der Haken „Navigable“ weggenommen, wodurch hier keine weiteren Eingaben mehr möglich sind.



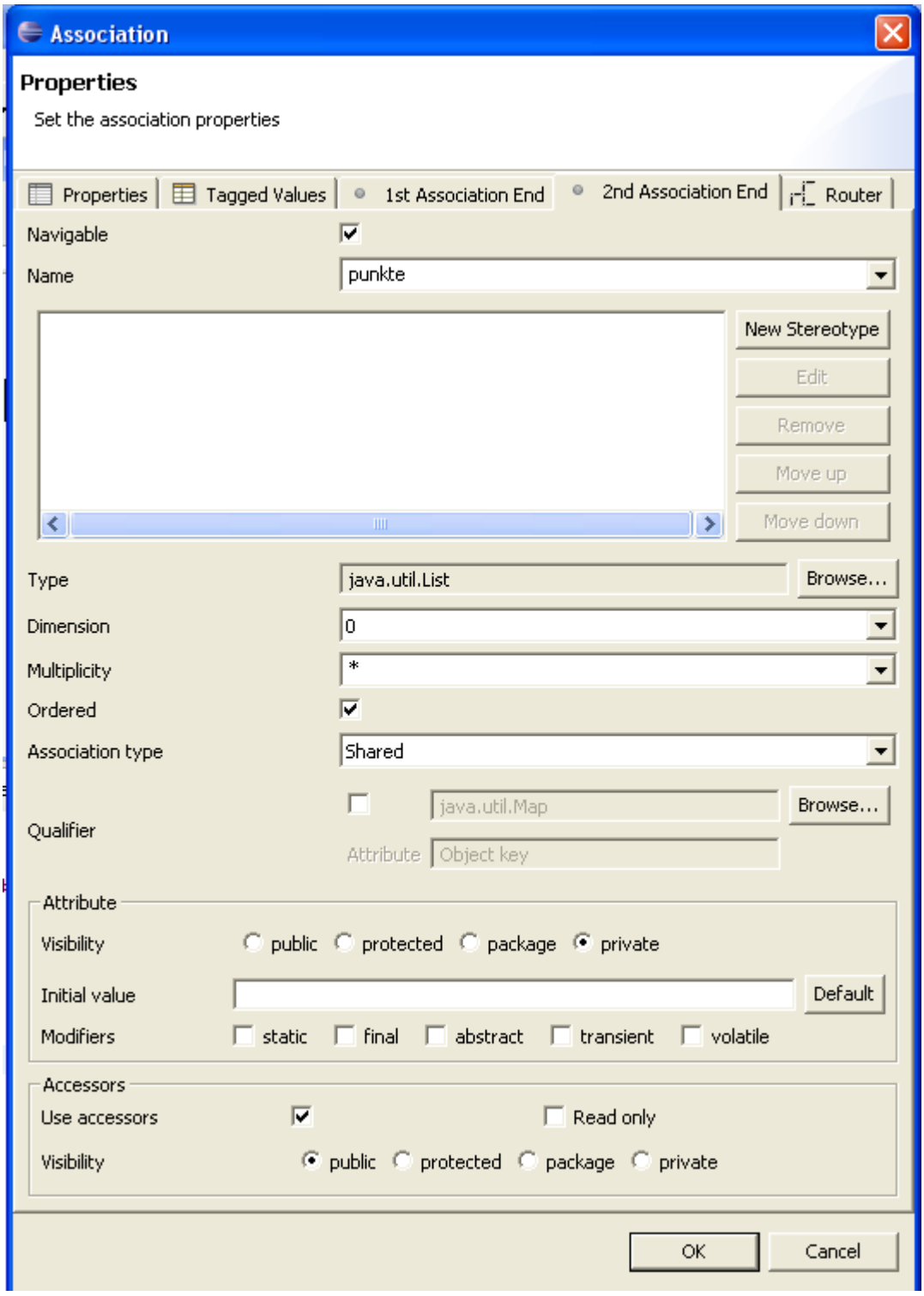
Beim „2nd Association End“ wird die Rolle des Punktes aus Sicht der Linie eingetragen. Dazu wird zunächst der Name der Exemplarvariablen start eingegeben. Die „Multiplicity“ für die Kardinalität kann im Beispiel so gelassen werden. Der „Association type“ wird auf „Shared“ gesetzt, was in der UML einer Aggregation entspricht. Für die durch die Assoziation definierte Exemplarvariable können unter „Attribute“ und „Accessors“ die Eigenschaften der Exemplarvariablen definiert werden. Bei „Accessors“ geht es wieder um mögliche Get- und Set-Methoden.



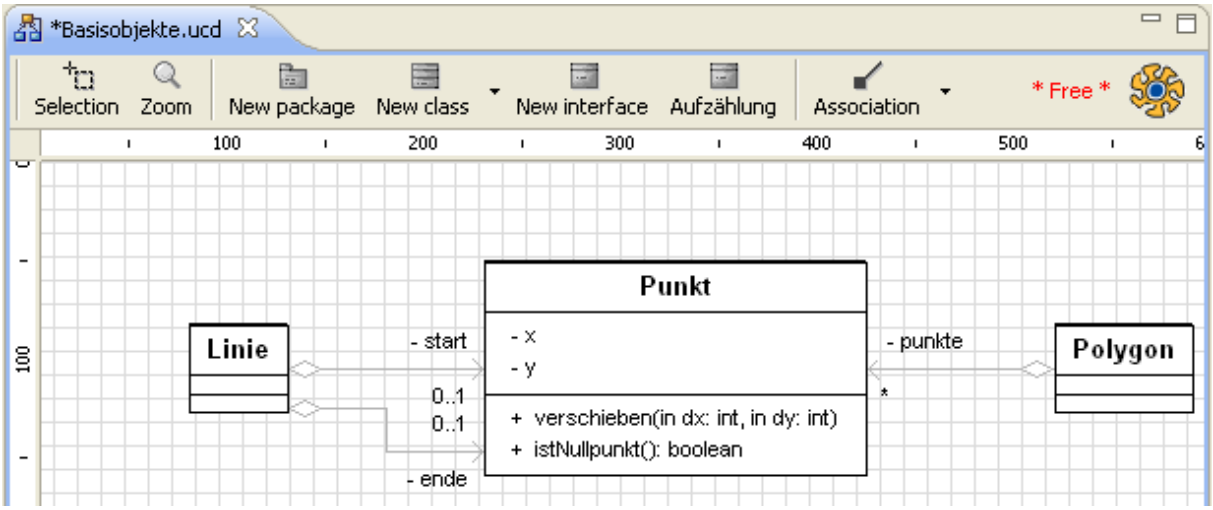
Nachdem die Eingabe mit „Ok“ abgeschlossen wurde, ist die Assoziation im Diagramm sichtbar.



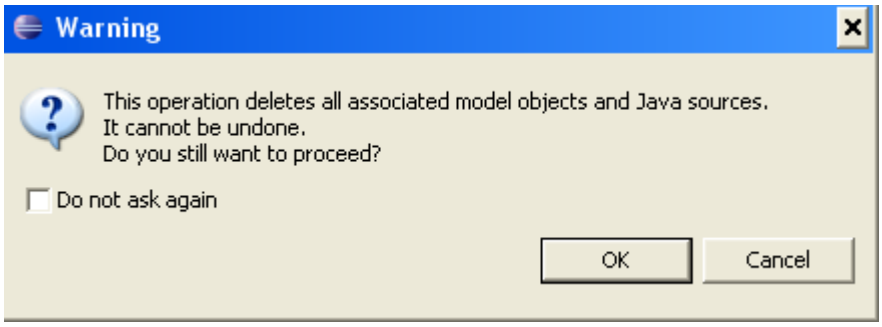
Im nächsten Beispiel wird eine Klasse Polygon angelegt, wobei ein Polygon von mehreren Punkten begrenzt wird. Grundsätzlich ist das Vorgehen identisch mit der vorherigen Eingabe einer Assoziation, nur einige Details müssen geändert werden. Dabei wird die „Multiplicity“ auf „*“ gesetzt. Man beachte, dass sich dabei der Eintrag „Type“ von Punkt auf „java.util.Collection“ ändert. Für die Art der benutzten Sammlung ist es noch wichtig, ob die Reihenfolge der Elemente eine Rolle spielt. Ist dies der Fall, muss der Haken bei „Ordered“ gesetzt sein. Durch den Knopf „Browse“ beim Eintrag „Type“ kann die Art der genutzten Sammlung konkretisiert werden und z. B. das Interface Collection durch ArrayList ersetzt werden. Da man bei Typen von Exemplarvariablen möglichst allgemein bleiben sollte, kann der ursprüngliche Eintrag stehen gelassen werden.



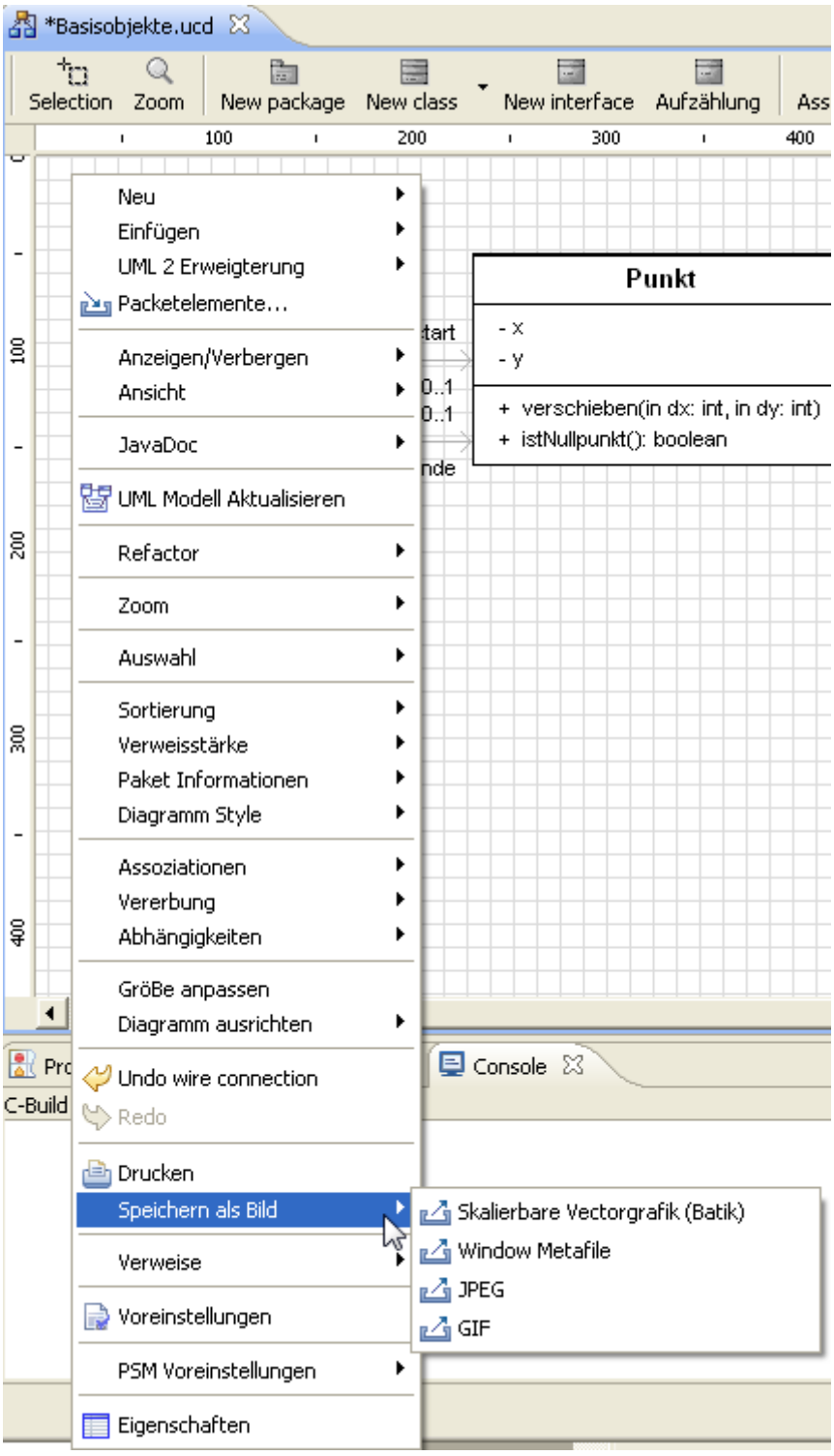
Das resultierende Diagramm sieht wie folgt aus.



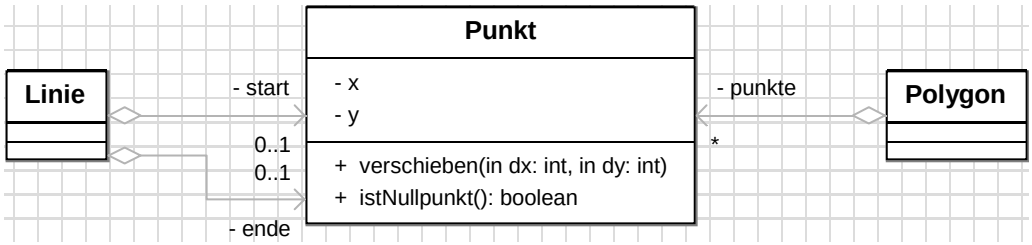
Warnung: Man kann im Diagramm Elemente löschen. Da das Diagramm mit dem Java-Code verknüpft ist, bedeutet dies, dass damit auch der Java-Code gelöscht wird. Will man Klassen in Klassendiagrammen nicht mehr sehen, so kann im Menü „Hide“ gewählt werden.



Möchte man ein Diagramm als Bild zur Nutzung in Dokumenten exportieren, so macht man einen Rechtsklick neben dem Diagramm und wählt „Speichern als Bild“ aus.



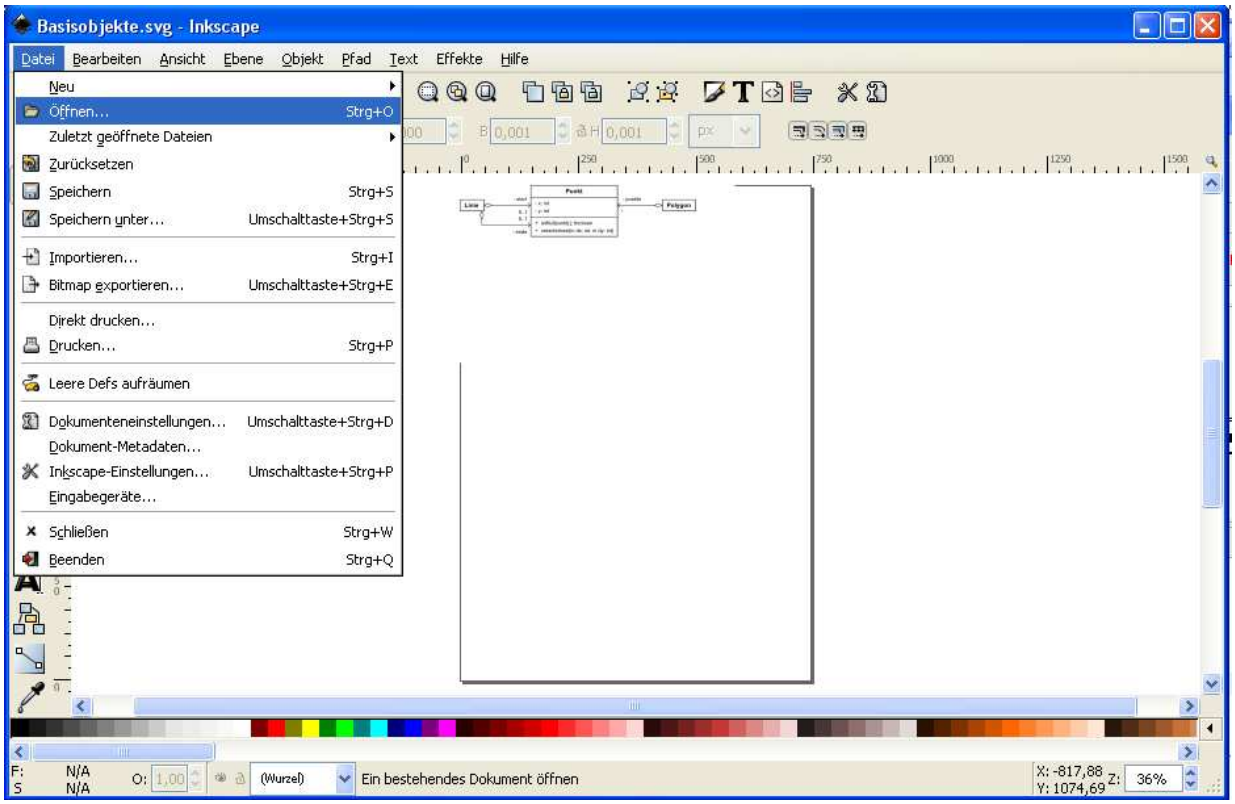
Die Typen WMF, JPG und GIF, die z. B. in Präsentationen genutzt werden können, haben allerdings das Karo-Raster im Hintergrund, wie folgendes WMF-Bild zeigt.



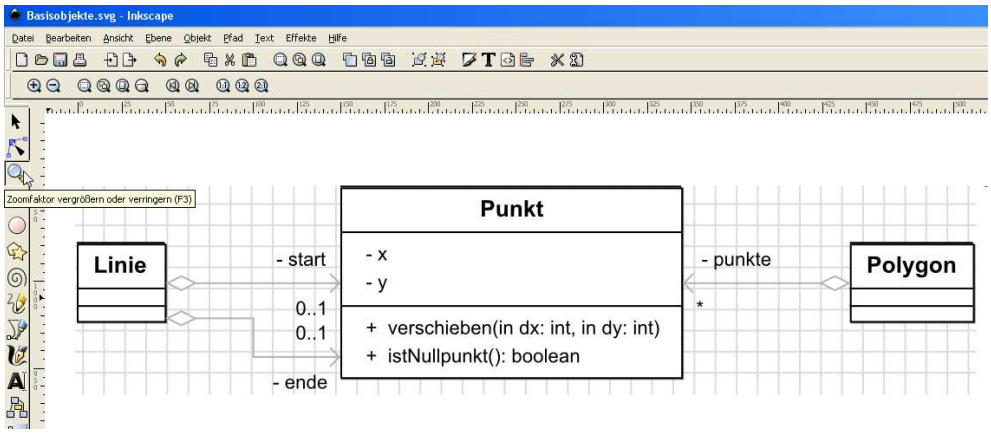
Alternativ steht „Scalable Vector Graphics (Batik)“ zur Verfügung.

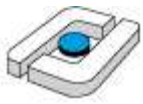
Da viele Programme leider nicht oder nicht ordentlich mit svg-Graphiken umgehen können, kann es sinnvoll sein, die Vektorgraphik in eine Bitmap-Graphik zu verwandeln. Eine Lösung ist die Installation von Inkscape (http://www.inkscape.org/), mit dem svg-Dateien gelesen und bearbeitet werden können.

Nach dem Einlesen der svg-Datei über „Datei“ und „Öffnen...“ steht das Bild zur Verfügung.

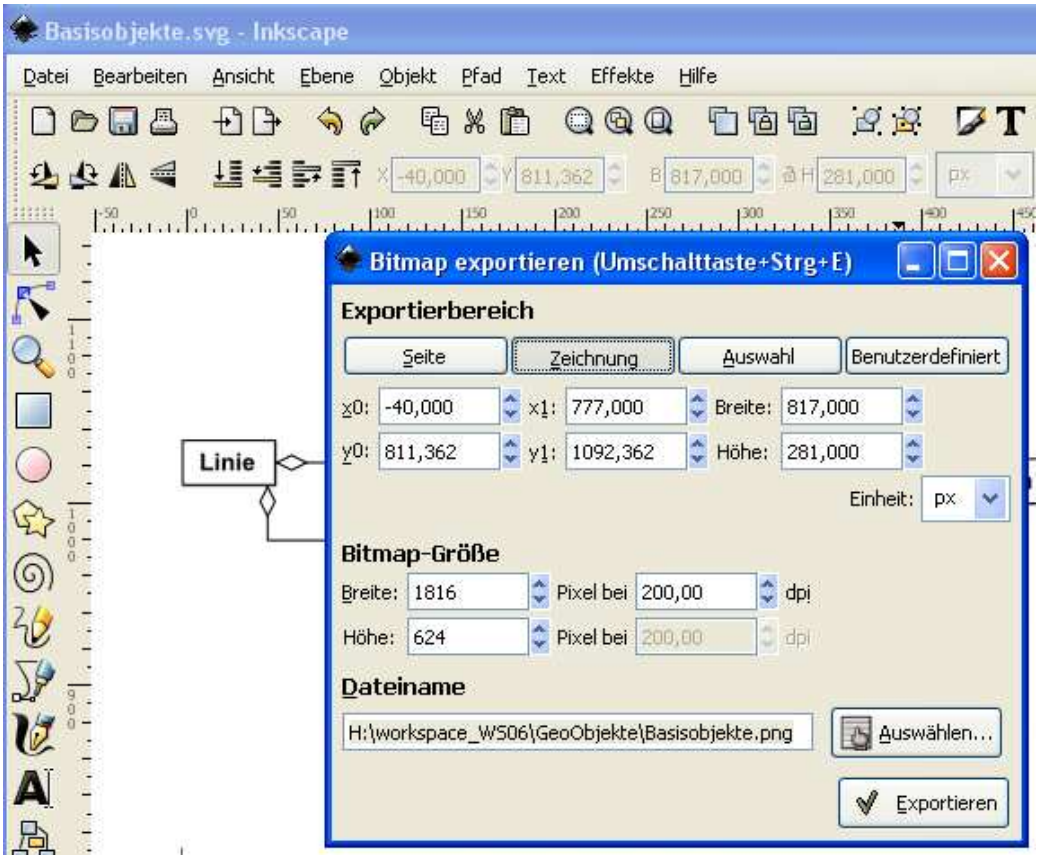


Leider gehört auch hier der Hintergrund zur Datei.

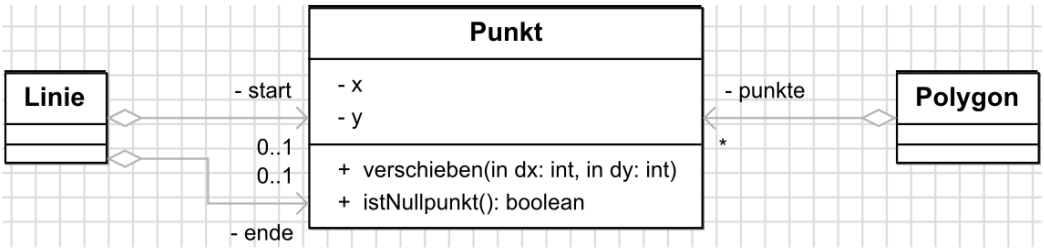


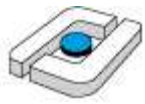


Unter „Datei“ und „Bitmap exportieren“ exportieren kann eine png-Datei erzeugt werden, dabei sollte man den Wert „Pixel bei“ mindestens auf 200 setzen und die obige Auswahl auf „Zeichnung“ setzen. Etwaige unschöne Ränder können dann mit einem weiteren Graphik-Werkzeug oder mit etwas Erfahrung auch in Inkscape weggeschnitten werden.



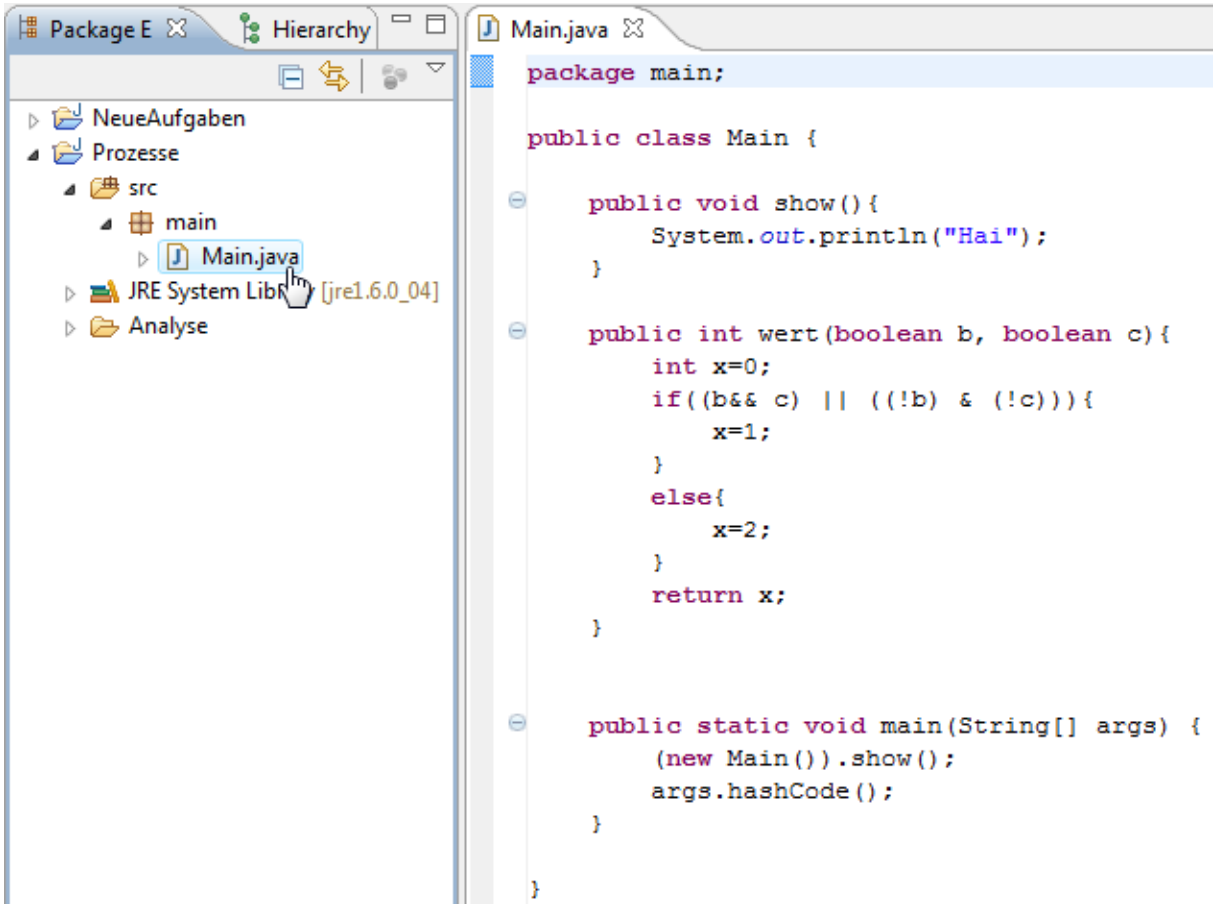
Man beachte, dass nach dem Klicken von „Exportieren“ das Fenster nicht schließt. Es kann, wie für eine Unix-Applikationen typisch, für weitere Arbeiten offen gelassen werden. Ein exportiertes Ergebnis kann dann wie folgt aussehen.



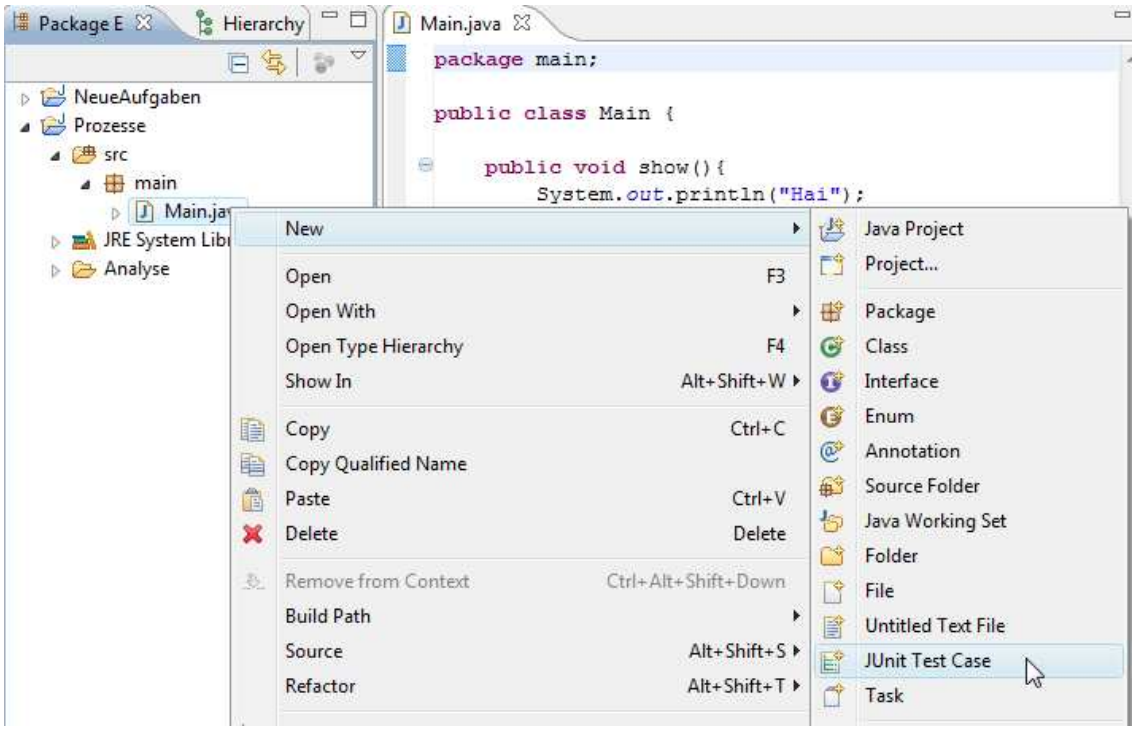


13 JUnit und Testüberdeckung mit CodeCover

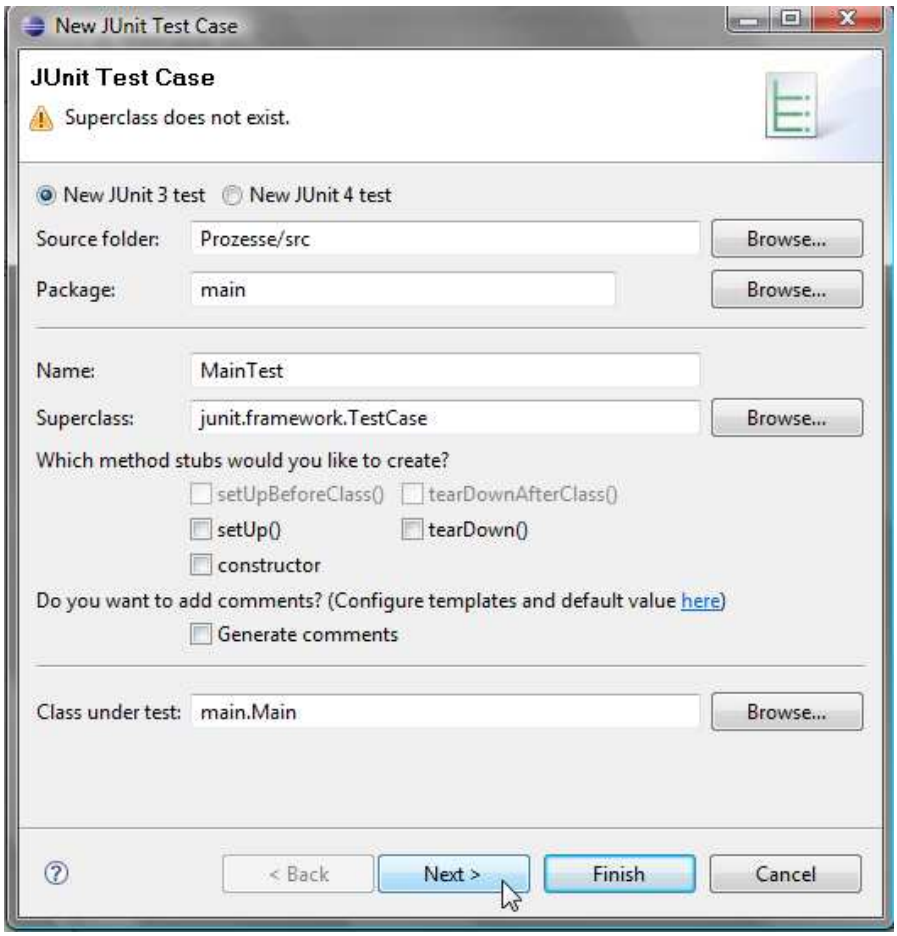
Mit JUnit wird Software getestet, mit CodeCover kann die Testüberdeckung bei einer Programmausführung gemessen werden. Dazu wird z. B. das folgende Programm genommen.

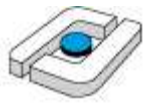


Um Programme testen zu können, werden Testfälle mit JUnit geschrieben. Eine einfache Möglichkeit zu einer Klasse Testfälle zu schreiben, ergibt sich z. B. durch einen Rechtsklick auf eine Klasse. Unter „New“ wird der Punkt „JUnit Test Case“ gewählt.

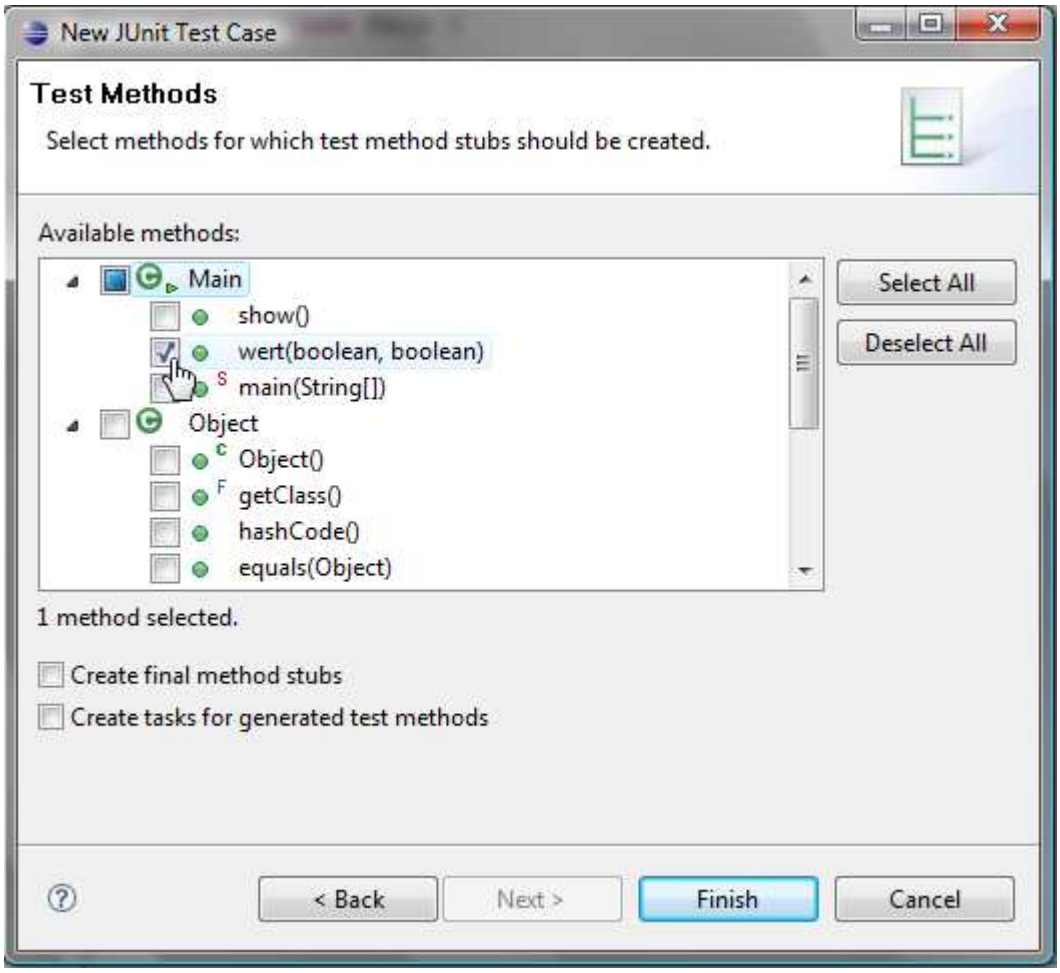


Im folgenden Fenster kann man sich für JUnit 3.8 oder 4.x entscheiden, weiterhin kann man die Initialisierungsmethoden für JUnit auswählen. Generell kann man die Auswahl mit „Finish“ abschließen, Hier werden mit „Next>“ weitere Einstellungsmöglichkeiten betrachtet.

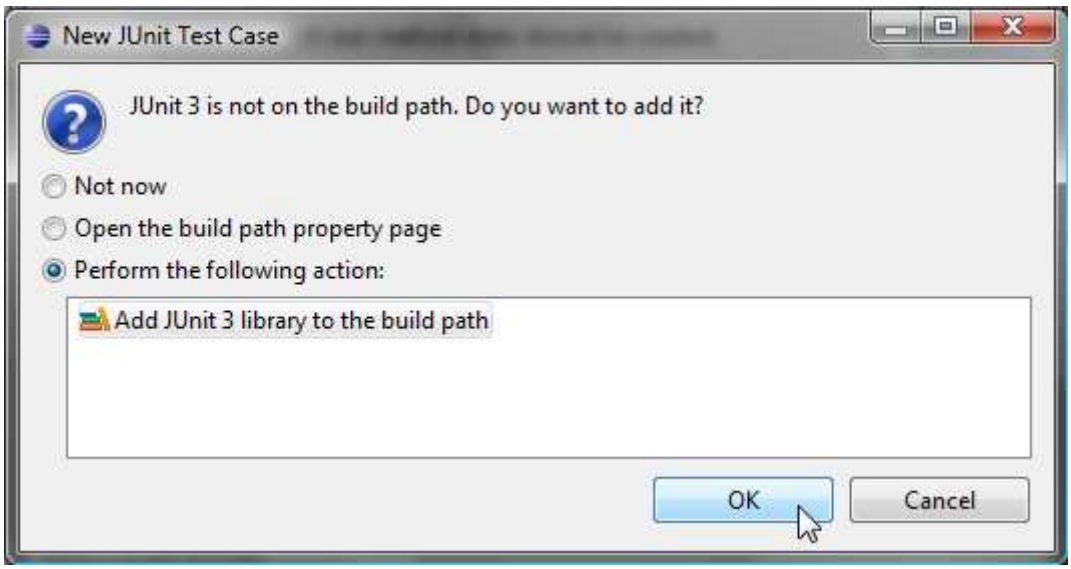


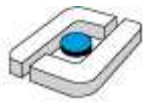


Man kann jetzt auswählen, für welche Methoden sogenannte Stubs erstellt werden sollen. Dies ist in diesem Fall ein leerer Testfall, der daran erinnern soll, dass hier noch ein Test fehlt.

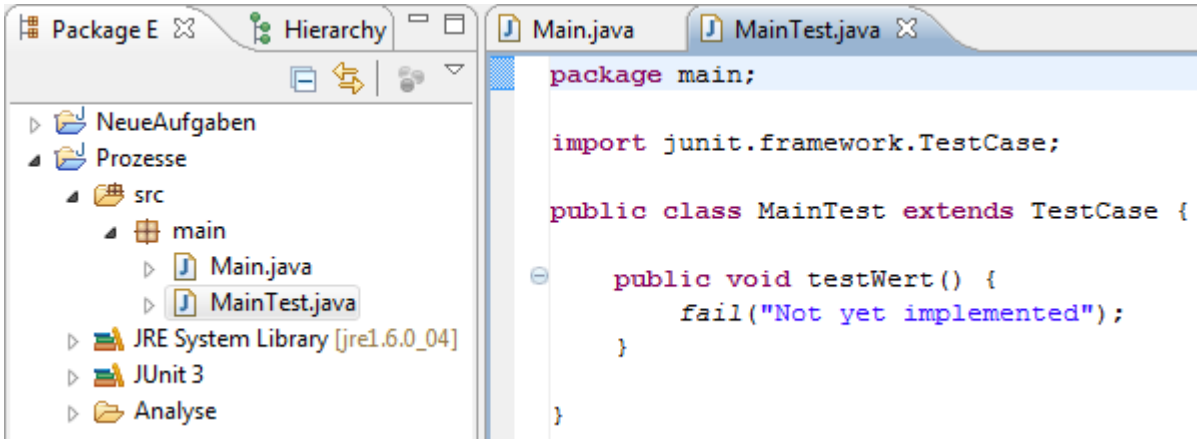


Wurde JUnit im Projekt noch nicht genutzt, muss das zugehörige jar-File in das Projekt eingebunden werden. Dies passiert z. B:, wenn man folgende Frage einfach mit „OK“ bestätigt.

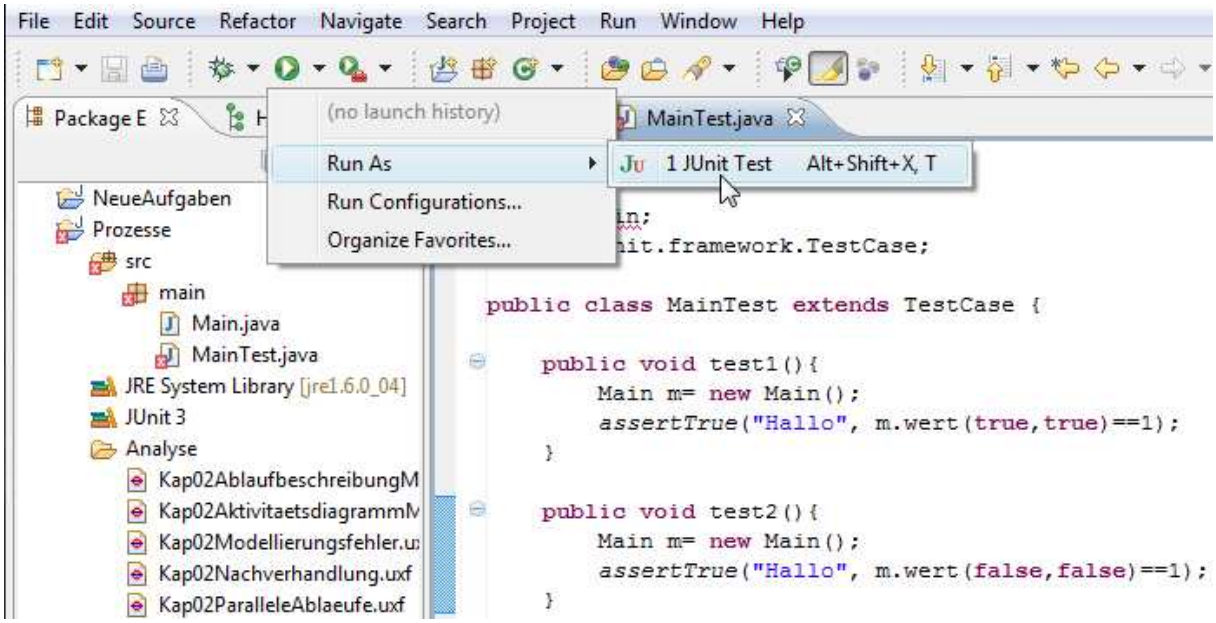




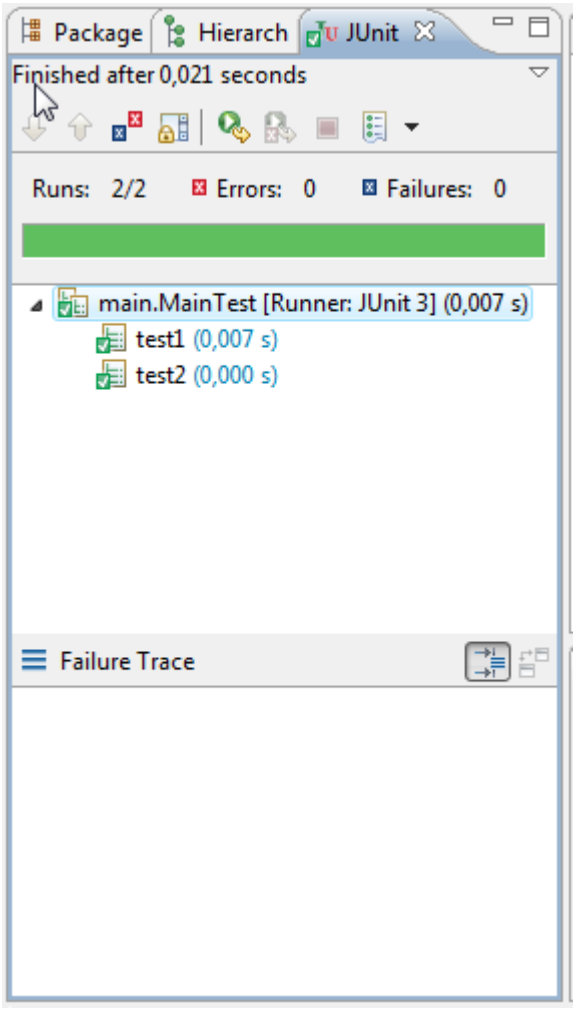
Man erhält dann z. B. folgende Testklasse.



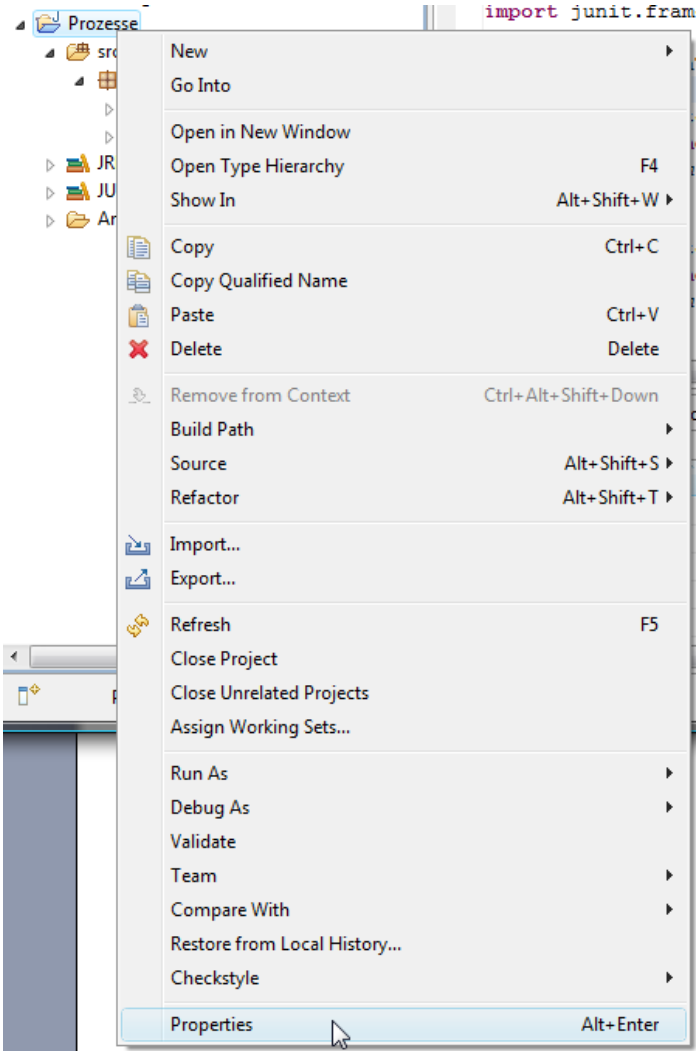
Man kann dann die Testklasse bearbeiten, weitere Testmethoden ergänzen und die Tests starten. Dazu wird im Startmenü beim Pfeil nach unten neben  dann „Run As“ und „JUnit Test“ gewählt.



Nach Ablauf der Tests wird das Ergebnis in einem eigenen Reiter auf der linken Seite angezeigt. Hier können dann Tests auch wiederholt werden. Durch Anklicken der Tests erhält man weitere Informationen. Fehlerinformationen werden unter „Failure Trace“ ausgegeben.

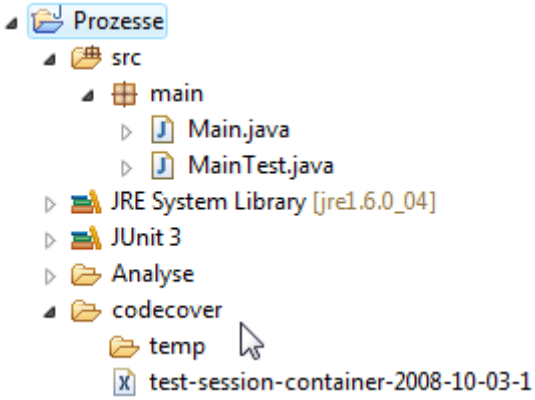


Es stellt sich nun die Frage, was alles im Test berücksichtigt wurde. Dies kann mit CoderCover analysiert werden. Es wird protokolliert, was alles ausgeführt wurde. Um CodeCover zu nutzen, muss dies zunächst für das Projekt eingeschaltet werden. Dies erfolgt z. B. durch ein Rechtsklick auf das Projekt und die Auswahl des „Properties“-Eintrags.

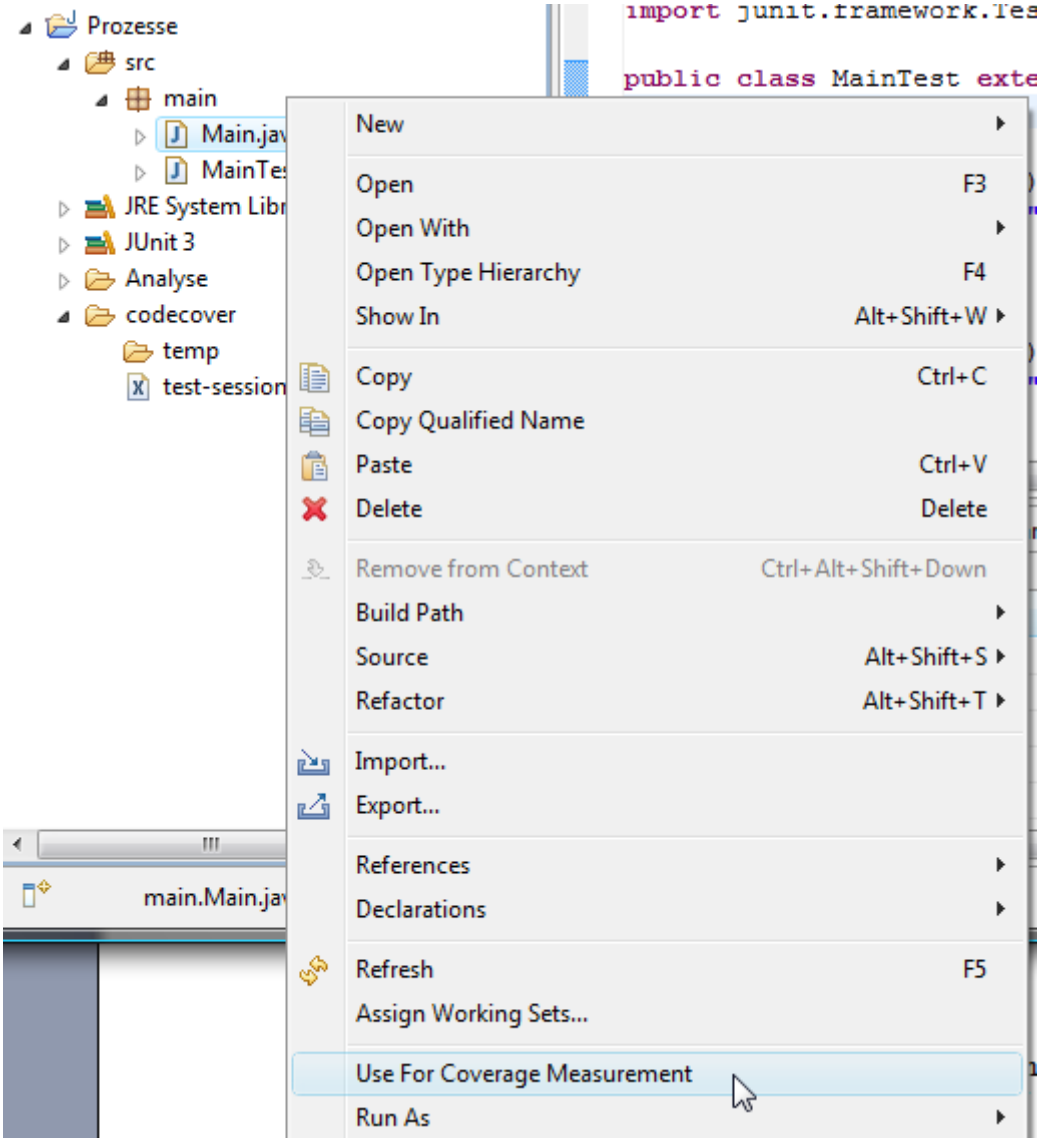


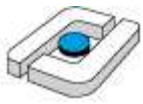
Hier wird dann beim Eintrag zu CodeCover ein Haken bei „Enable CodeCover“ gesetzt. Weiterhin können die Arten der Überdeckung, hier alle, ausgewählt werden.

Im Projektordner wird ein codecover-Ordner angelegt, in dem Informationen zu ausgeführten tests verwaltet werden.



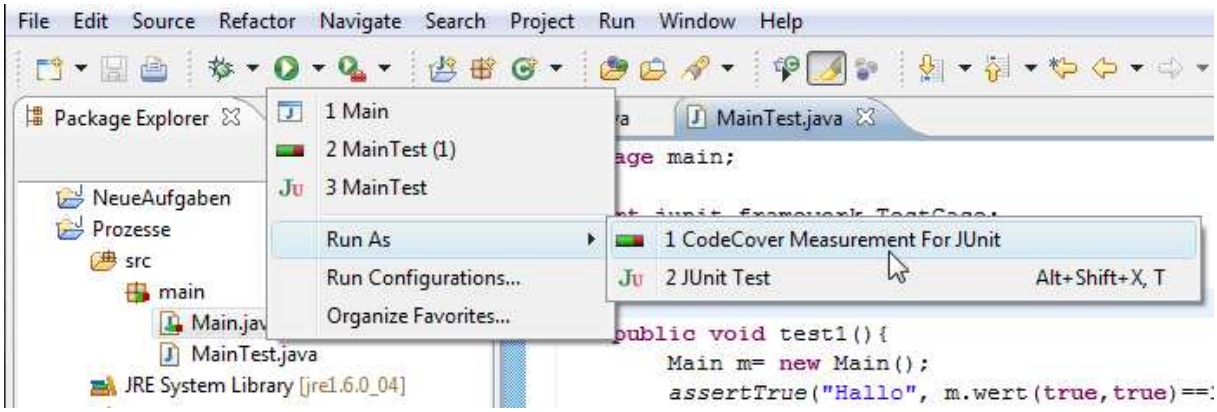
CodeCover muss für Klassen bzw. Pakete, die analysiert werden sollen, eingeschaltet werden. Dies ist etwas aufwändig, aber recht flexibel, da die Analyse Rechenzeit und Rechenleistung benötigt. Zum Einschalten macht man z. B. einen Rechtsklick auf die jeweilige Klasse oder das Paket und wählt „Use For Coverage Measurement“.



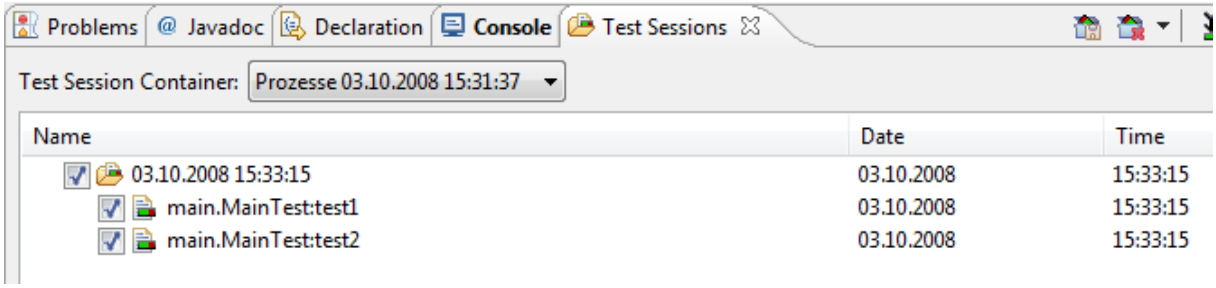


Man erkennt die ausgewählten Klassen und Pakete an einer kleinen grün-roten Markierung.

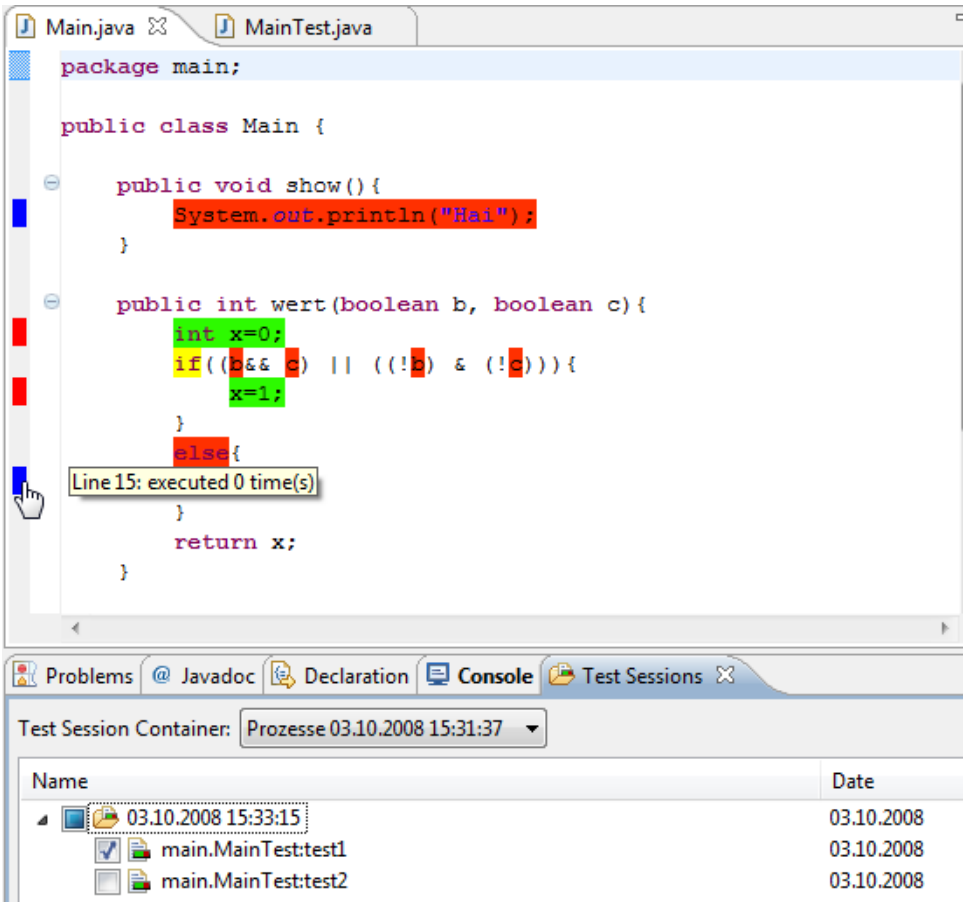
Will man jetzt die Tests starten, erhält man zusätzlich unter „Run As“ die Möglichkeit, CodeCover zu nutzen.



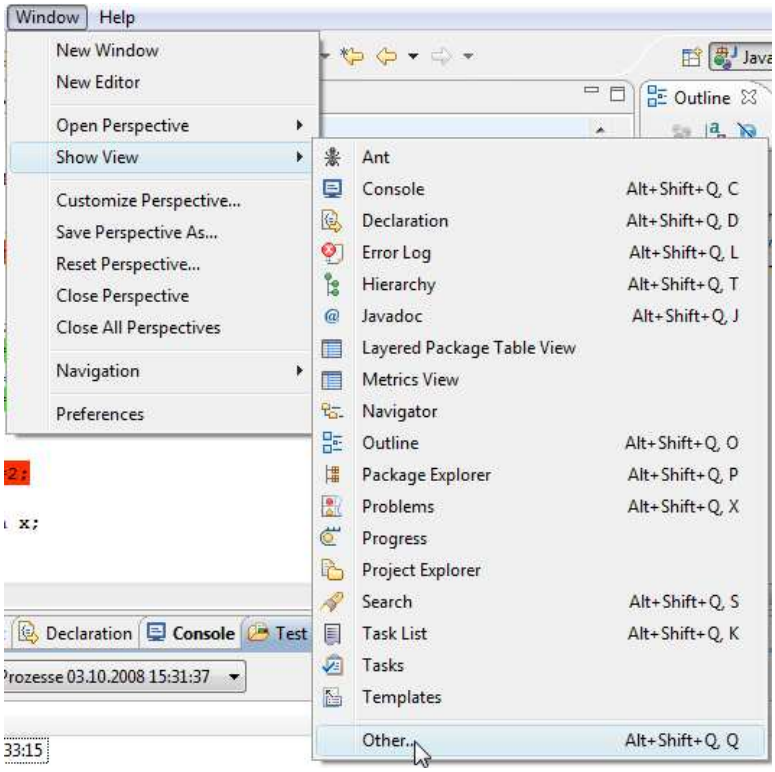
Nach der Programmausführung wird unten ein Reiter „Test Sessions“ mit den Tests angezeigt. Dieser Reiter wird bei der ersten Nutzung angelegt.

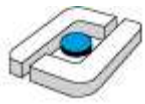


Hier kann man einzelne Tests oder alle Tests zusammen auswählen. Der Programmcode der getesteten Klassen wird dann markiert.

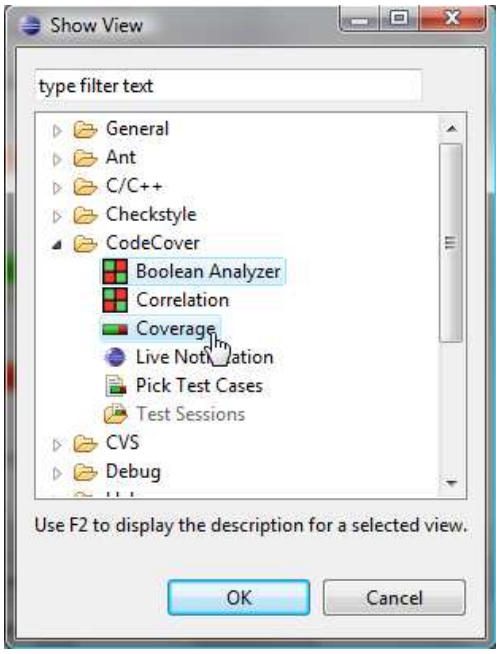


Um weitere Informationen zu erhalten, müssen die weiteren Views von CodeCover eingeblendet werden. Die geschieht unter „Window“, „Show View“ und „Other“.

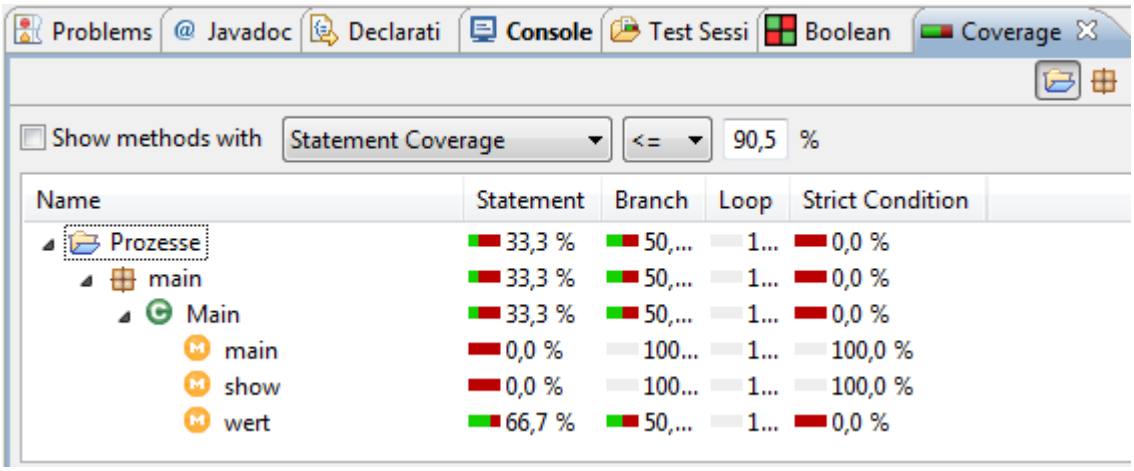




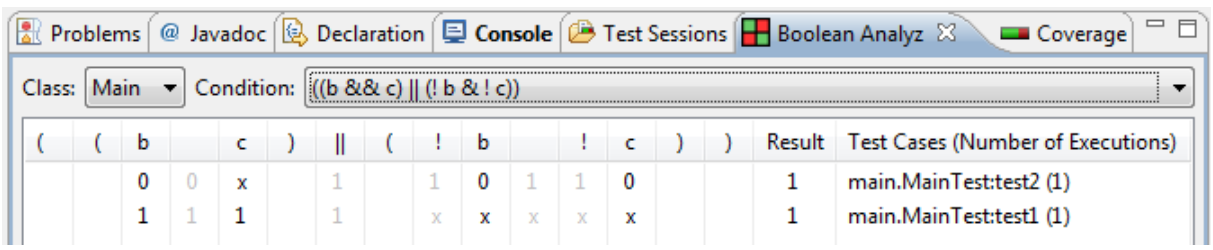
Interessant sind z. B. die Views „Boolean Analyzer“ und „Coverage“, die man zusammen bei gedrückter Strg-Taste (Control- Taste) auswählen kann.

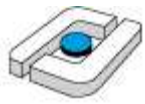


Unter Coverage erhält man Prozentangaben, dabei betrachtet „Statement“, ob alle Anweisungen ausgeführt wurden (in der betrachteten Version ist „return“ leider keine Anweisung), „Branch“, ob alle Booleschen Bedingungen einmal nach false und true ausgewertet wurden, „Loop“, ob alle Schleifen durchlaufen und beendet wurden. Eine graue „100%“ bedeutet, dass hier nichts zu untersuchen war.



Der Boolean Analyzer erlaubt eine sehr genaue Betrachtung, welche Werte einzelne Boolesche Ausdrücke angenommen haben. Wichtig ist, dass das Ergebnis immer von den unter „Test Session“ eingestellten Tests abhängt.





14 EclEmma

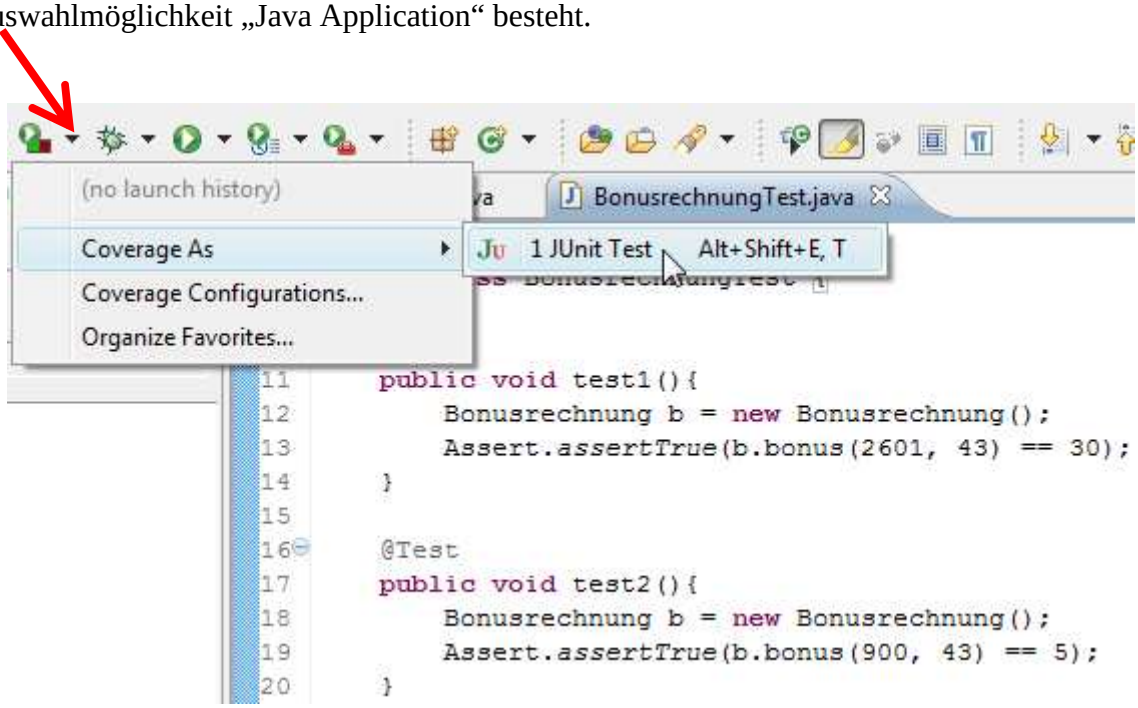
EclEmma ist ein Überdeckungswerkzeug für C0/C2-Tests, also die Anweisungsüberdeckung und die einfache Bedingungsüberdeckung. Es können entweder JUnit-Tests gestartet werden, so dass die Testüberdeckung gemessen wird oder einfach EClEmma zum Programmstart genutzt werden, wobei man nach dem Programmende erkennt, welche Zeilen ausgeführt wurden.

14.1 Integration von Eclemma in Eclipse

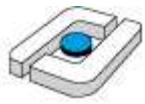
Das Plugin für Eclemma kann wie andere Plugins auch, direkt über das in Kapitel 8 beschriebene Erweiterungsverfahren in Eclipse eingebaut werden. Der zentrale Eintrag für die Erweiterung ist http://update.eclemma.org. Eine detaillierte gelungene Einführung findet man unter http://www.eclemma.org/.

14.2 Analyse der Überdeckung mit EclEmma

Zum Start gibt es verschiedene Möglichkeiten. Die einfachste Variante ist der zusätzlich in der oberen Werkzeugleiste erscheinende Knopf zum Start von EclEmma, wobei genauer der kleine Pfeil rechts daneben für das Drop-Down-Menü genutzt wird. Der Start erfolgt dann über „Coverage As“, wobei man bei JUnit-Testklassen mit eigener main-Methode hier zwei Auswahlmöglichkeiten hat und für einfache Klassen und einfache JUnit-Tests nur die Auswahlmöglichkeit „Java Application“ besteht.



Nach dem Programmende werden durchlaufene Zeilen grün, teilweise durchlaufene Zeilen gelb, nicht durchlaufene Zeilen rot angezeigt. Ohne weitere Einstellungen werden auch die Tests markiert, da sie als „normale“ Programme behandelt werden.



Die direkte Einfärbung laufender Programme hat den Vorteil, dass man mit EclEmma beliebige Programme ausführen und ihre Überdeckung messen kann. Ein gutes Beispiel sind Programme mit Oberfläche, bei der man verschiedene Szenarien durchklickt und am Ende der Programmausführung Informationen über die Überdeckung erhält. Insofern er nicht schon gestartet ist, wird weiterhin unten der Coverage-View mitgestartet.

Bonusrechnung.java | BonusrechnungTest.java

```
8
9  public class BonusrechnungTest {
10
11     @Test
12     public void test1(){
13         Bonusrechnung b = new Bonusrechnung();
14         Assert.assertTrue(b.bonus(2601, 43) == 30);
15     }
16
17     @Test
18     public void test2(){
19         Bonusrechnung b = new Bonusrechnung();
20         Assert.assertTrue(b.bonus(900, 43) == 5);
21     }
22
23     @Ignore
24     @Test
25     public void test3(){
26         Bonusrechnung b = new Bonusrechnung();
27         Assert.assertTrue(b.bonus(2600, 88) == 15);
28     }
29
30 }
```

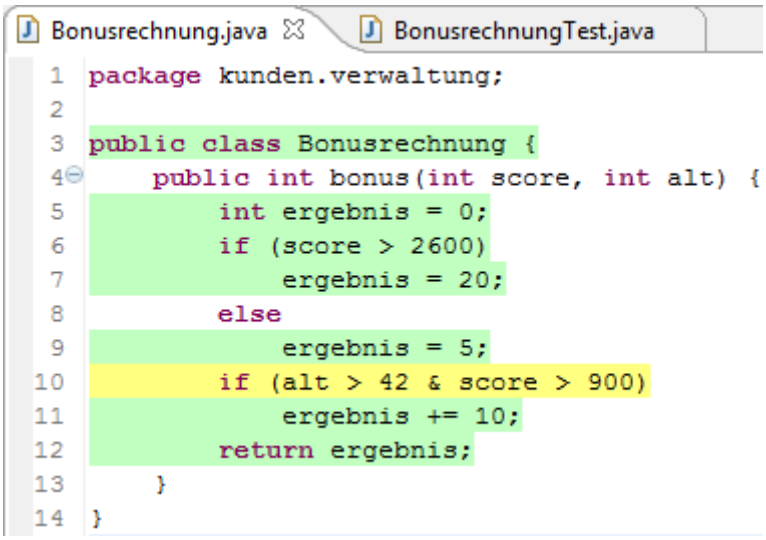
Console | Problems | Declaration | TestNG | Coverage

BonusrechnungTest (15.10.2010 18:28:23)

Element	Coverage	Covered In...	Missed In...	Total Instr...
Ueberdeckungen	76,9 %	60	18	78
src	76,9 %	60	18	78
kunden.verwaltung	76,9 %	60	18	78
BonusrechnungTest.java	64,6 %	31	17	48
Bonusrechnung.java	96,7 %	29	1	30
Bonusrechnung	96,7 %	29	1	30
bonus(int, int)	96,3 %	26	1	27

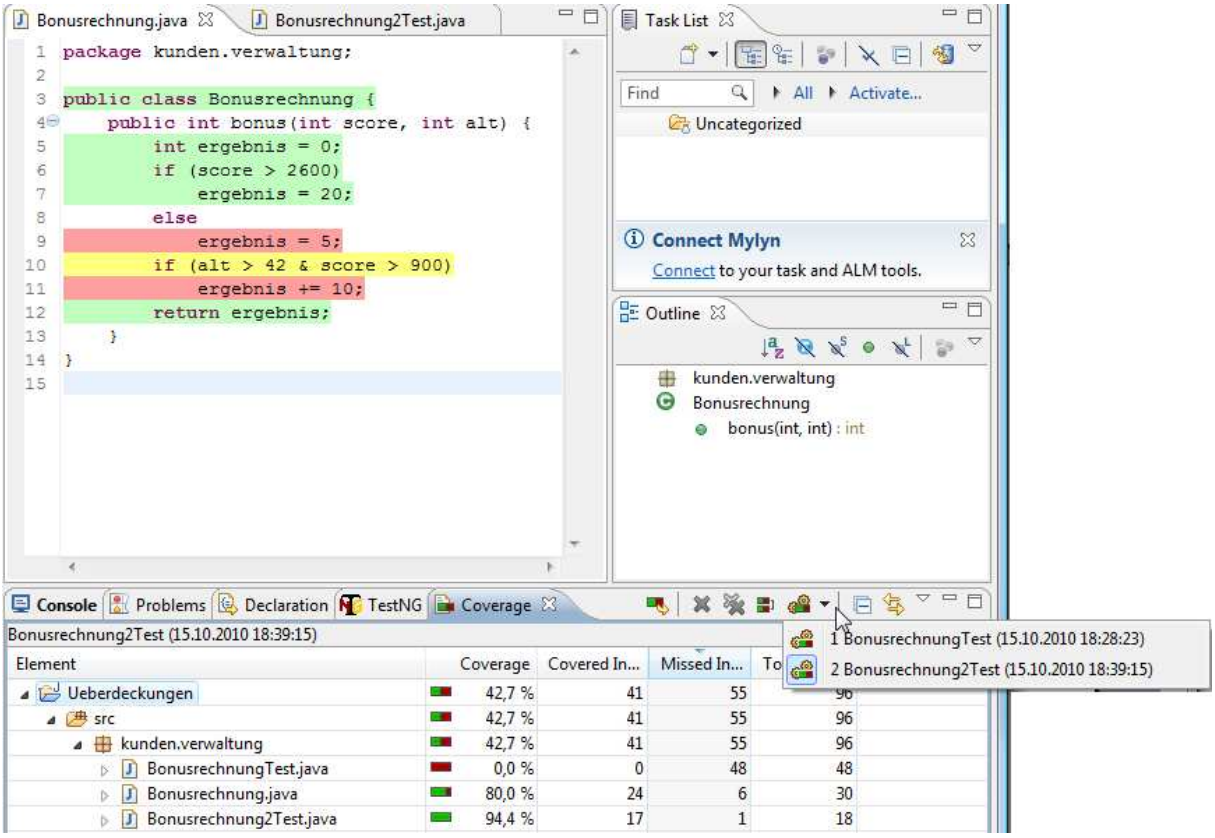
Kleine Ungenauigkeiten sind bei der Berechnung möglich, da Leerzeilen nicht immer sauber behandelt werden. Bei Enumerations ist zu beachten, dass der von Java generierte Code analysiert wird und es sich so bei jeder Enumeration um eine Klasse mit mindestens 50 Zeilen handelt. Weiterhin basiert die Zählung der Anweisungen (Instructions) auf dem Byte-Code.

Gelb markierte Zeilen bedeuten, dass es in dieser Zeile einen Booleschen Ausdruck gibt, der nicht nach true und nach false ausgewertet wurde.

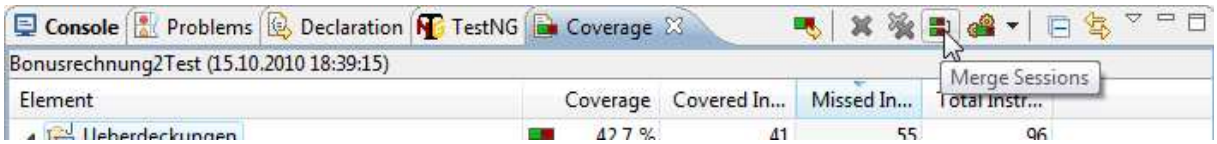


14.3 Vereinigung von Testdurchläufen

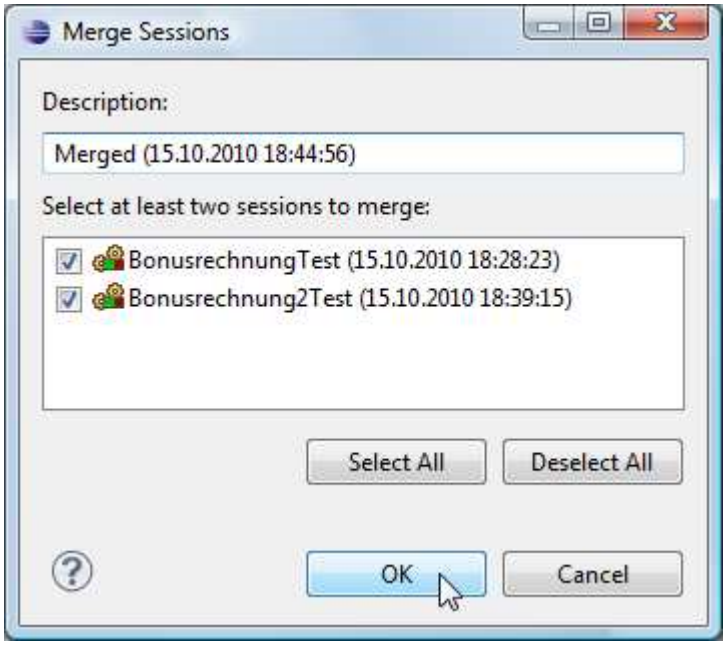
Lässt man EclEmma mehrfach laufen, kann man die Ergebnisse der Testdurchläufe vereinigen. Das folgende Bild zeigt das Überdeckungsergebnis für eine zweite Testklasse, man sieht rechts unten die Möglichkeit, über den kleinen Pfeil nach unten, zwischen den Ergebnissen der Testdurchführungen umzuschalten.



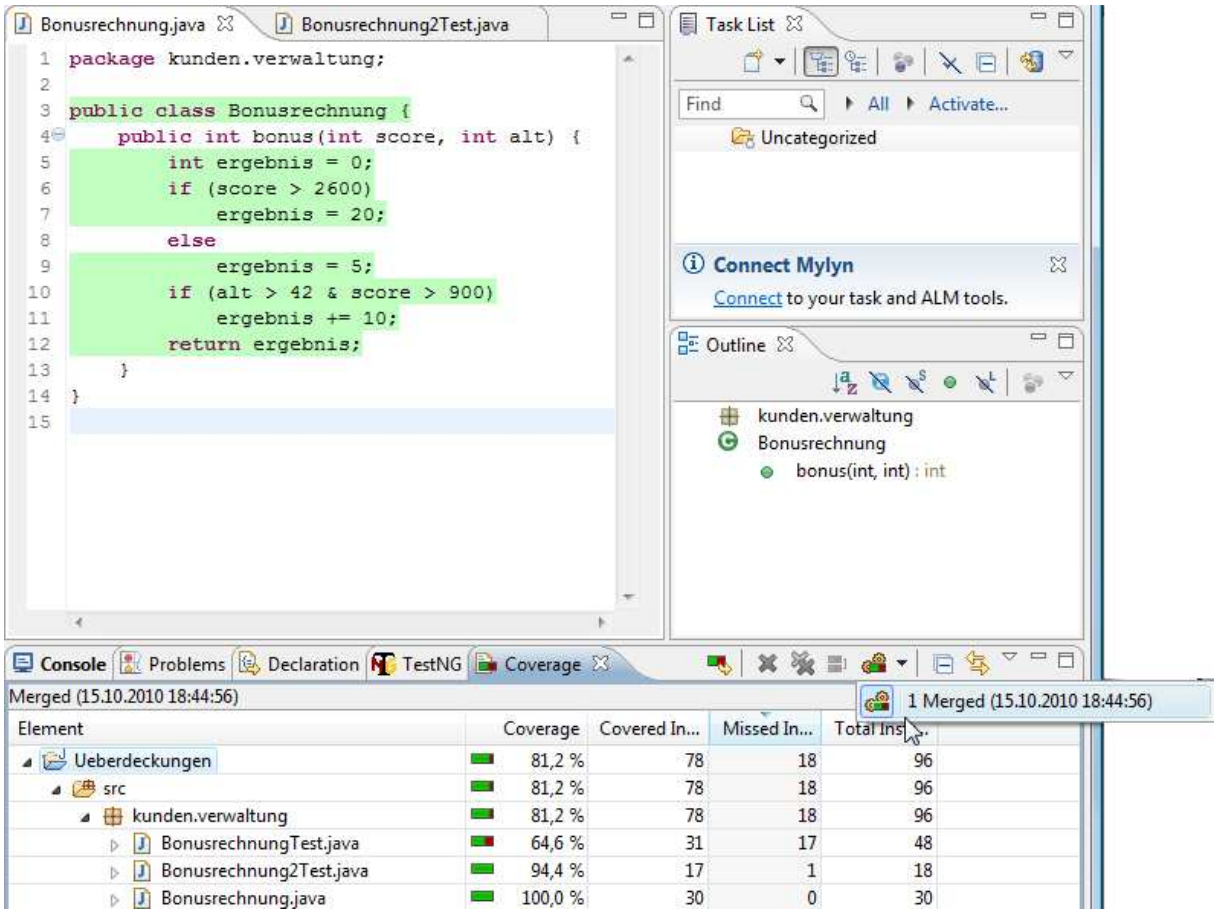
Der Merge-Knopf ermöglicht es, die Ergebnisse mehrerer Testdurchführungen zu vereinigen.



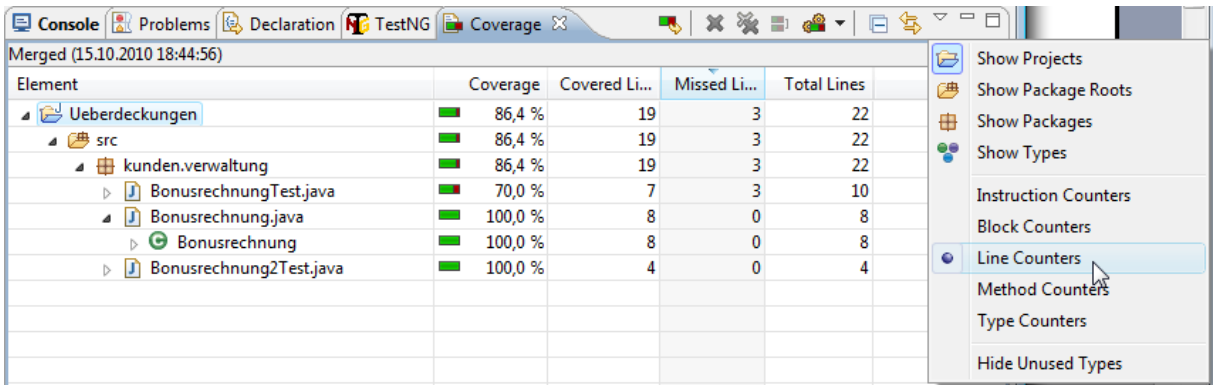
Man kann dann auswählen, welche Sessions vereinigt werden sollen.



Das folgende Bild zeigt das Ergebnis der Vereinigung. Positiv ist zu bemerken, dass die bei beiden Testdurchführungen gelb markierte Zeile jetzt grün wird, da alle Booleschen Teilausdrücke nach true und false ausgewertet wurden. Kritisch bleibt anzumerken, dass man die Vereinigung nicht mehr rückgängig machen kann.

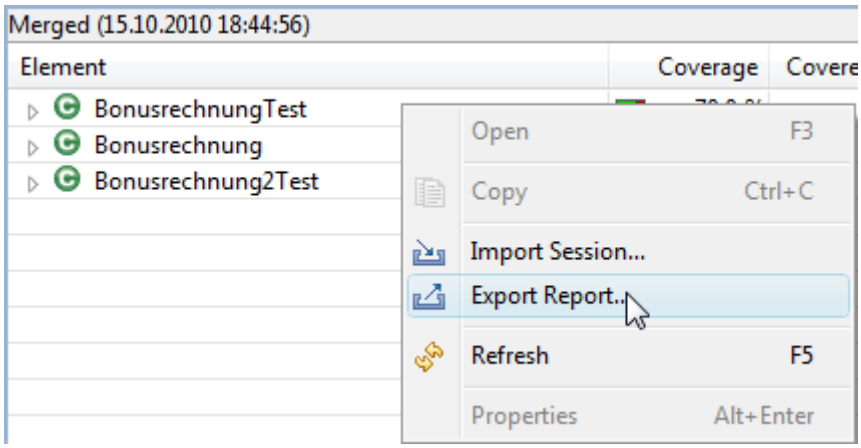


Mit dem weißen Pfeil nach unten, dritter von rechts, kann man verschiedene Zählweisen für EclEmma einstellen. Eventuell etwas intuitiver ist die Zählung von Programmzeilen.

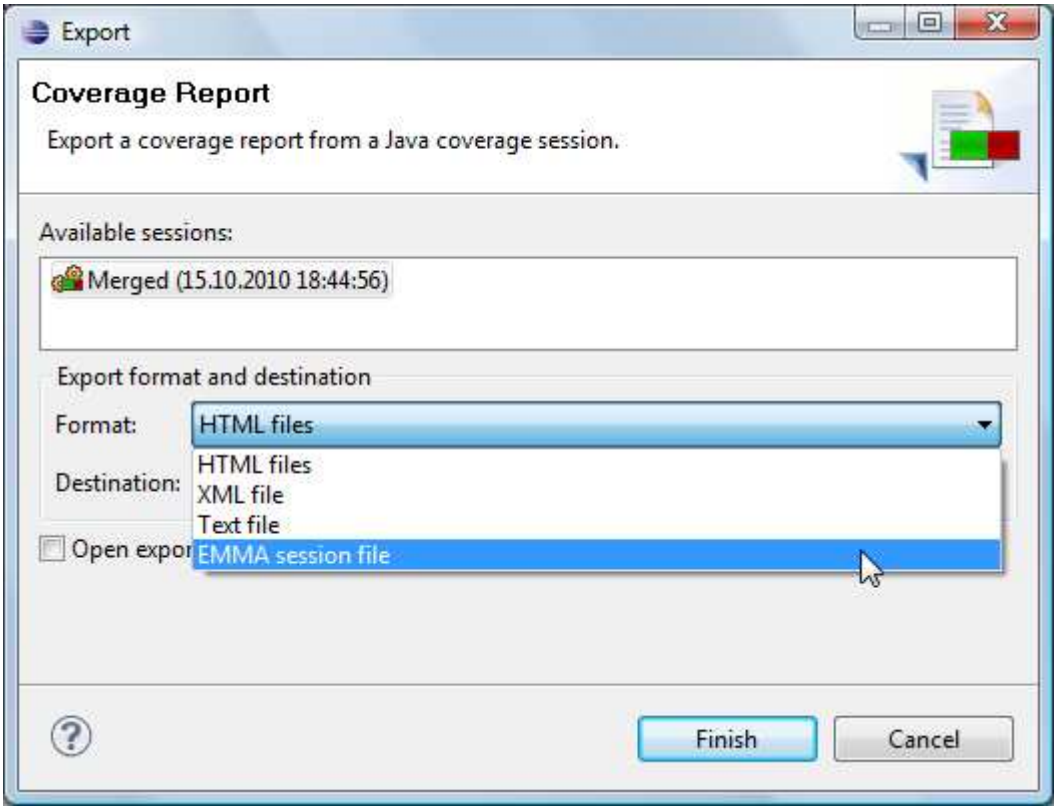


14.4 Verwaltung von Testdurchläufen

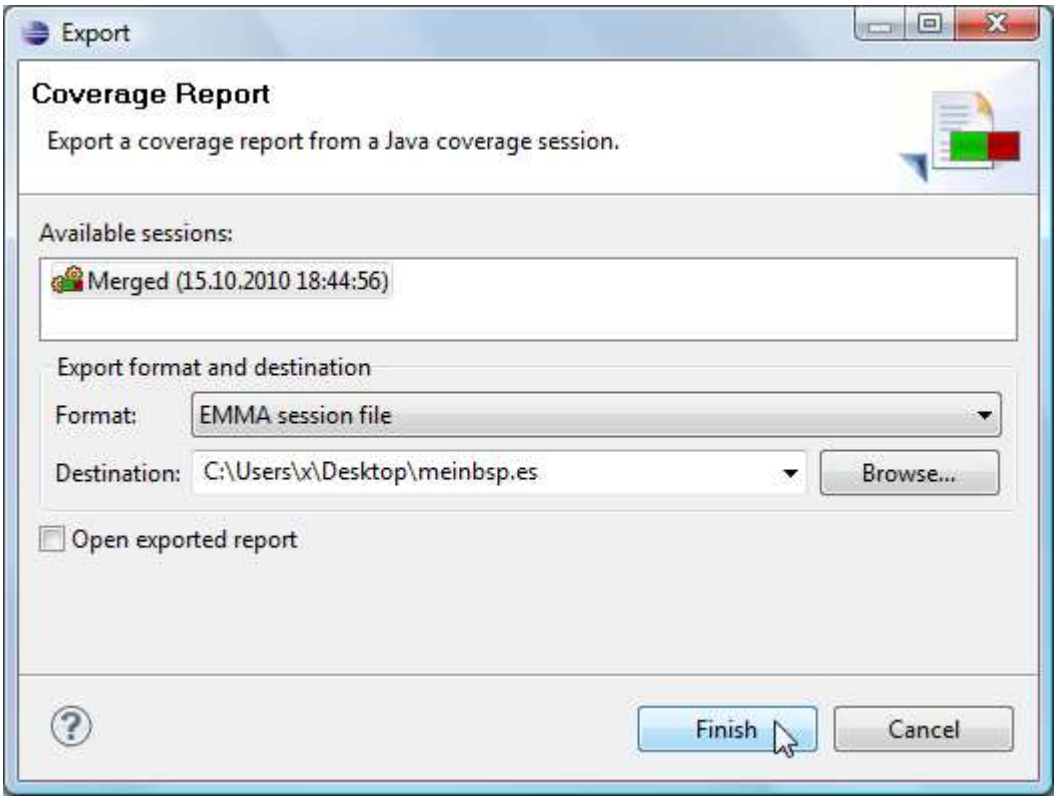
Zunächst erkennt man nicht, dass man die Daten der Testdurchläufe auch weiter verwalten und aufbereiten kann. Diese Möglichkeiten bestehen aber, wenn man einen Rechtsklick im Coverage-Fenster macht.



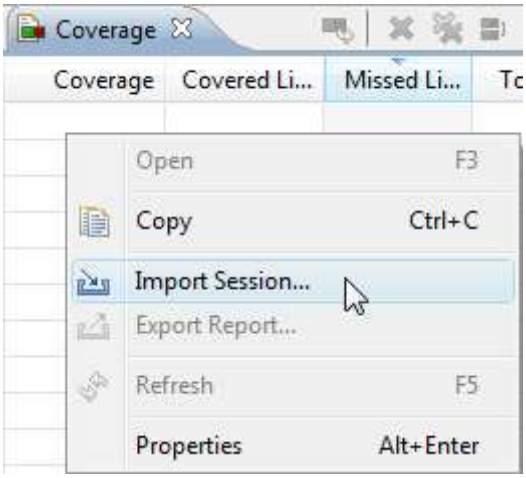
Nach der Wahl von „Export Report“ bestehen folgende Auswahlmöglichkeiten. Mit HTML gibt es eine einfache Webaufbereitung, die auch den farbig gekennzeichneten Code zeigt. Mit XML besteht die Möglichkeit zur Weiterverarbeitung. Interessant ist die Auswahl als „EMMA session file“, die hier weiter verfolgt wird.



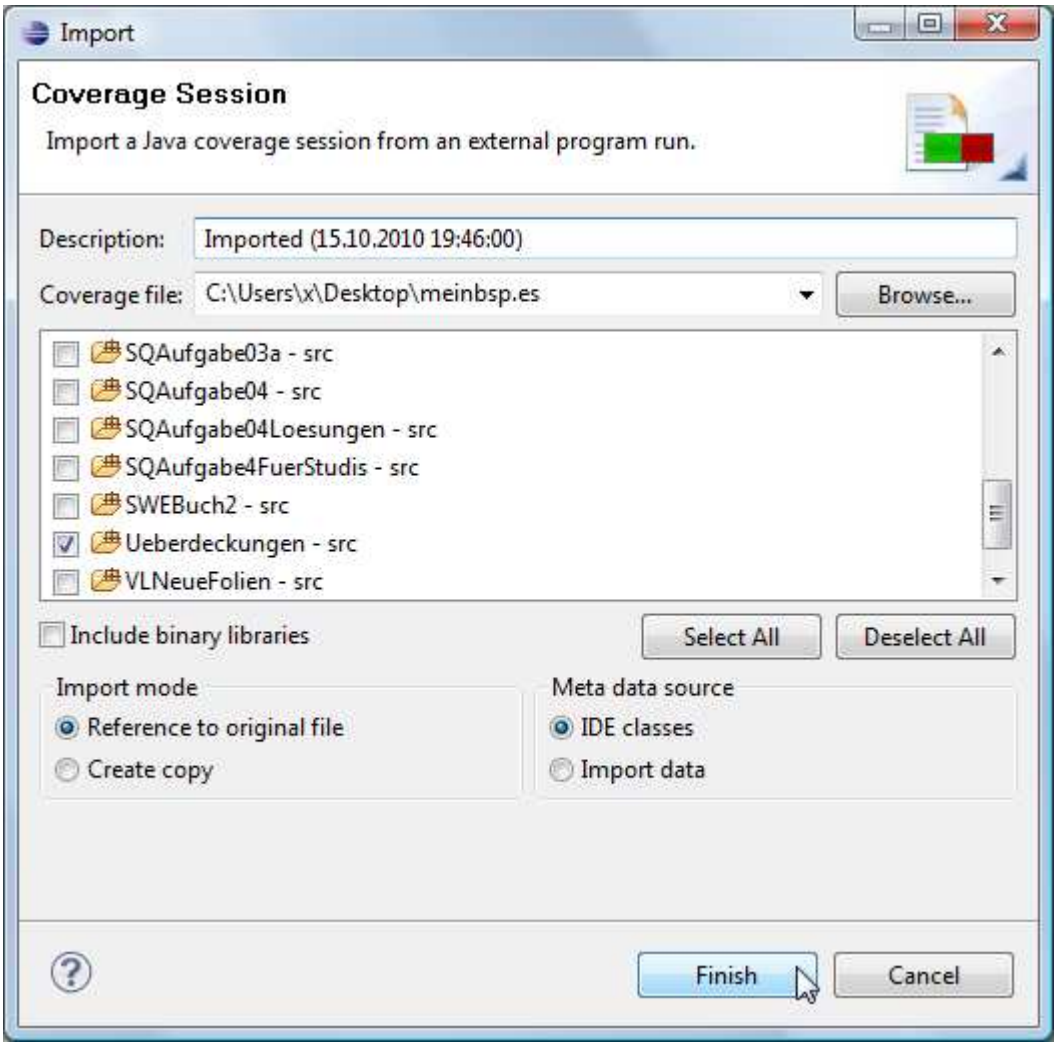
Der Speicherort kann frei und sinnvoller als in der Abbildung gewählt werden, die Endung „.es“ sollte beibehalten werden.

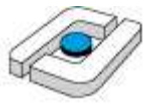


Beendet man Eclipse und öffnet den Workspace wieder, sind zunächst alle Testmarkierungen verloren. Hat man diese als „EMMA session file“ gespeichert, kann man diese wieder wie folgt importieren.



Hier muss die gespeicherte Datei unter „Browse...“ als Eintrag für „Coverage File:“ gesucht werden. Den Namen des Eintrags kann man in der „Description:“ ändern, muss es aber nicht. Wichtig ist, dass man im mittleren Fenster das Projekt markiert, zu dem die aufgezeichneten Daten gehören. Die Einstellungen werden mit „Finish“ abgeschlossen.


Import
Coverage Session
Import a Java coverage session from an external program run.
Description:
Imported (15.10.2010 19:46:00)
Coverage file:
C:\Users\x\Desktop\meinbsp.es
Browse...
SQAufgabe03a - src
SQAufgabe04 - src
SQAufgabe04Loesungen - src
SQAufgabe4FuerStudis - src
SWEBuch2 - src
Ueberdeckungen - src
VLNeueFolien - src
Include binary libraries
Select All
Deselect All
Import mode
Reference to original file
Create copy
Meta data source
IDE classes
Import data
Finish
Cancel



15 Testen mit TestNG

TestNG (http://testng.org/) ist neben JUnit ein zweites weit verbreitetes Testframework, dass etwas mehr Funktionalität als JUnit anbietet. Da allerdings viele weitere Testwerkzeuge auf JUnit basieren, also z. B. JUnit-Testfälle generieren, kann man auch bei JUnit als Testsbasis bleiben.

15.1 Integration von TestNG in Eclipse

Das Plugin für TestNG kann wie andere Plugins auch, direkt über das in Kapitel 8 beschriebene Erweiterungsverfahren in Eclipse eingebaut werden. Der zentrale Eintrag für die Erweiterung ist http://beust.com/eclipse.

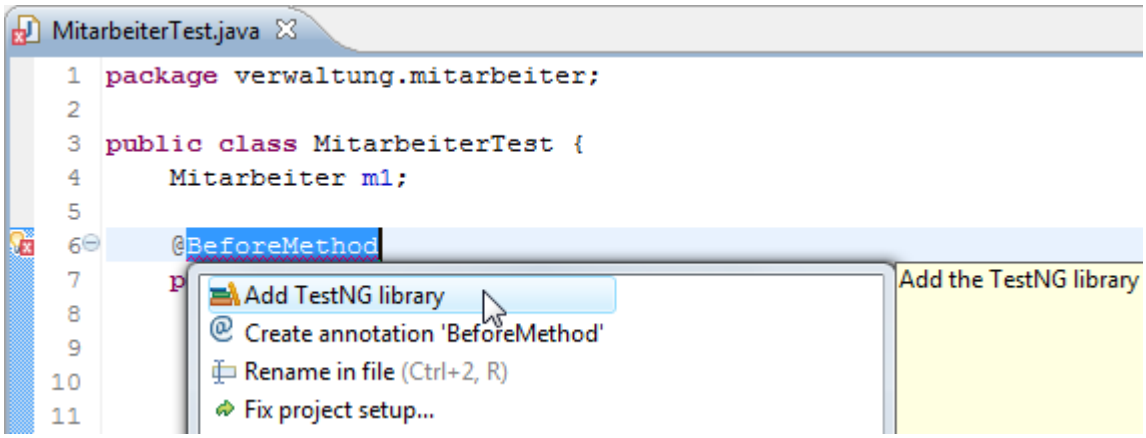
15.2 Erstellung eines ersten Testfalls mit TestNG

Wie immer gibt es verschiedene Wege die auch bei der Testerstellung genutzt werden können. Eine Möglichkeit besteht darin, einfach eine Java-Klasse zu schreiben, die die TestNG –Annotationen nutzt. Ein Ausschnitt kann wie folgt aussehen.

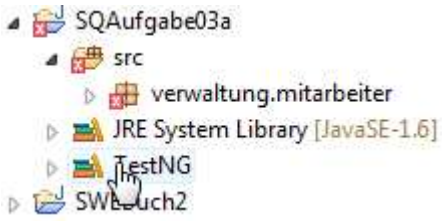
MitarbeiterTest.java

```
1  package verwaltung.mitarbeiter;
2
3  public class MitarbeiterTest {
4      Mitarbeiter m1;
5
6      @BeforeMethod
7      public void setUp() throws Exception {
8          m1= new Mitarbeiter("Uwe","Mey");
9          m1.addFachgebiet(Fachgebiet.ANALYSE);
10         m1.addFachgebiet(Fachgebiet.C);
11         m1.addFachgebiet(Fachgebiet.JAVA);
12     }
```

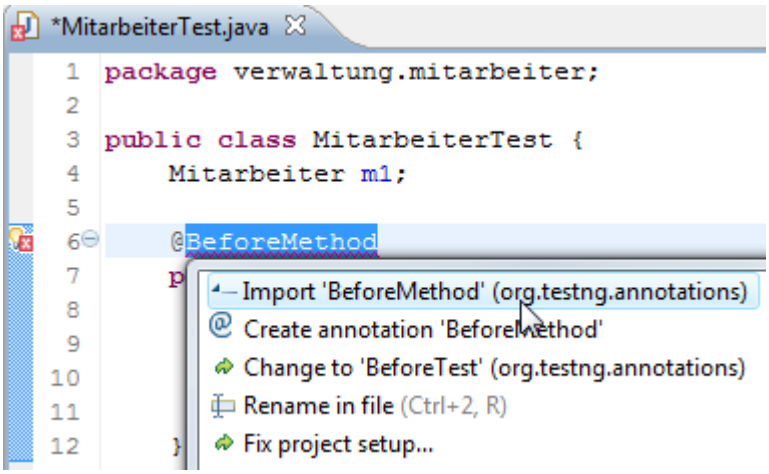
Macht man dann einen Linksklick auf die Fehlerkennzeichnung am linken Rand, so wird automatisch die Einbindung von TestNG vorgeschlagen, die dann ausgeführt wird.



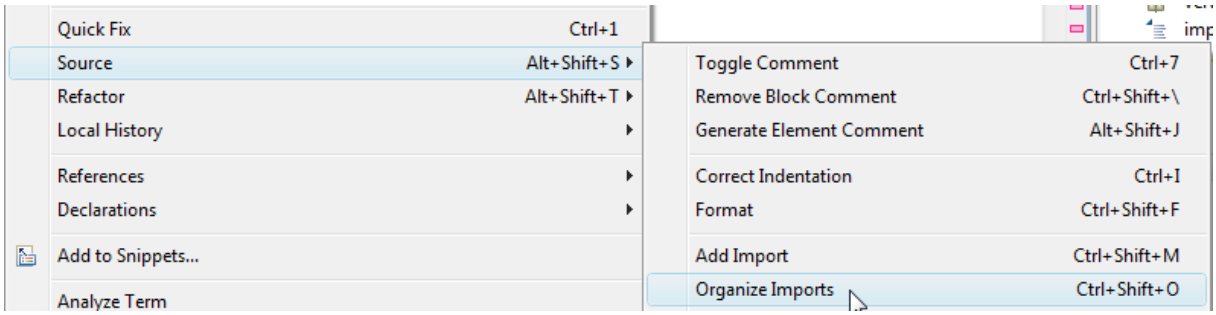
TestNG ist dann unter den benutzten Bibliotheken eingetragen.



Die Fehlermeldung ist nach wie vor sichtbar, da die zugehörige import-Anweisung fehlt. Diese kann wieder durch einen Links-Klick auf die Fehlermeldung nachgeholt werden.



Alternativ hilft bei mehreren verwendeten Annotationen auch ein Rechtsklick in einem freien Bereich des Editorfensters, hier wird dann „Source->Organize Imports“ gewählt.

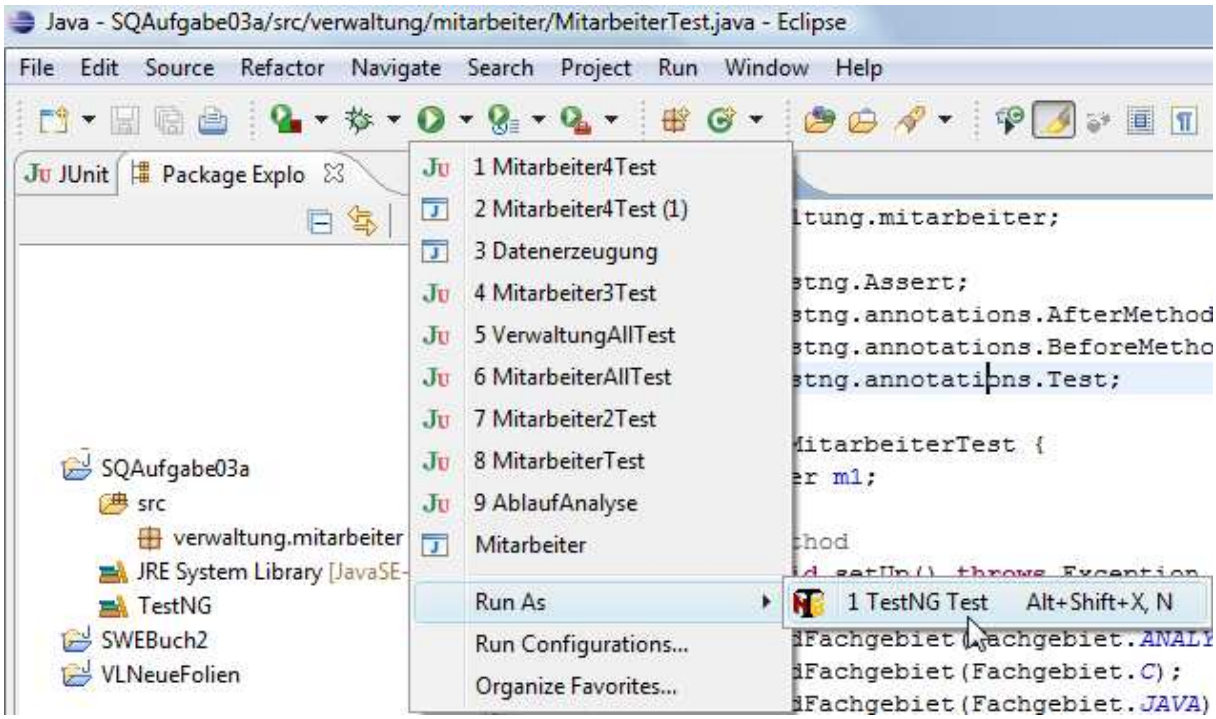


Die import-Anweisungen für eine Testklasse können dann wie folgt aussehen.

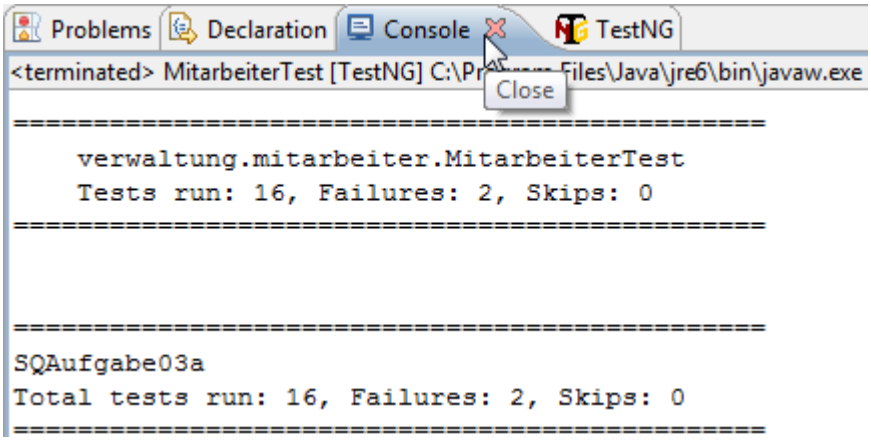
MitarbeiterTest.java

```
1  package verwaltung.mitarbeiter;
2
3  import org.testng.Assert;
4  import org.testng.annotations.AfterMethod;
5  import org.testng.annotations.BeforeMethod;
6  import org.testng.annotations.Test;
7
8  public class MitarbeiterTest {
9      Mitarbeiter m1;
10
11     @BeforeMethod
12     public void setUp() throws Exception {
13         m1= new Mitarbeiter("Uwe","Mey");
14         m1.addFachgebiet(Fachgebiet.ANALYSE);
15         m1.addFachgebiet(Fachgebiet.C);
16         m1.addFachgebiet(Fachgebiet.JAVA);
```

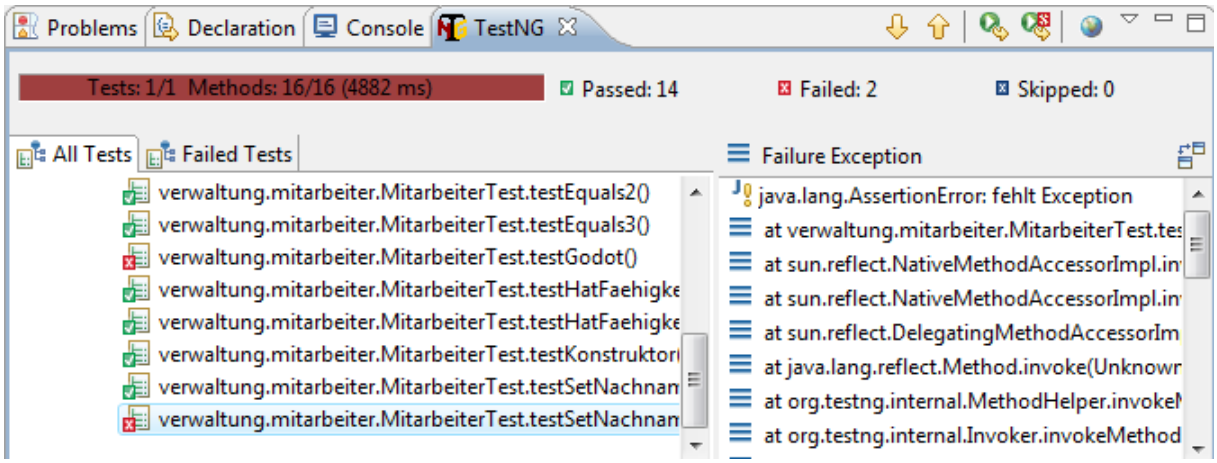
TestNG-Tests können wie normale Java-Programme und JUnit-Tests über viele Wege ausgeführt werden. Befindet sich die Klasse mit den Tests im aktuellen Editorfenster, kann der Aufruf über das Menü gestartet werden, dass man durch den kleinen Pfeil neben dem Run-Knopf erreicht. 



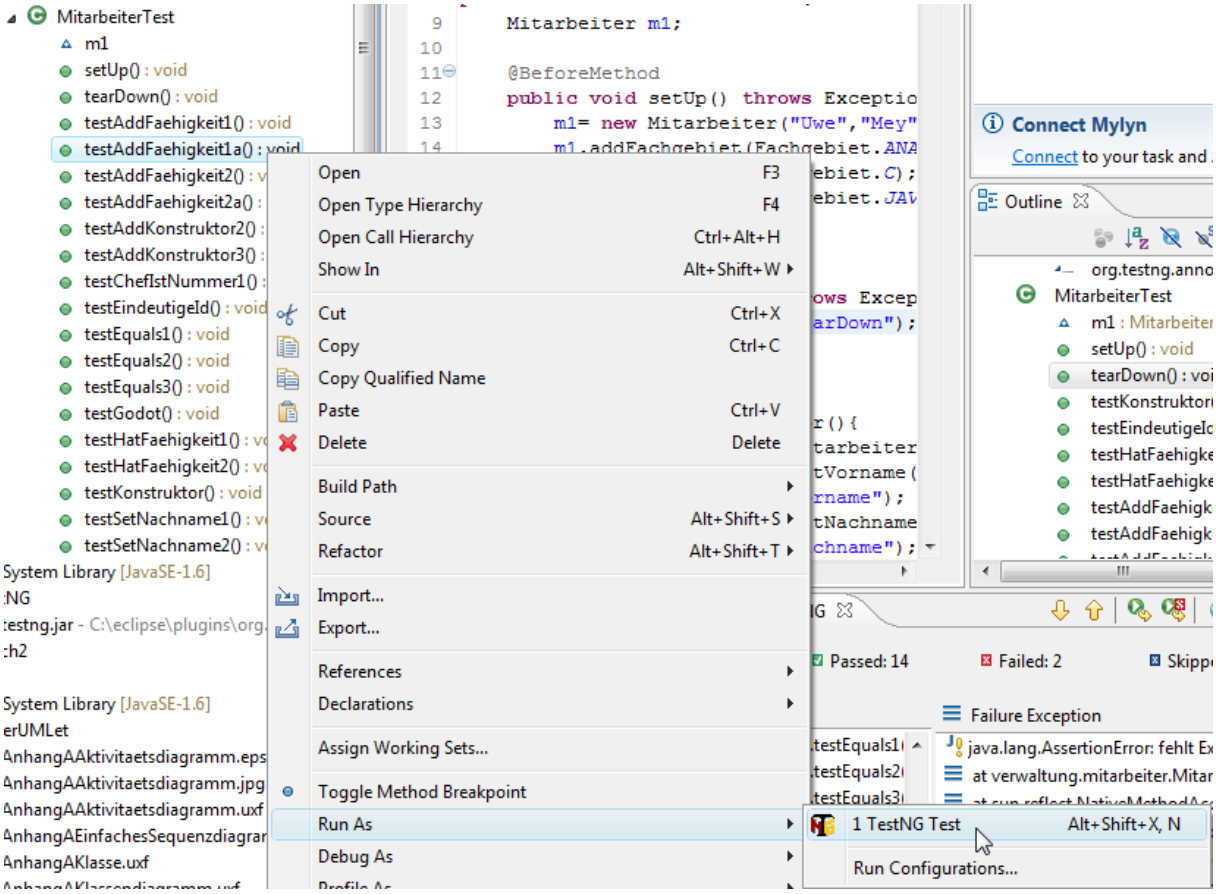
Die Ergebnisse stehen zum Einen in der Consolen-Ausgabe.



Zum Zweiten gibt es einen eigenen TestNG-View, der beim ersten Aufruf automatisch bei den unteren Reitern angezeigt wird und der eine ähnliche Informationsaufbereitung, wie die JUnit-Ausgabe bietet.

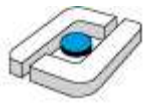


Man kann einzelne Tests auch direkt starten. Dazu wird ein Rechtsklick auf dem Testfall durchgeführt und „Run As -> TestNG Test“ gewählt.



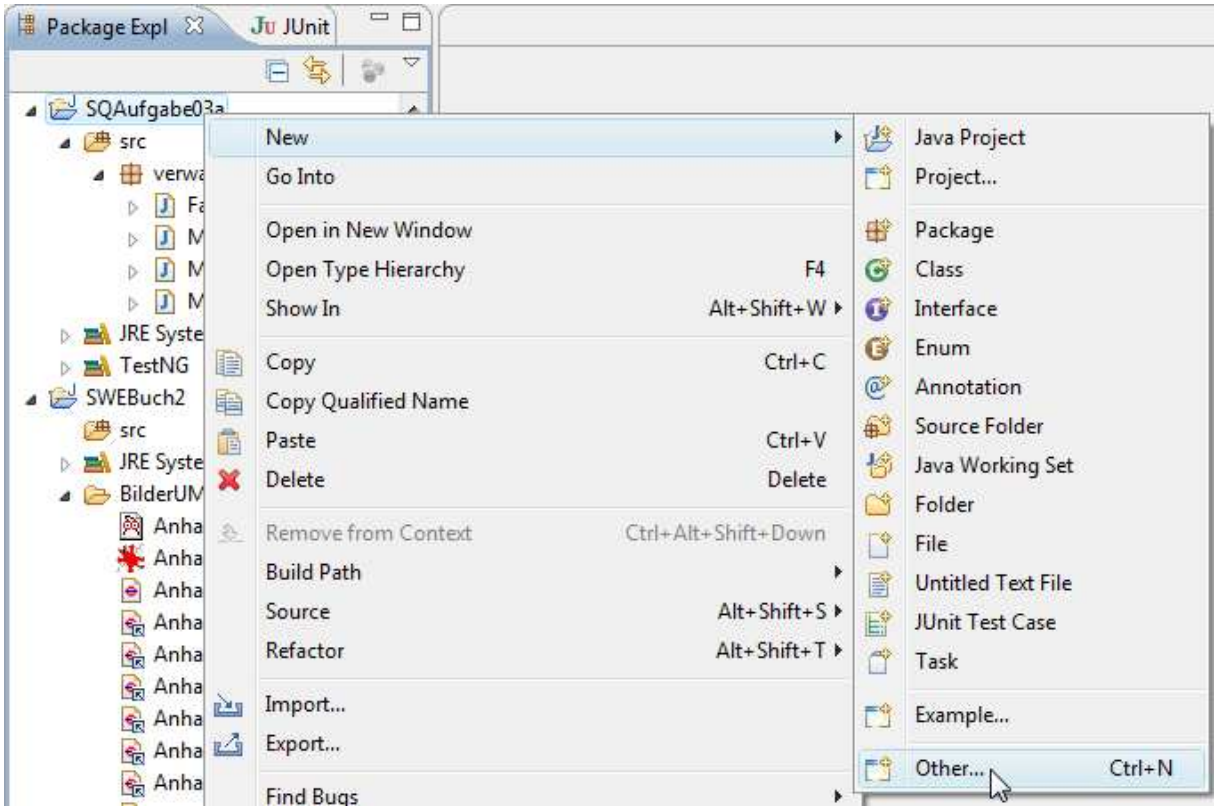
15.3 Erstellung von Start-Konfigurationen

TestNG kann generell sehr flexibel gestartet werden. Interessant sind dabei die Möglichkeiten die Testausführung üner die XML-Datei testng.xml zu steuern. Das folgende Beispiel zeigt

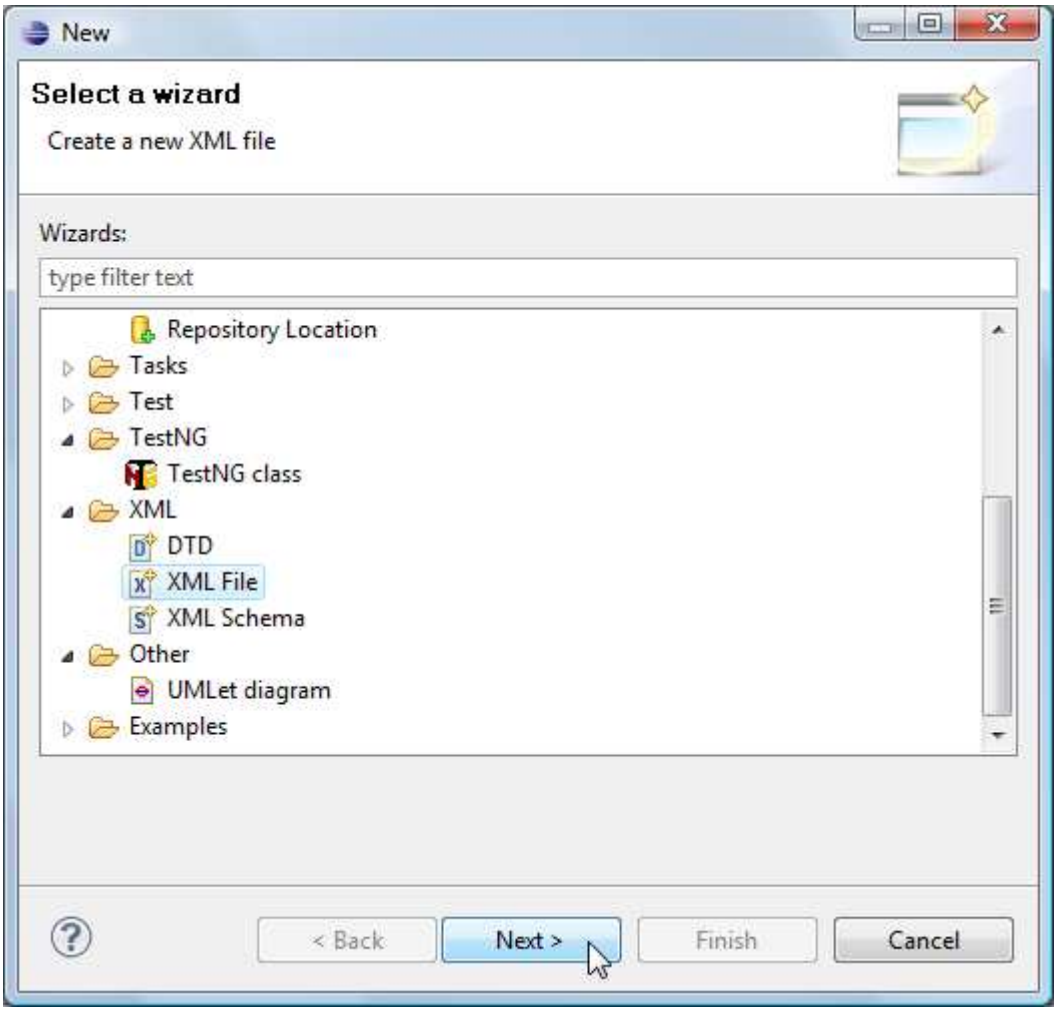


als Einstieg die Möglichkeit mehrere Testfälle, hier Testklassen zu einer TestSuite zusammen zu fassen.

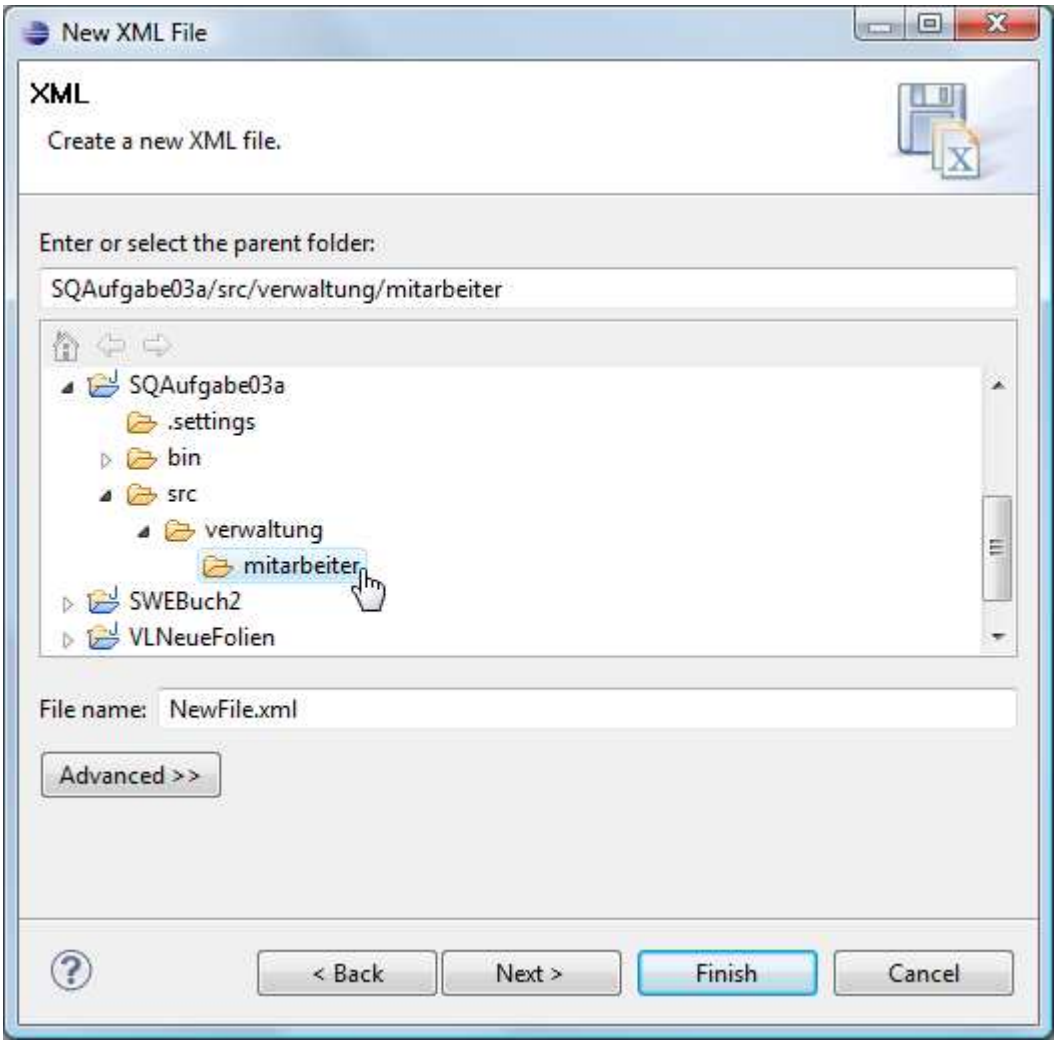
Abhängig von der Eclipse-Installation, kann man eine XML-Datei systematisch anlegen oder muss sie von Hand als Textdatei anlegen. Zur Erstellung einer XML-Datei kann man wie folgt vorgehen. Zunächst wird „New->Other...“ gewählt.



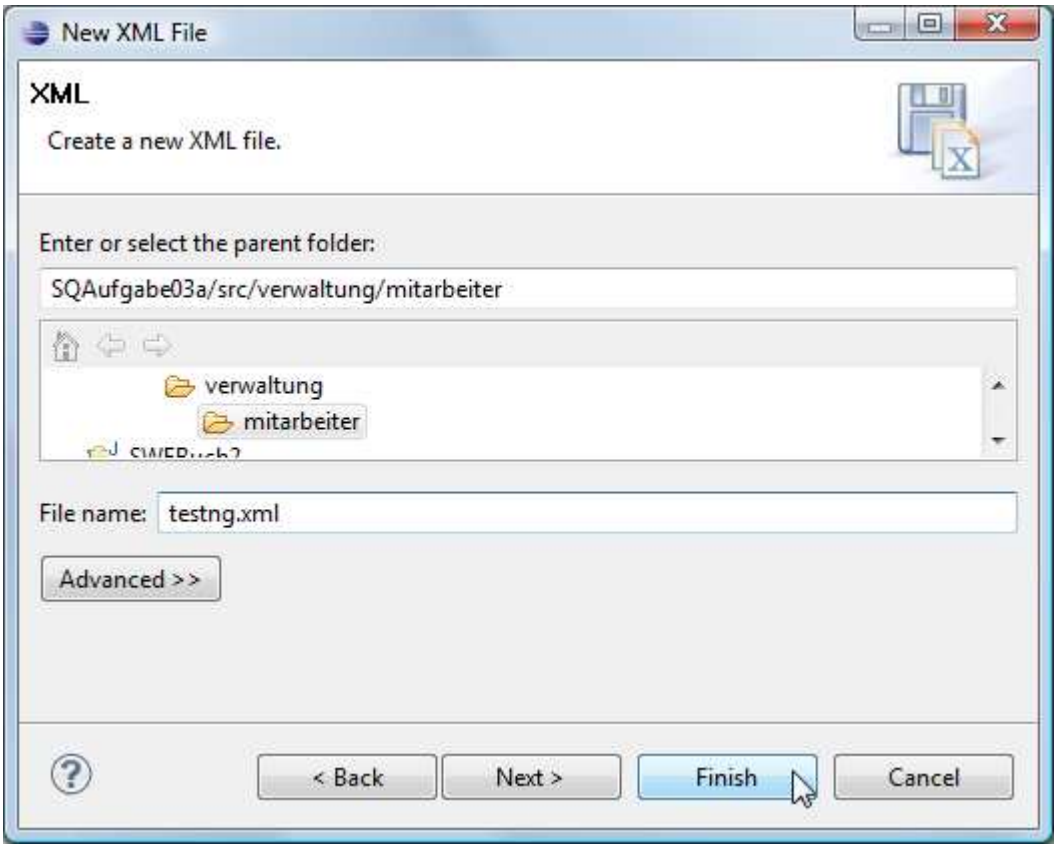
Hier wird der Punkt „XML“ aufgeklappt und „XML File“ gewählt und „Next>“ gedrückt.



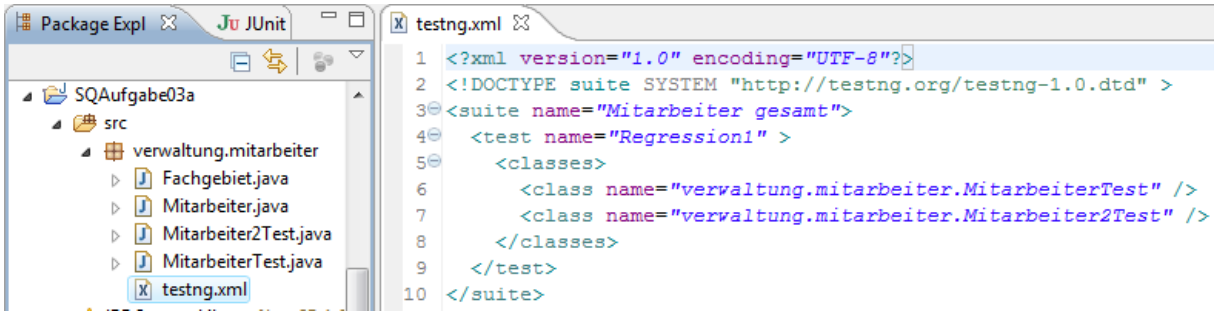
Zunächst wird im mittleren Browser ein passendes Verzeichnis, z. B. im src-Verzeichnis das Paket mit den zu testenden Klasssen gewählt. Generell erlaubt es TestNG. dass mehrere XML-Dateien existieren, die dann zur Testausführung ausgewählt werden müssen.



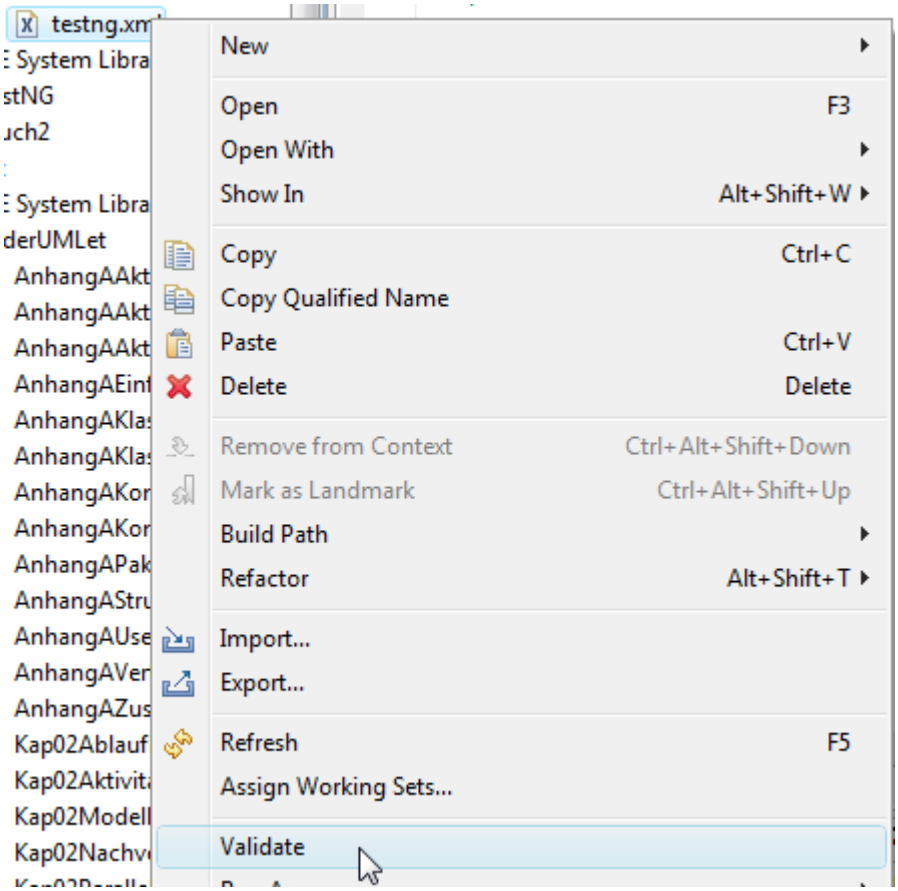
Als „File name“ wird testng.xml eingegeben und „Finish“ gedrückt.



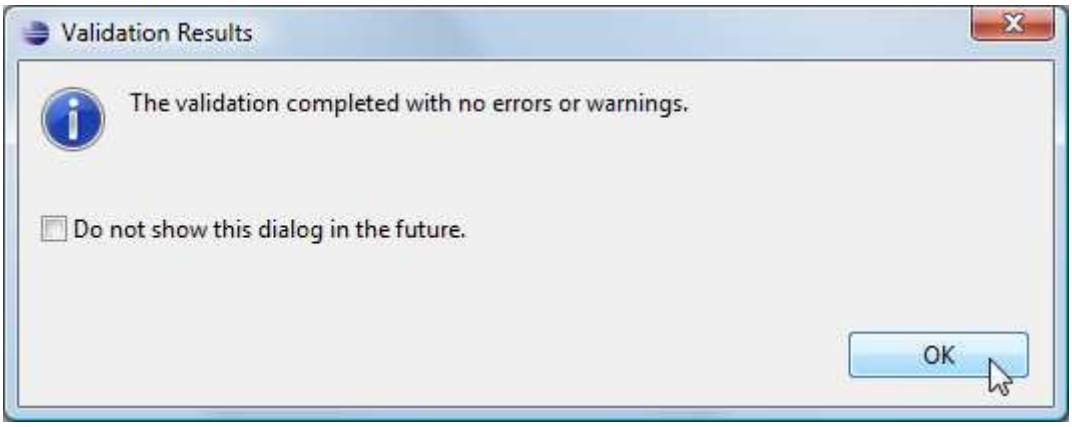
Nun kann das XML-File geschrieben werden.



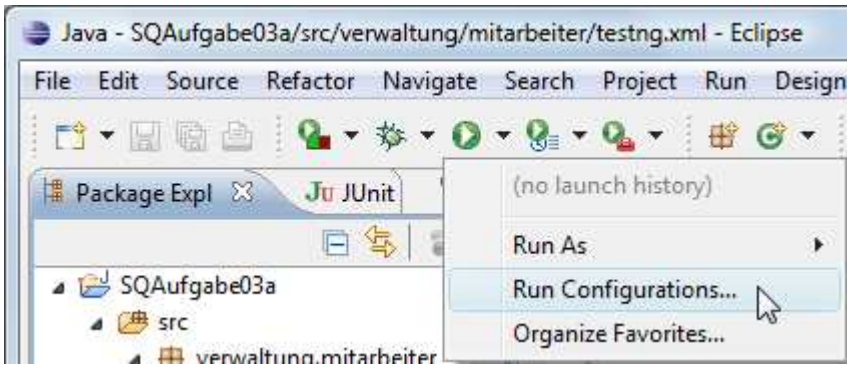
Mit einem Rechtsklick auf der Datei textng.xml kann man die Datei validieren und ihr ordnungsgemäße Form überprüfen.



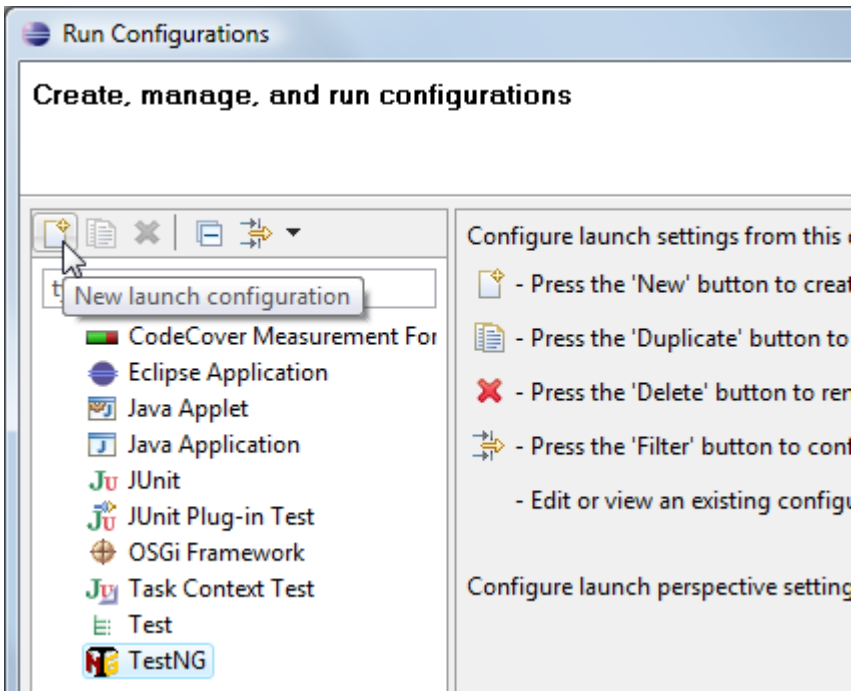
Man erhält entweder Fehlerinformationen, die auch im Editor markiert werden, oder die Nachricht, dass die Datei in Ordnung ist. Diese Meldung sollte man nicht abschalten, da man sonst keine klare Meldung bekommt, dass die Prüfung überhaupt stattgefunden hat.



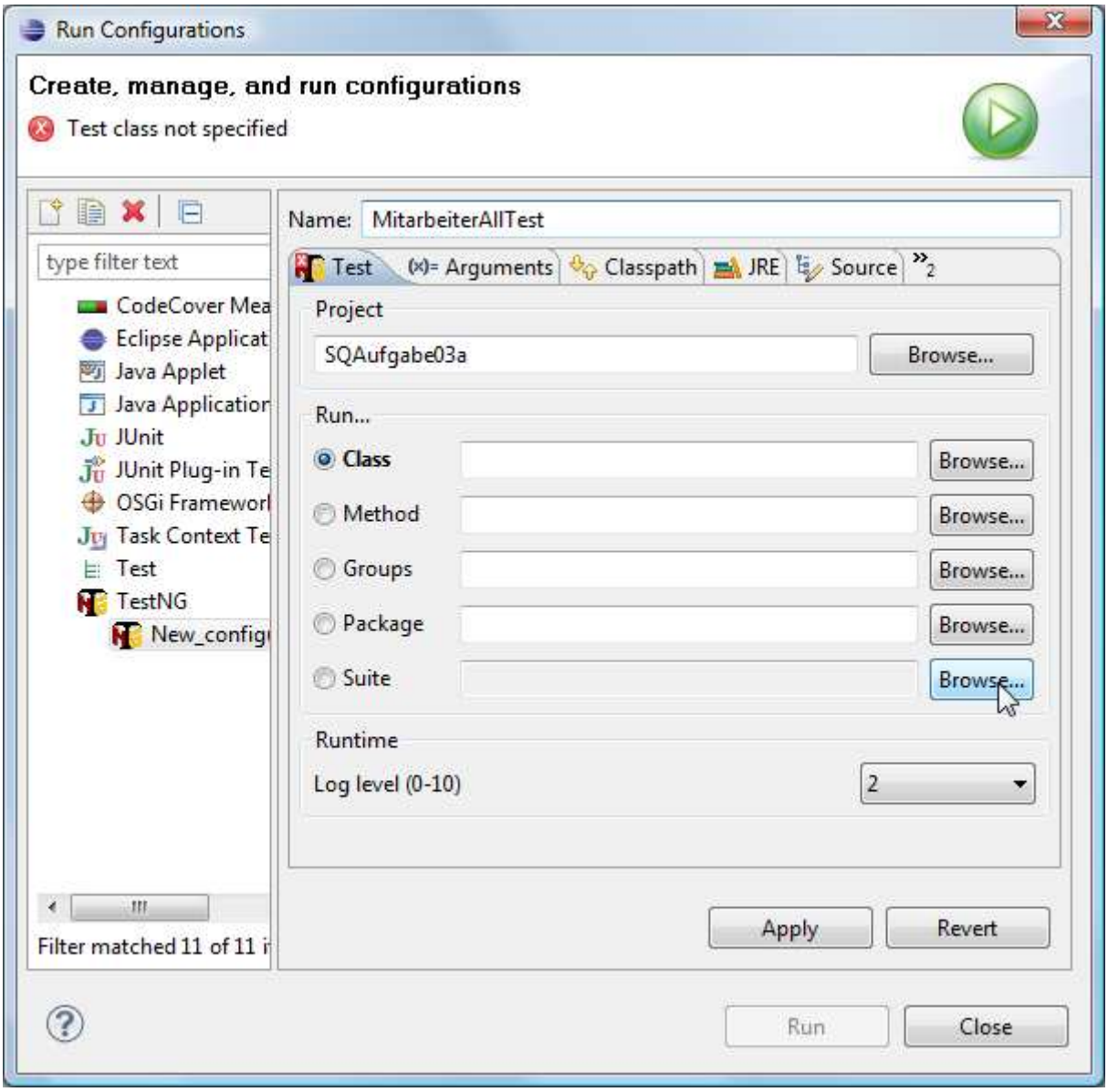
Um die zur XML-Datei gehörenden Tests auszuführen, gibt es wieder verschiedene Varianten. Wenn sich testng.xml im aktiven Editorfenster befindet, gibt es eine Möglichkeit durch das Drücken des Pfeils nach unten neben dem „Run“-Button  eine neue Startkonfigurationsdatei über „Run Configurations...“ zu wählen.



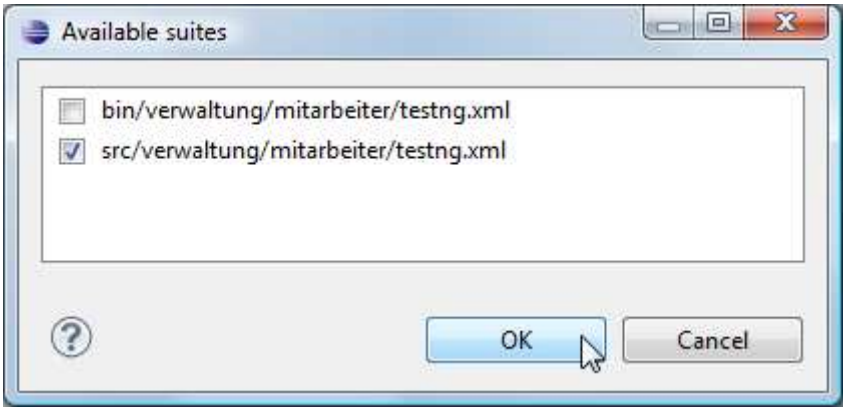
Hier wählt man zunächst rechts TestNG und drückt dann oben den Knopf „New launch Configuration“.



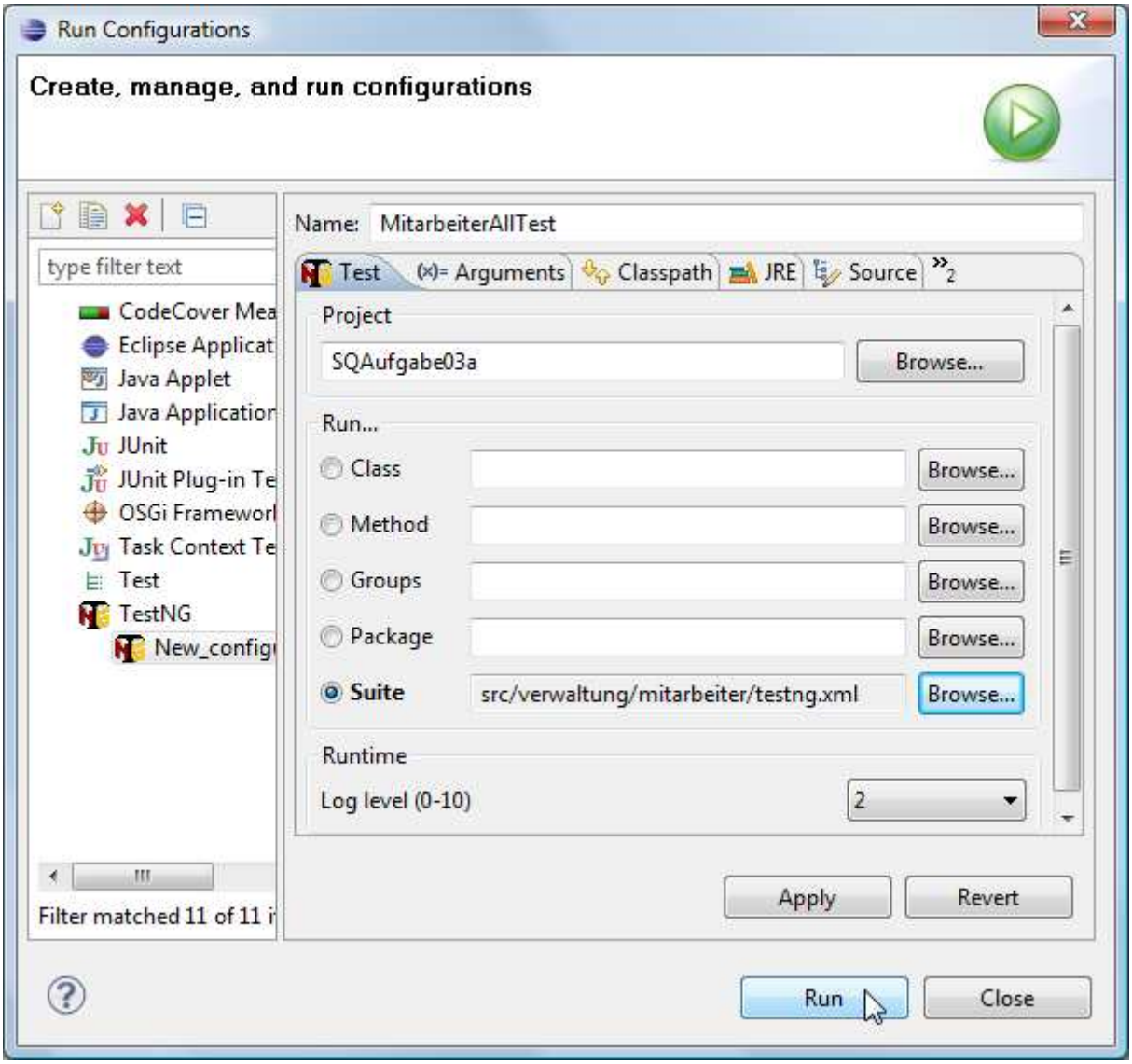
Man kann der Konfiguration unter „Name“ einen Namen geben“. Im mittleren Fenster unter „Run“ kann man dann zusammenstellen welche Tests laufen sollen. Man erkennt die sehr flexiblen Möglichkeiten hier Tests aus verschiedenen Quellen zusammen zu stellen. Um Elemente auszuwählen, wird rechts der Knopf „Browse“ gedrückt. Man erhält dann ein Auswahlfenster mit potenziellen Elementen, die zu dieser Konfiguration hinzugefügt werden sollen. In diesem Beispiel soll die Datei testng.xml genutzt werden, deshalb wird der Knopf im Bereich „Suite“ gedrückt



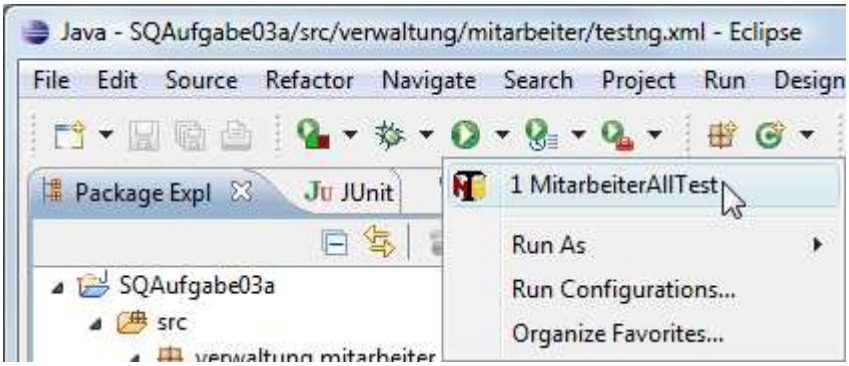
Man kann dann die interessanten testng.xml-Dateien mit einem Haken links auswählen. In den bin-Ordnern stecken Kopien der XML-Datei, deshalb müssen dies nicht angewählt werden. Die Auswahl wird dann mit „OK“ abgeschlossen.

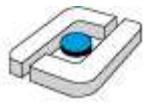


Im nächsten Schritt kann man die neu erstellte Konfiguration mit „Run“ ausführen.



Die zuletzt ausgeführten Konfigurationen können immer über den kleinen Pfeil nach unten neben dem Run-Knopf ausgewählt werden. Alternativ kann man vorhandene immer über „Run Configurations...“ suchen und dann starten.



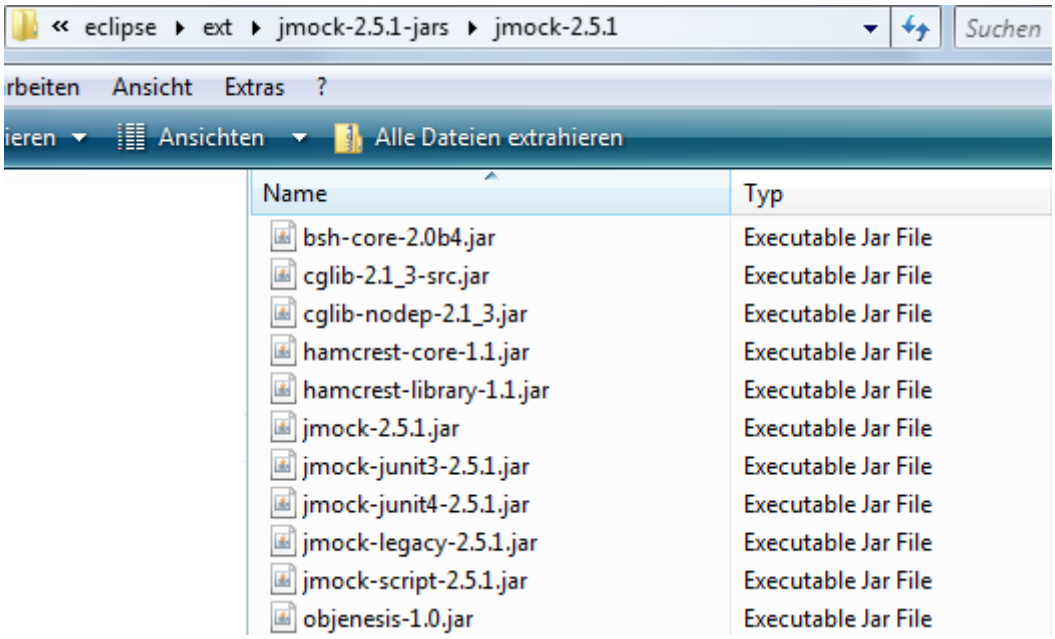


16 Einführung in JMock

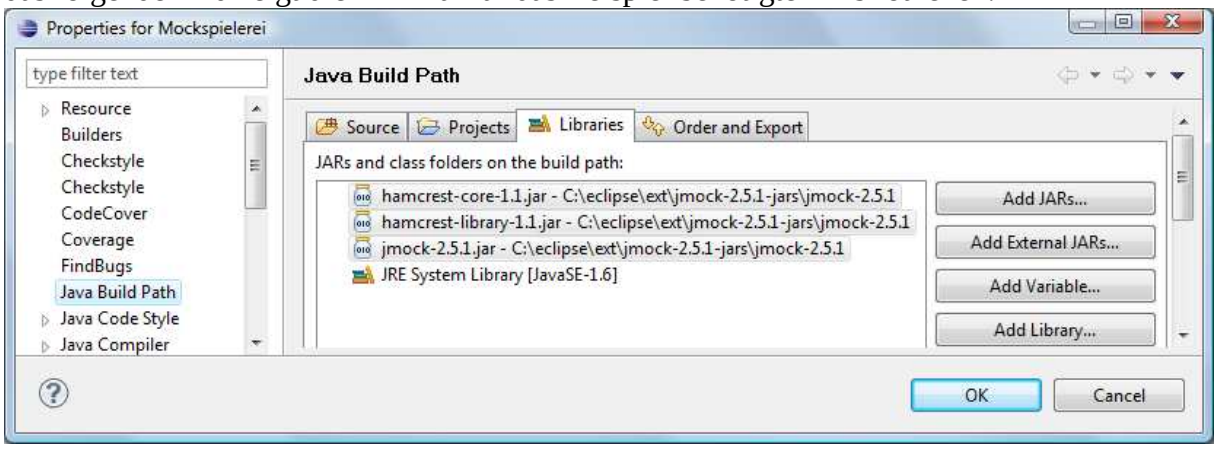
Möchte man seine eigene Software testen, ist diese häufig von Klassen abhängig, die von anderen Entwicklern stammen, die ggfls. die Programmierung noch nicht abgeschlossen haben. In diesem Fall muss man für die eigenen Tests sogenannte Mock-Klassen erstellen, also minimale Implementierungen der fehlenden Klassen, die es ermöglichen, die eigene Klasse zu testen. Neben den anzubietenden Methoden soll der Mock so gestaltet sein, dass, wenn benötigt, unterschiedliche Ergebnisse z. B. zum Testen von Fallunterscheidungen von Mock-Methoden zurückgegeben werden. Im Folgenden Programm sollte die Methode istLiquide() false und für einen anderen Test true liefern können. Die Klasse BuchungException soll als einfache Erweiterung von Exception ebenfalls vorliegen.

```
package verwaltung.finanzen;

public class Buchung {
	public static LogDatei logging;

	public synchronized void abbuchen(int id, Konto konto, int betrag)
			throws BuchungsException {
		if (konto.istLiquide(betrag)) {
			konto.abbuchen(betrag);
			logging.schreiben(id + " bearbeitet");
		} else {
			logging.schreiben(id + " abgebrochen, insolvent");
			throw new BuchungsException("insolvent");
		}
	}
}
```

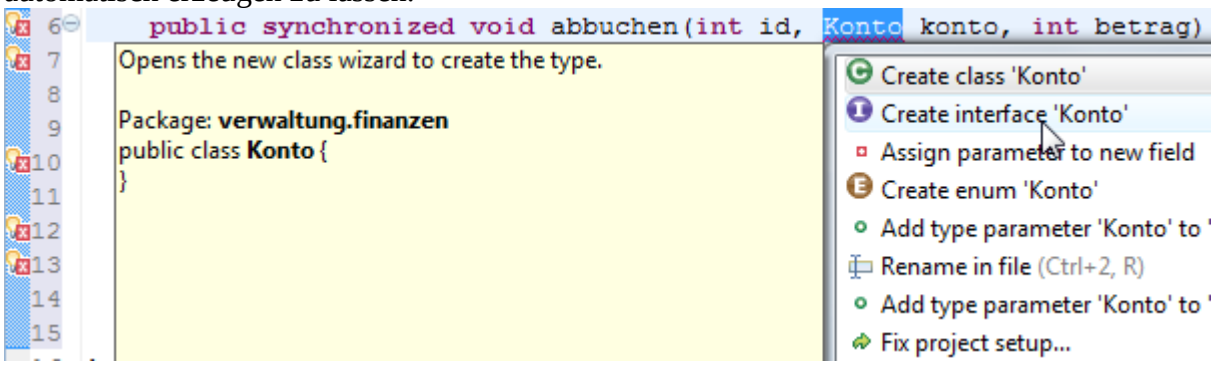
Statt die Mock-Klassen von Hand zu implementieren, was auch eine sinnvolle Variante ist, kann man hier Hilfswerkzeuge nutzen, die es Einem erlauben, die Mock-Erzeugung dynamisch innerhalb von Testfällen durchzuführen. Ein solches Werkzeug ist JMock, das unter http://www.jmock.org/download.html erhältlich ist. Man beachte, dass sich einige Jar-Dateien nach dem Auspacken im Verzeichnis jmock-2.5.1 (Nummer abhängig von der benutzen Version) befinden, die ggfls. ins Java-Projekt mit eingebunden werden müssen.



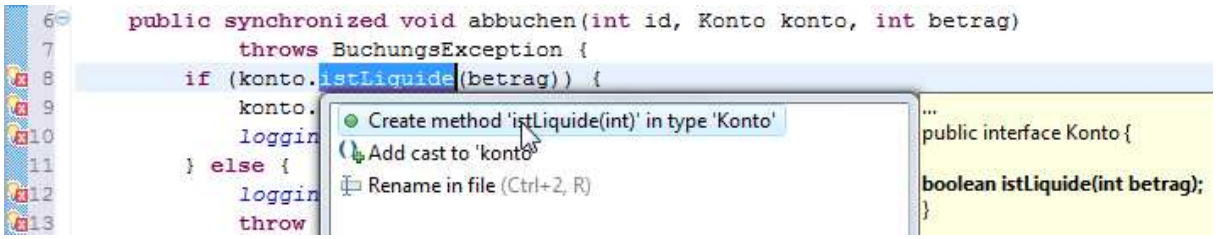
Die Möglichkeiten Jar-Dateien einzubinden wurden bereits im Abschnitt 7.10 beschrieben, das folgende Bild zeigt die minimal für das Beispiel benötigten Bibliotheken.



Für jede fehlende Klasse muss ein Interface mit den benötigten Methoden angelegt werden. Macht man einen Linksklick auf die Fehlermeldung, erhält man die Möglichkeit das Interface automatisch erzeugen zu lassen.

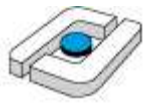


Fehlende Methoden können ähnlich ergänzt werden.



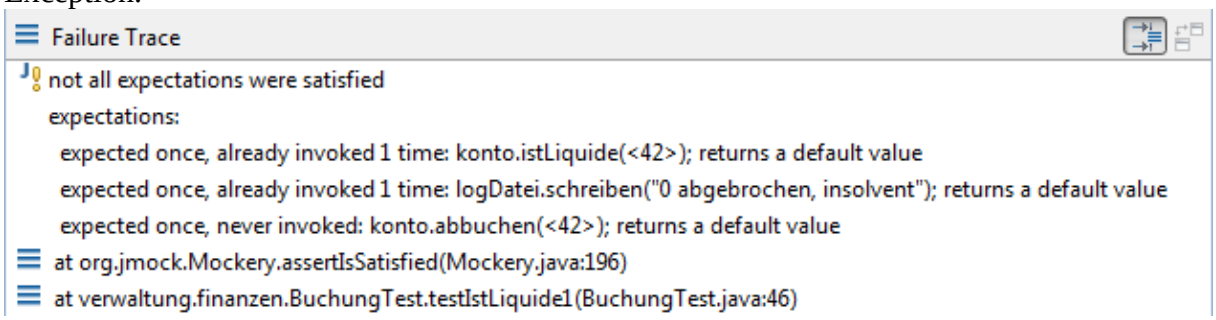
Das folgende Beispiel zeigt eine sehr kurze Einführung in die JMock-Nutzung, die detaillierter im Cookbook von der JMock-Seite nachgelesen werden kann.

```
package verwaltung.finanzen;

import org.jmock.Expectations;
import org.jmock.Mockery;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class BuchungTest {

  private Mockery context = new Mockery();
  private Buchung buchung;
  private final int BETRAG1=42;

  @Before
  public void setUp() throws Exception {
    buchung = new Buchung();
  }

  @Test
  public void testIstLiquide1(){
    final Konto k = context.mock(Konto.class);
    Buchung.logging = context.mock(LogDatei.class);
    context.checking(new Expectations(){
      {
        oneOf(k).istLiquide(BETRAG1);
      }
    });
    context.checking(new Expectations(){
      {
        oneOf(Buchung.logging).schreiben("0 abgebrochen, insolvent");
      }
    });
    /*
    context.checking(new Expectations(){
      {
        oneOf(k).abbuchen(42);
      }
    });
    */
    try {
      buchung.abbuchen(0, k, BETRAG1);
```



```
            Assert.fail("fehlender Abbruch");
          } catch (BuchungsException e) {
            context.assertIsSatisfied();
          }
        }
      }
```

Man erkennt das zentrale Mockery-Objekt zur Steuerung der Mock-Nutzung. Weiterhin kann mit der Methode mock(Class-Object) ein beliebiges Mock-Objekt einer Klasse, genauer eines Interfaces, angelegt werden. Erwartete Methodenaufrufe von Mock-Objekten sind vor ihrem Auftreten als Expectation-Objekt anzugeben. Das Beispiel zeigt zwei erwartete Aufrufe, neue Expectations-Objekte werden einfach ergänzt. Danach kann der Test in gewohnter Form durchgeführt werden. Da der Test erfolgreich durchläuft, kann man erahnen, dass ohne weitere Verfeinerung der Expectations-Objekte, einer zentralen Idee von JMock, die istLiquide-Mehode false als Ergebnis liefert. Mit der Methode context.assertIsSatisfied(); wird geprüft, dass jede der Expectations auch genutzt wurde. Wird im Beispiel die auskommentierte dritte Expectation ergänzt, liefert die assertIsSatisfied-Methode die folgende Exception.



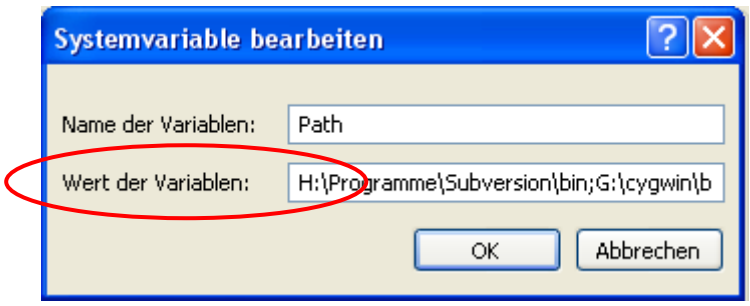
17 Versionsmanagement mit Subversion

Subversion ist ähnlich wie CVS ein frei nutzbares und in vielen Projekten eingesetztes Versionsmanagementwerkzeug. Typischerweise werden die Originaldateien auf einem zentralen Server verwaltet, es ist aber genauso möglich Subversion lokal auf dem eigenen Rechner zu nutzen. Hier wird weiterhin die Integration mit Windows gezeigt.

Die ursprüngliche Quellseite dieses Werkzeugs ist http://subversion.tigris.org, wobei die Version 1.4.3 genutzt wird (svn-win32-1.4.3.zip). Dazu gibt es ein freies, gut geschriebenes Buch unter http://svnbook.red-bean.com, dessen Lektüre zumindest der ersten drei Kapitel für das Projekt unerlässlich ist.

17.1 Subversion-Installation

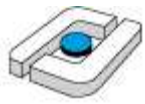
Das Programm lässt sich durch „Extract here“ hier beispielhaft in H:\programme auspacken. Der entstehende Ordner svn-win32-1.4.3 sollte in Subversion umbenannt werden. Nach der Installation wird das zugehörige bin-Verzeichnis in die Pfadvariable eingetragen.



Falls Sie cygwin auf ihrem Rechner installiert haben, muss der Subversion-Pfad vor dem cygwin-Pfad stehen, da cygwin typischerweise bereits eine veraltete svn-Version enthält. Dies können Sie in einer DOS-Box (cmd-Shell) wie folgt überprüfen.

```
G:\Dokumente und Einstellungen\Administrator>svn --version
svn, Version 1.4.3 (r23084)
   übersetzt Jan 18 2007, 07:47:40
Copyright (C) 2000-2006 CollabNet.
Subversion ist Open Source Software, siehe http://subversion.tigris.org/
Dieses Produkt enthält Software, die von CollabNet (http://www.Collab.Net/) entwickelt
wurde.
Die folgenden ZugriffsModule (ZM) für Projektarchive stehen zur Verfügung:
* ra_dav : Modul zum Zugriff auf ein Projektarchiv über das Protokoll WebDAV (DeltaV)
  - behandelt Schema »http«
  - behandelt Schema »https«
* ra_svn : Modul zum Zugriff auf ein Projektarchiv über das svn-Netzwerkprotokoll
  - behandelt Schema »svn«
* ra_local : Modul zum Zugriff auf ein Projektarchiv auf der lokalen Festplatte
  - behandelt Schema »file«
```

Danach kann Subversion wie in der Veranstaltung beschrieben genutzt werden.



17.2 Subversive-Installation

Subversive sorgt für die Integration von Subversion in Eclipse und kann unter `http://www.eclipse.org/subversive/` bezogen werden. Generell erfolgt eine Installation über das Update-Verfahren von Eclipse oder über das Entpacken der zugehörigen Zip-Files mit „Extract here“ im Eclipse-Ordner. Für Subversive werden zwei Dateien benötigt, die z. B.

```
Subversive-incubation-0.7.0.v20080218.zip
Subversive-connectors-2.0.0.v20080214.zip
```

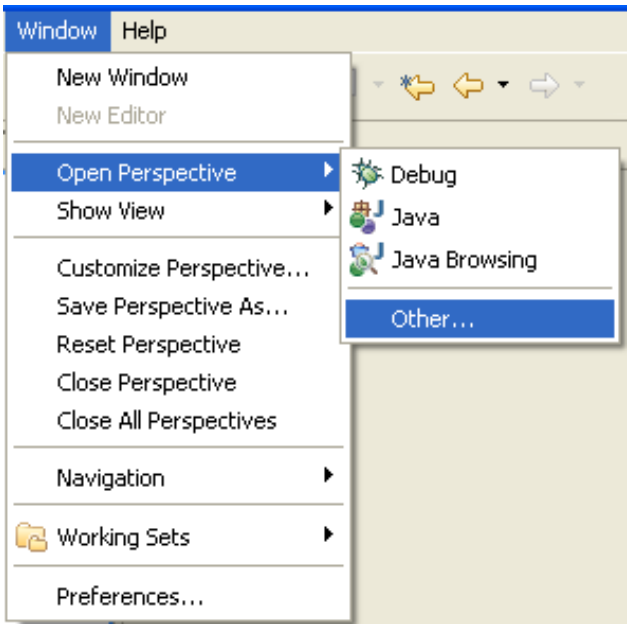
heißen können. Subversive ist im zur Verfügung gestellten Eclipse-Paket integriert.

17.3 Einrichtung des Projekts unter Subversion

Bevor Sie Subversion in Eclipse ausprobieren, wird dringend empfohlen, zunächst alle Möglichkeiten mit svn in einer Console (DOS-Box) auszuprobieren, da sonst die Gefahr besteht, dass Sie zwar einem „Klick hier und Klick da-Manual“ folgen können, Ihnen aber nicht wirklich klar ist, was warum passiert.

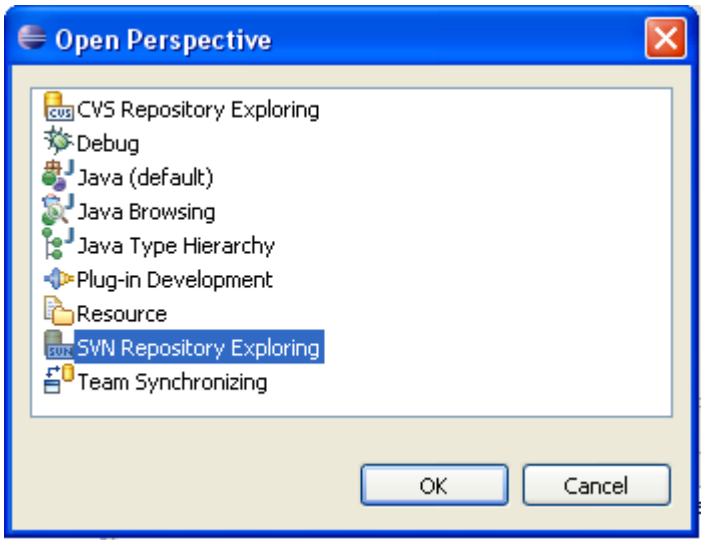
Vor der Eclipse-Nutzung muss eine sinnvolle Struktur für das Repository festgelegt werden. Diese Struktur kann direkt als Dateibaum irgendwo angelegt und dann mit `svn import` eingespielt werden. Alternativ können die Ordner auch von Eclipse aus angelegt werden (was weder Vor- noch Nachteile hat). Es wird angenommen, dass sich unter d:\repository ein funktionierendes Subversion-Repository befindet (z. B. erzeugt durch `svnadmin create d:\rrr`). Dieses Respository wird hier lokal genutzt, dabei ist zu beachten, dass eine Nutzung über http, hhtps und svn+ssh mit einer gültigen URL ebenfalls möglich ist.

Zunächst muss in Eclipse in die Subversion-Ansicht gewechselt werden. Dazu wird „Window > Open Perspective > Other“ ausgewählt.

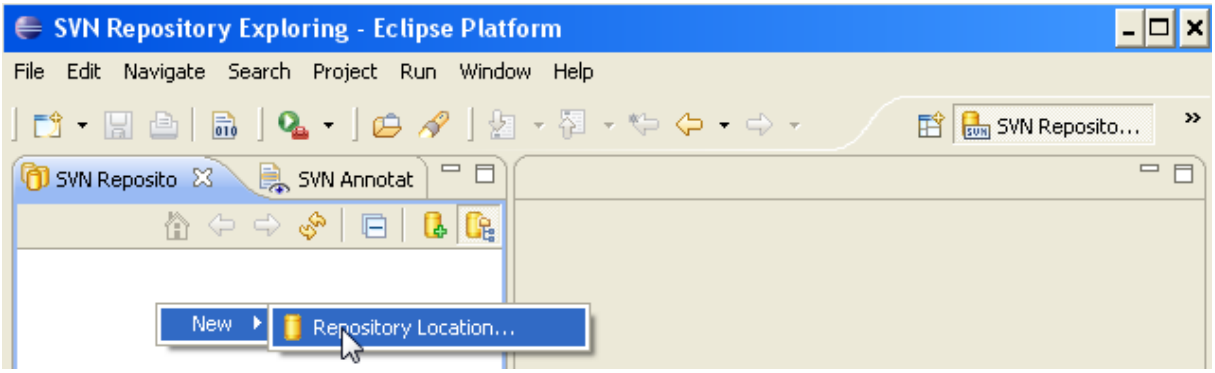


Hier kann unter dem Reiter „General“ die URL zum Repository angegeben werden. Das Protokoll am Anfang gibt an, wie der Zugriff erfolgen soll. Dabei steht file:// für ein lokales Verzeichnis. Unter Windows ist der dritte Schrägstrich und die Art der Schrägstriche im File-

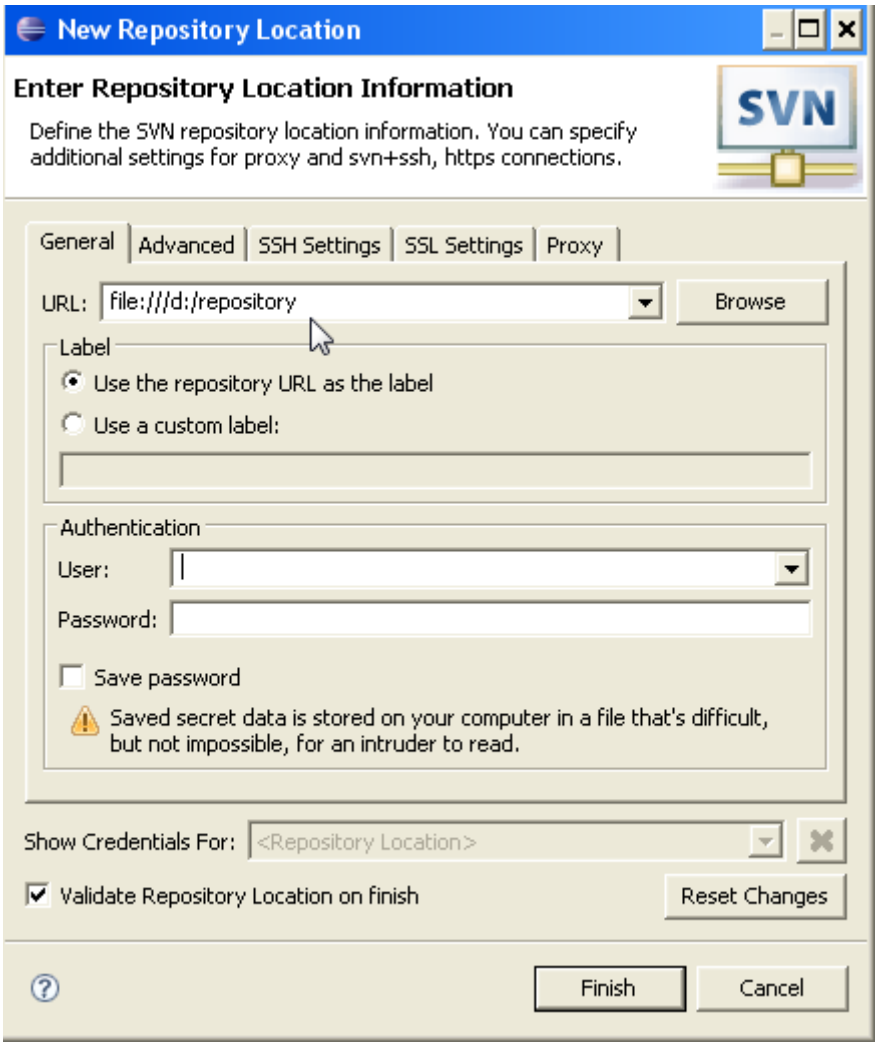
Namen zu beachten. Weiterhin sind die weiteren Reiter interessant, wenn man z. B. eine SSH-Verbindung nutzen möchte.



In der dann aufgehenden Perspektive können durch einen Rechtsklick im ganz linken Fensterbereich dann über „New > Repository Location...“ Repositories hinzugefügt werden. Nach dem Rechtsklick sieht die Auswahl wie folgt aus.

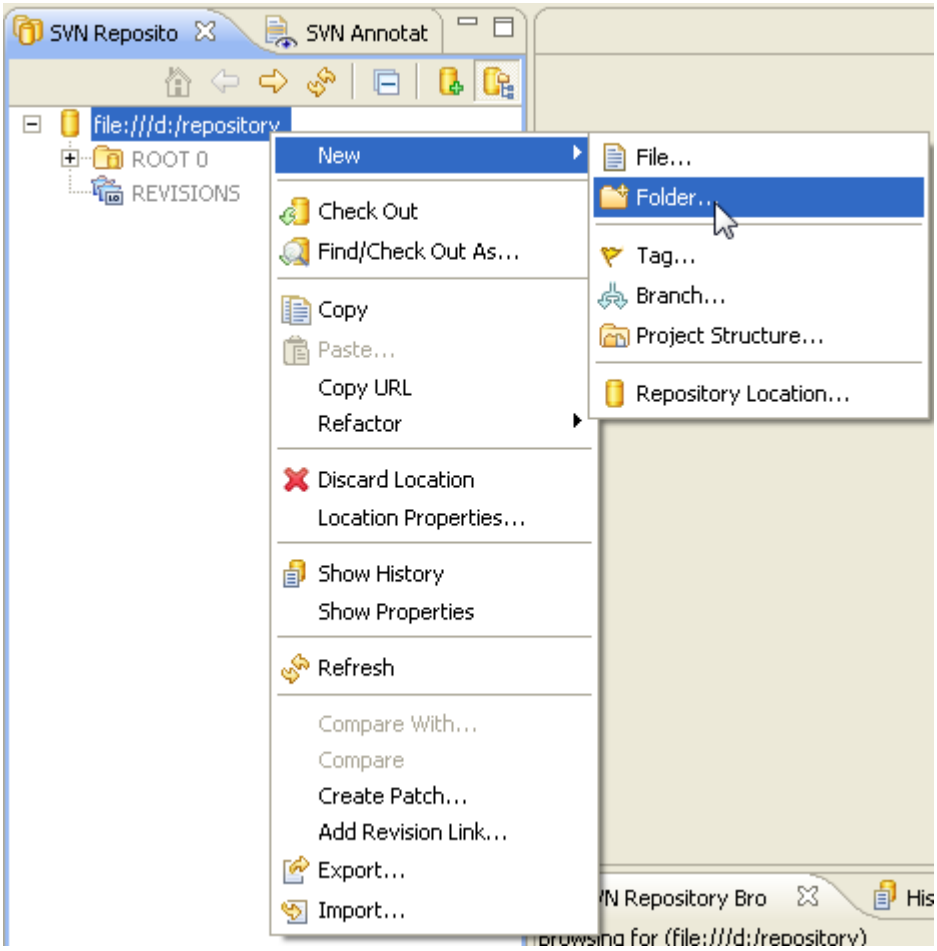


Dazu wird im folgenden Fenster dann die URL des Repositories eingegeben, hier ein lokales Verzeichnis. Man beachte die weiteren Einstellungsmöglichkeiten in den verschiedenen Reitern. Die Eingabe wird mit „Finish“ abgeschlossen.

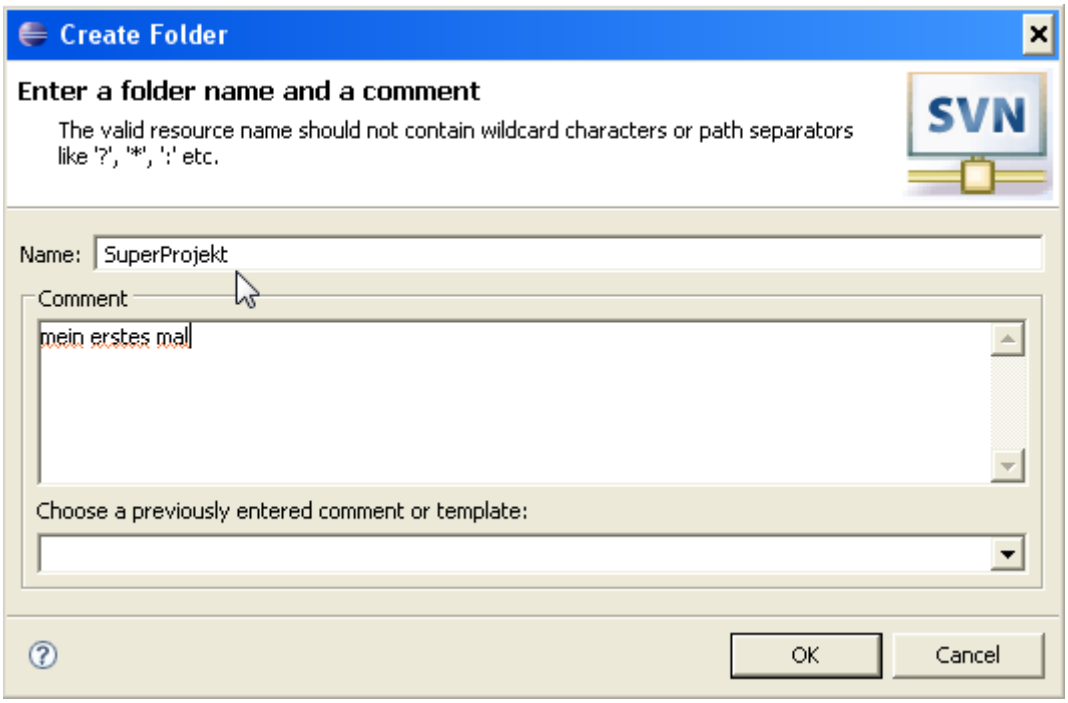


Man kann sich dann auf der linken Seite den Inhalt des Repositories ansehen und dieses auch bearbeiten. Dazu soll jetzt ein Ordner namens „SuperProjekt“ angelegt werden. Nach einem Rechtsklick auf das Repository (oder einen Ordner, falls ein Unterordner angelegt werden soll) wird dann „New > Folder...“ gewählt.

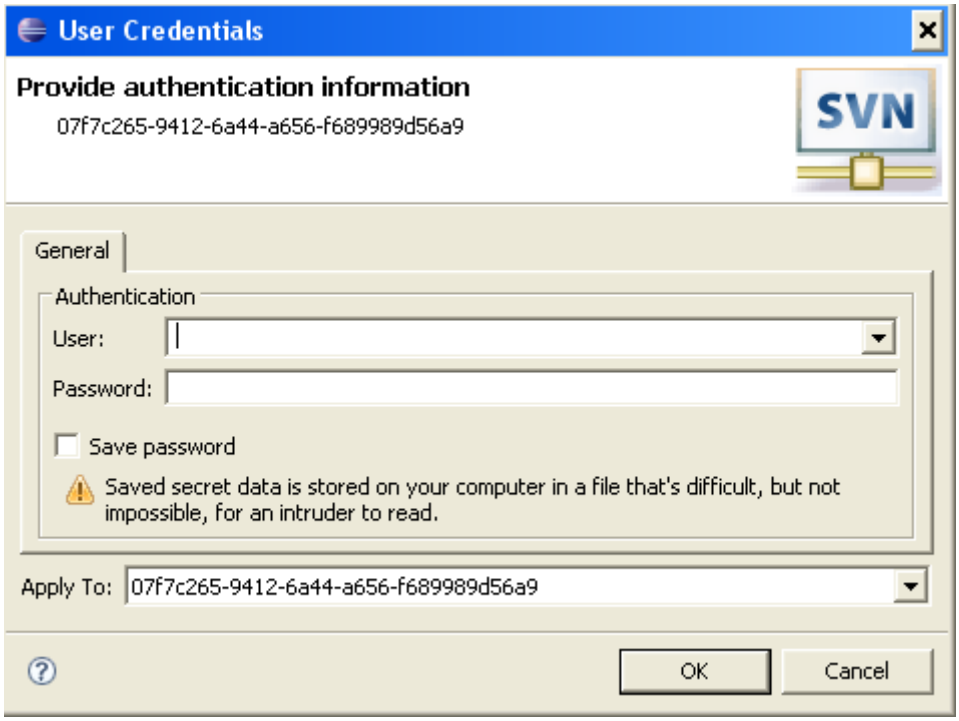
Angemerkt sei, dass man solch einen Ordner nicht benötigt, wenn man ausschließlich mit einem Java-Projekt arbeitet. Generell kann man aber Eclipse als Subversion-Client nutzen und so beliebige Dateien, wie Protokolle, verwalten.



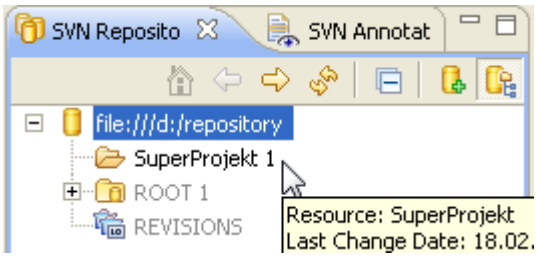
Hier kann dann der Name des neuen Projektordners angegeben werden.



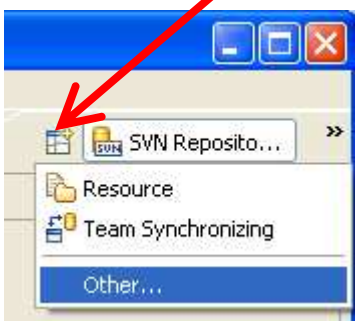
Es wird „OK“ gedrückt. Im folgenden Fenster müssen wieder die Nutzerdaten eingegeben werden. Bei einer lokalen Installation, wie hier im Beispiel, muss keine Eingabe gemacht werden, es wird direkt „OK“ gedrückt.



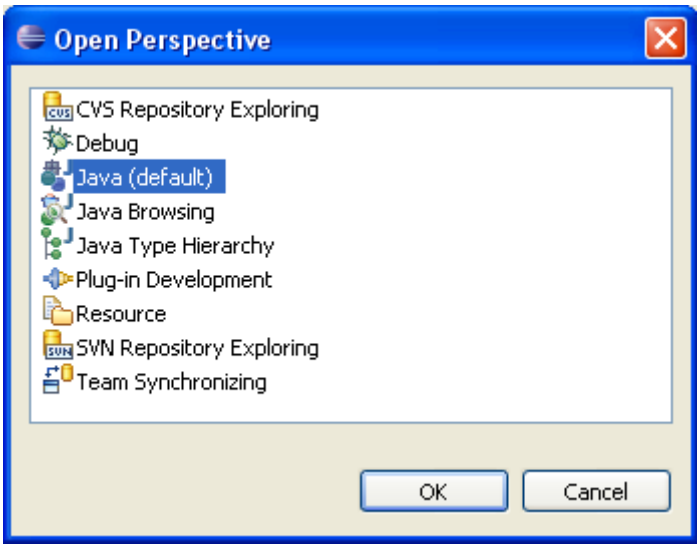
Der neue Ordner ist dann sichtbar.

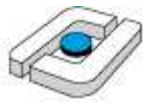


Um wieder in die Java-Perspektive zu wechseln, wird wie üblich rechts oben in Eclipse ein Linksklick auf den Knopf neben „SVN Reposito...“ gemacht und „Other“ gewählt.



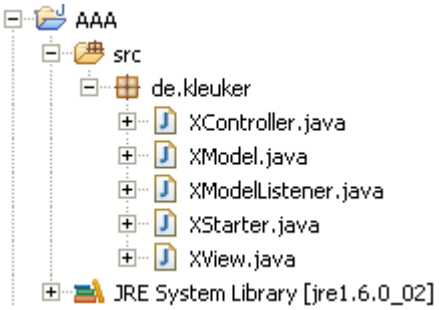
Man beachte dabei auch die Resource-Perspektive, mit der man sich das darunter liegende Dateisystem ansehen und so auch mit Subversion bearbeiten kann.


Open Perspective
CVS Repository Exploring
Debug
Java (default)
Java Browsing
Java Type Hierarchy
Plug-in Development
Resource
SVN Repository Exploring
Team Synchronizing
OK
Cancel

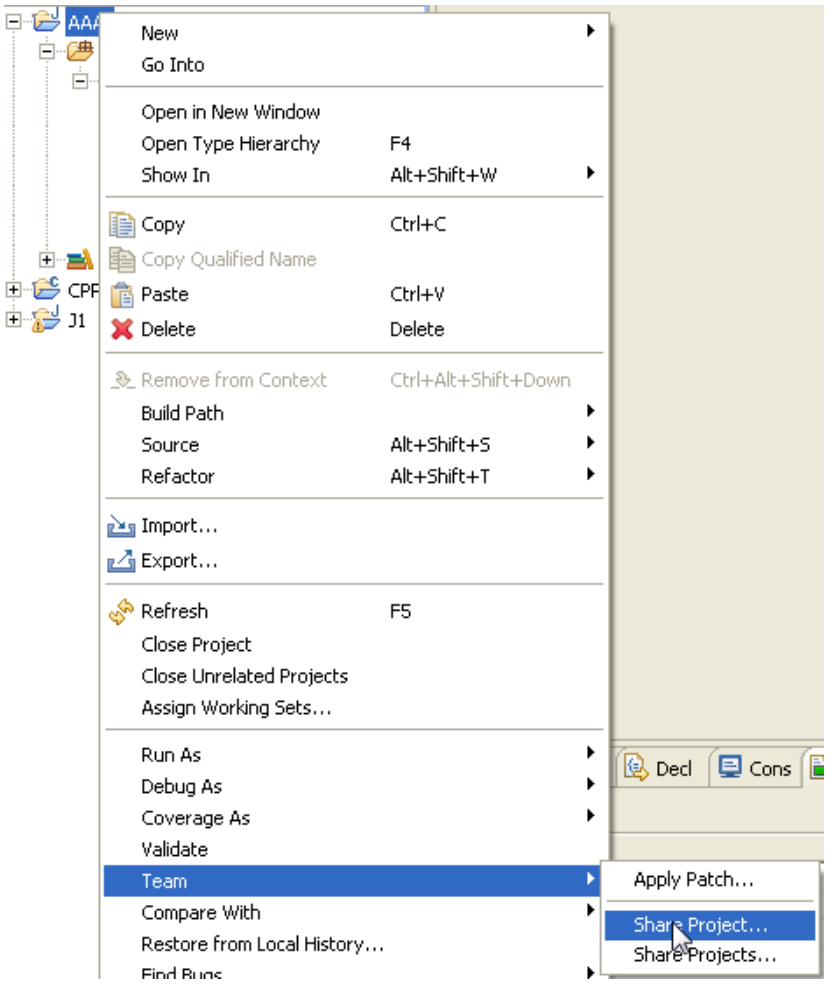


17.4 Einmalige Eclipse-Projekt-Einrichtung unter Subversion-Verwaltung

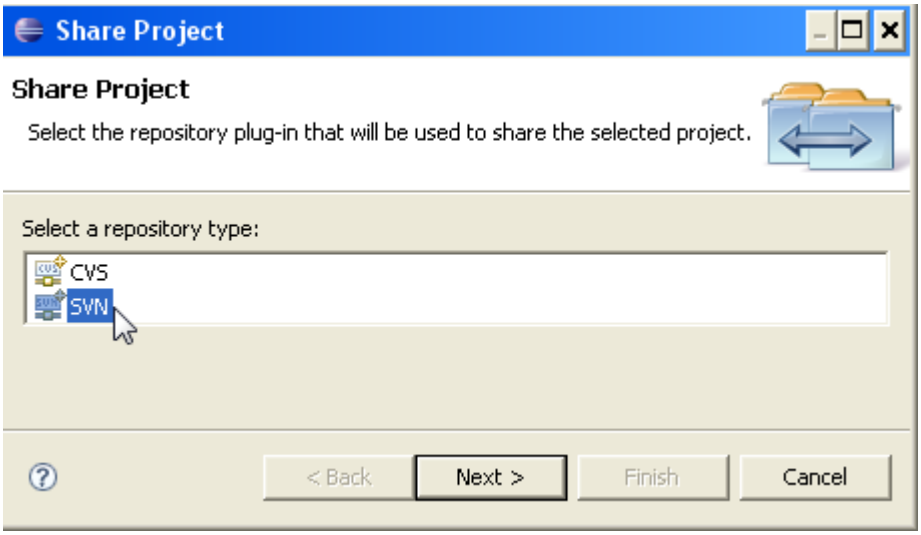
Zur Nutzung von Subversion zusammen in Eclipse empfiehlt es sich, am Anfang einmal ein Projekt in Eclipse einzurichten und dieses dann unter Versionskontrolle zu stellen. Im Beispiel wird ein Projekt AAA betrachtet, wobei sich noch keine Klassen im Projekt befinden müssen. Dieser Schritt wird nur einmal im gesamten Projekt durchgeführt.



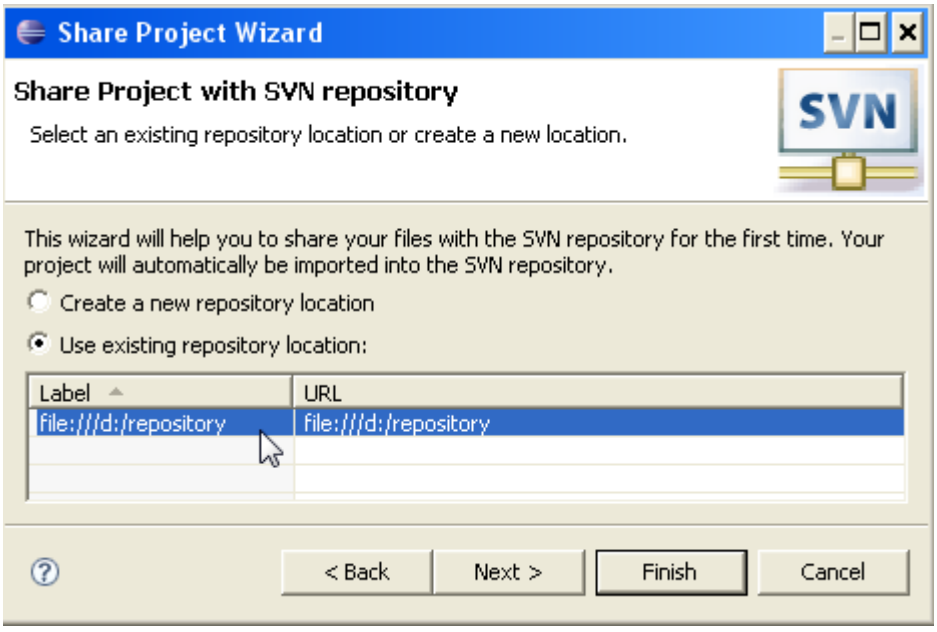
Durch einen Rechtsklick auf das Projekt kann man u. a. den Punkt „Team > Share Project...“ auswählen, was hier auch geschehen soll.



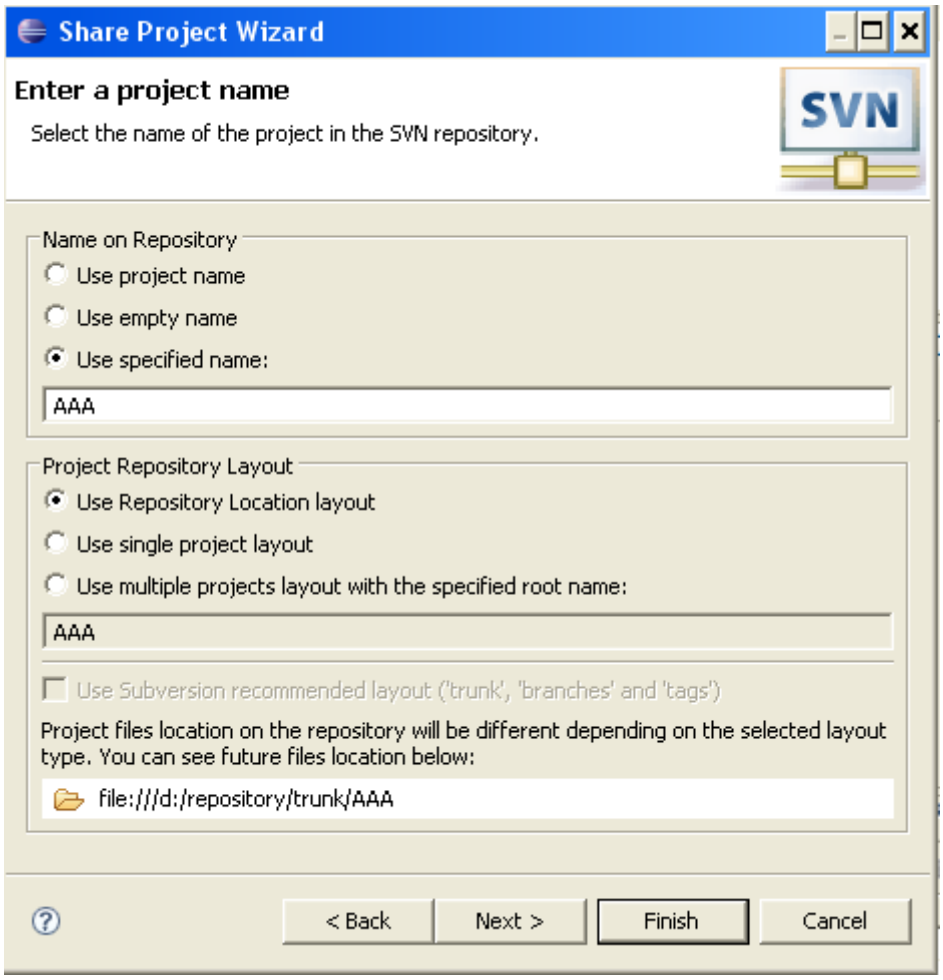
Es wird zunächst SVN ausgewählt.



Im nächsten Schritt wird dann das Repository ausgewählt. Wenn man noch nicht mit Subversion von Eclipse aus gearbeitet hat (also den beschriebenen Schritt zur Einrichtung des Repositories z. B. vergessen hat), muss man gegebenenfalls unter „Create a new repository location“ die zugehörige URL angeben.

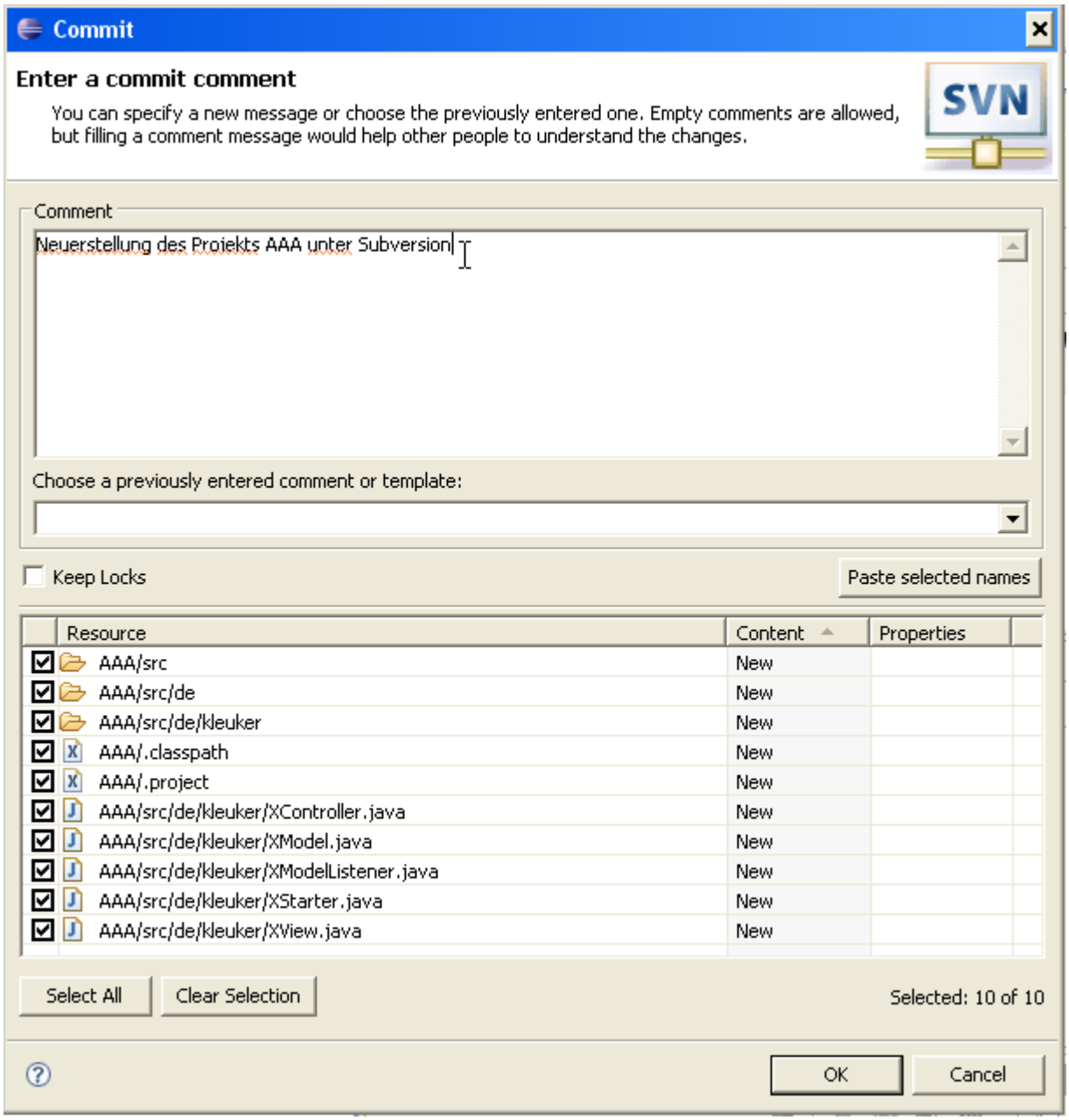


Falls man von Eclipse aus schon mit Subversion gearbeitet hat, kann man ein Repository auswählen, wie hier, und dann „Next>“ wählen.



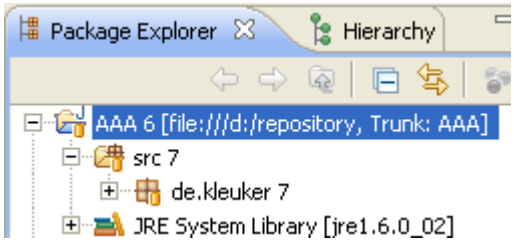
Die vorgeschlagenen Einstellungen können so übernommen werden. Wollte man nur einzelne Ordner einchecken, wäre das „Use single project layout“ zu empfehlen. Bleibt man bei den Standardeinstellungen, so werden die Daten unter dem unten angegebenen Pfad abgelegt. Die Erklärung zu den Ordnern trunk, branches und tags kann dem frei erhältlichen Subversion-Buch entnommen werden. Für Anfänger reicht die Vorstellung aus, dass zunächst nur unter „trunk“ gearbeitet wird.

Wählt man „Finish“, erscheint ein Dialog, bei dem man einen Kommentar für die ersten Daten eingeben muss, wie es von Subversion bei jedem commit gefordert wird. Man beachte den unteren Teil, in dem Subclipse bereits die Möglichkeit schafft, Dateien zu kennzeichnen, die nicht unter Versionsverwaltung gestellt werden sollen. Falls Sie mit unterschiedlichen Betriebssystemen arbeiten wollen (wovon aus mehreren Gründen dringend abgeraten wird), sollte der Haken bei .classpath und .project weggenommen werden, sonst sollen die Haken auf jeden Fall stehen bleiben.



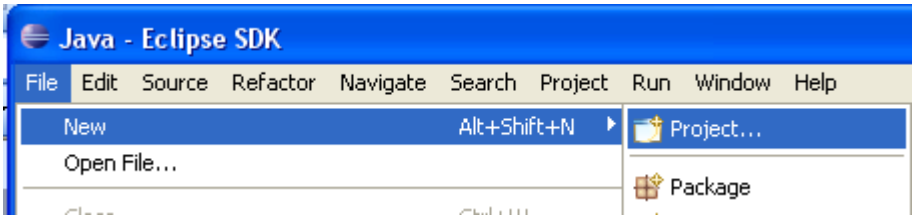
Nach einem Druck auf „OK“ werden alle selektierten Daten in das Repository eingespielt. Weiterhin wird für den zum Einspielen genutzten Projektordner automatisch ein checkout durchgeführt, so dass man in Eclipse normal mit einer Arbeitskopie weiterarbeiten kann.

Man erkennt die Versionskontrolle an der besonderen Markierung der einzelnen Ordner (die Nummern starten zunächst beim Wert 1).

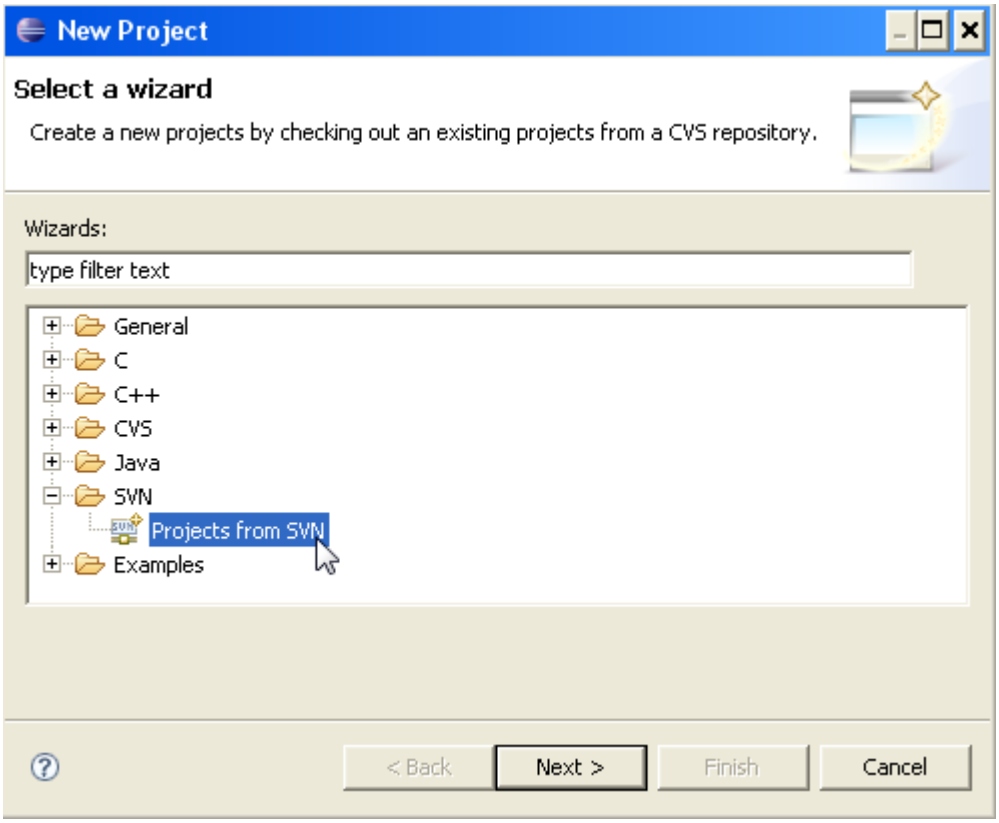


17.5 Nutzung des Eclipse-Projekts durch Projektmitarbeiter

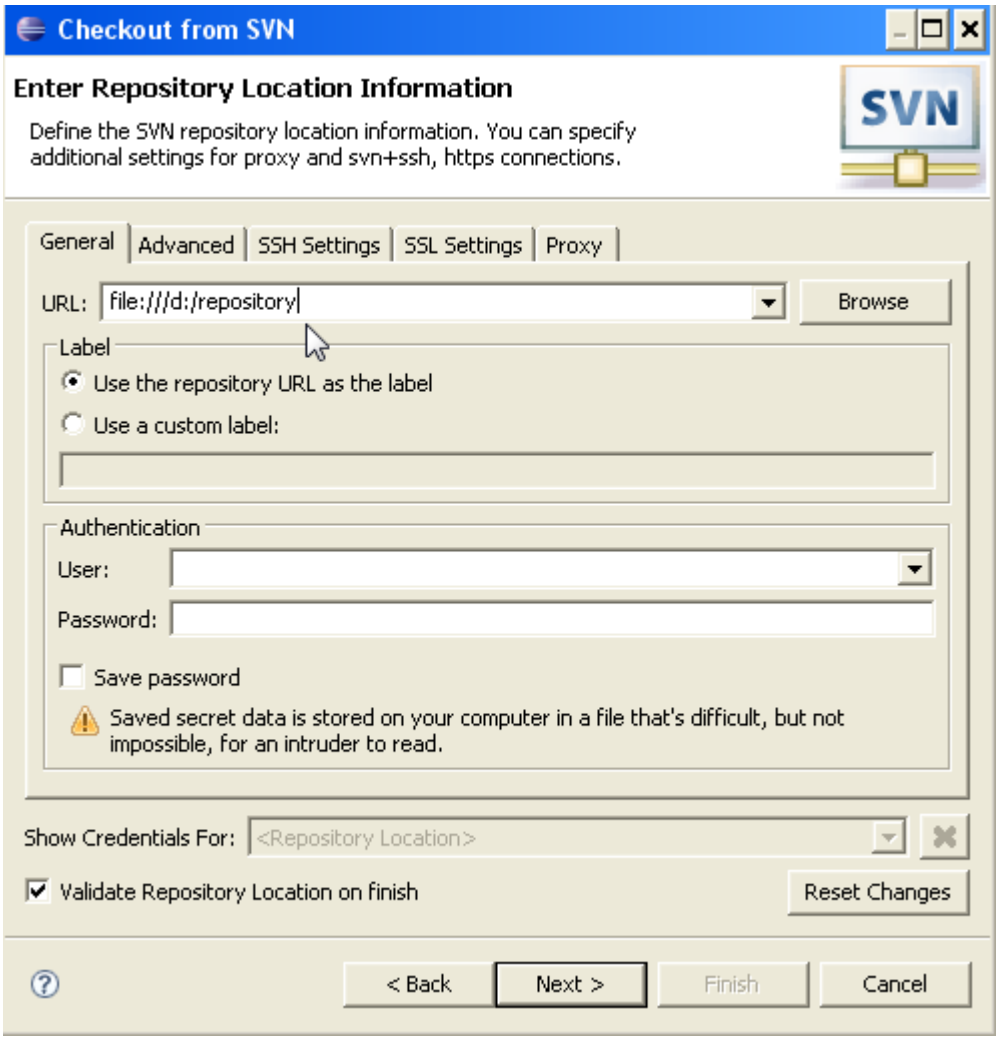
Alle weiteren Projektmitarbeiter werden das angelegte Projekt zur Bearbeitung in ein eigenes Projekt auschecken, so dass eine lokale Arbeitskopie entsteht. Dazu wird zunächst der Schritt „File > New > Project...“ ausgewählt.



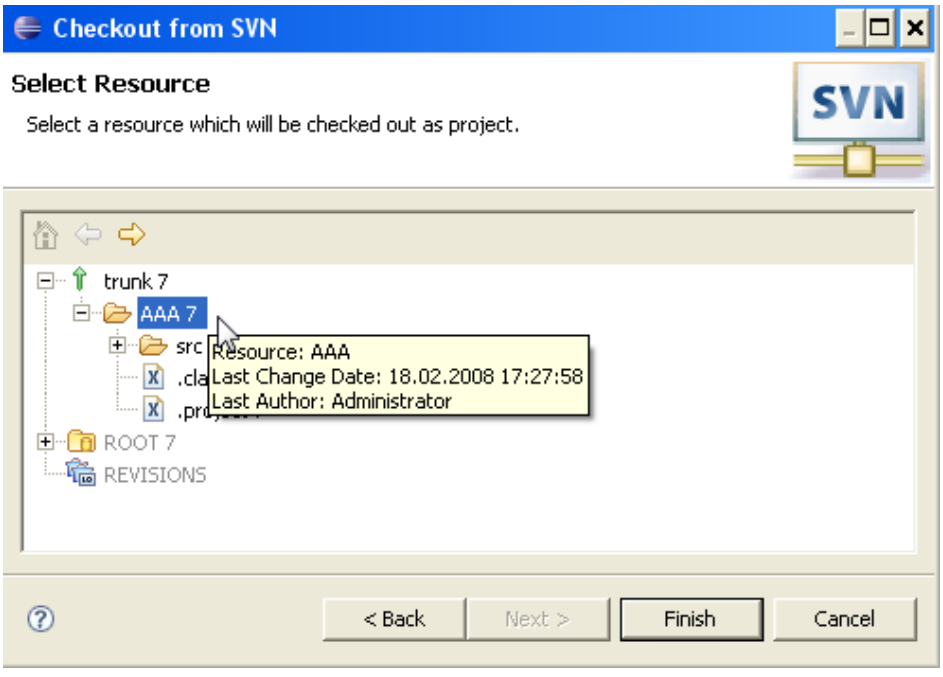
Danach wird „SVN > Checkout Projects from SVN“ ausgewählt.



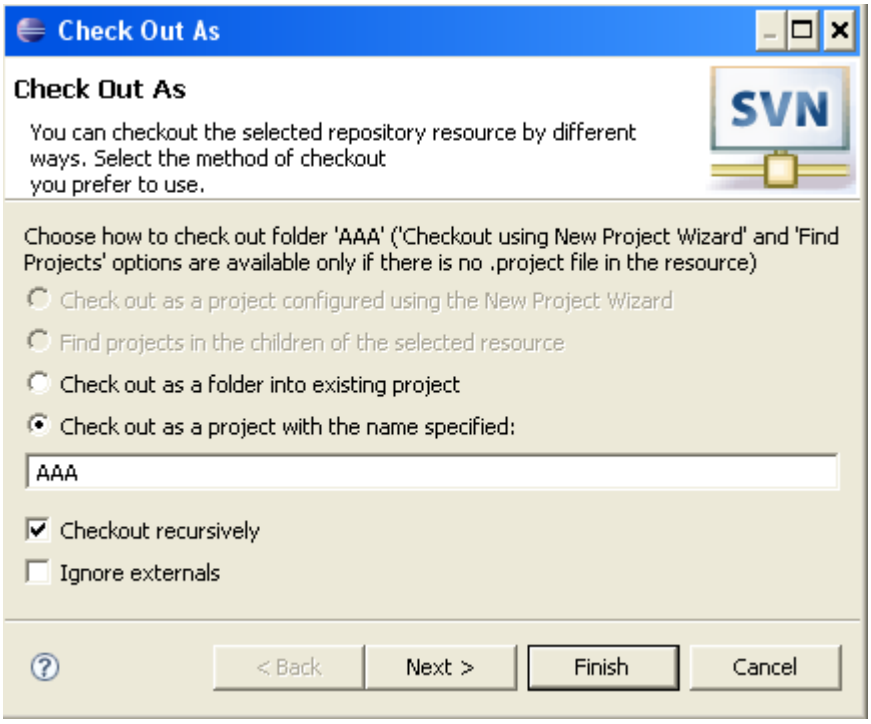
Im folgenden Fenster muss dann die Verbindung zu einem Repository aufgebaut werden.



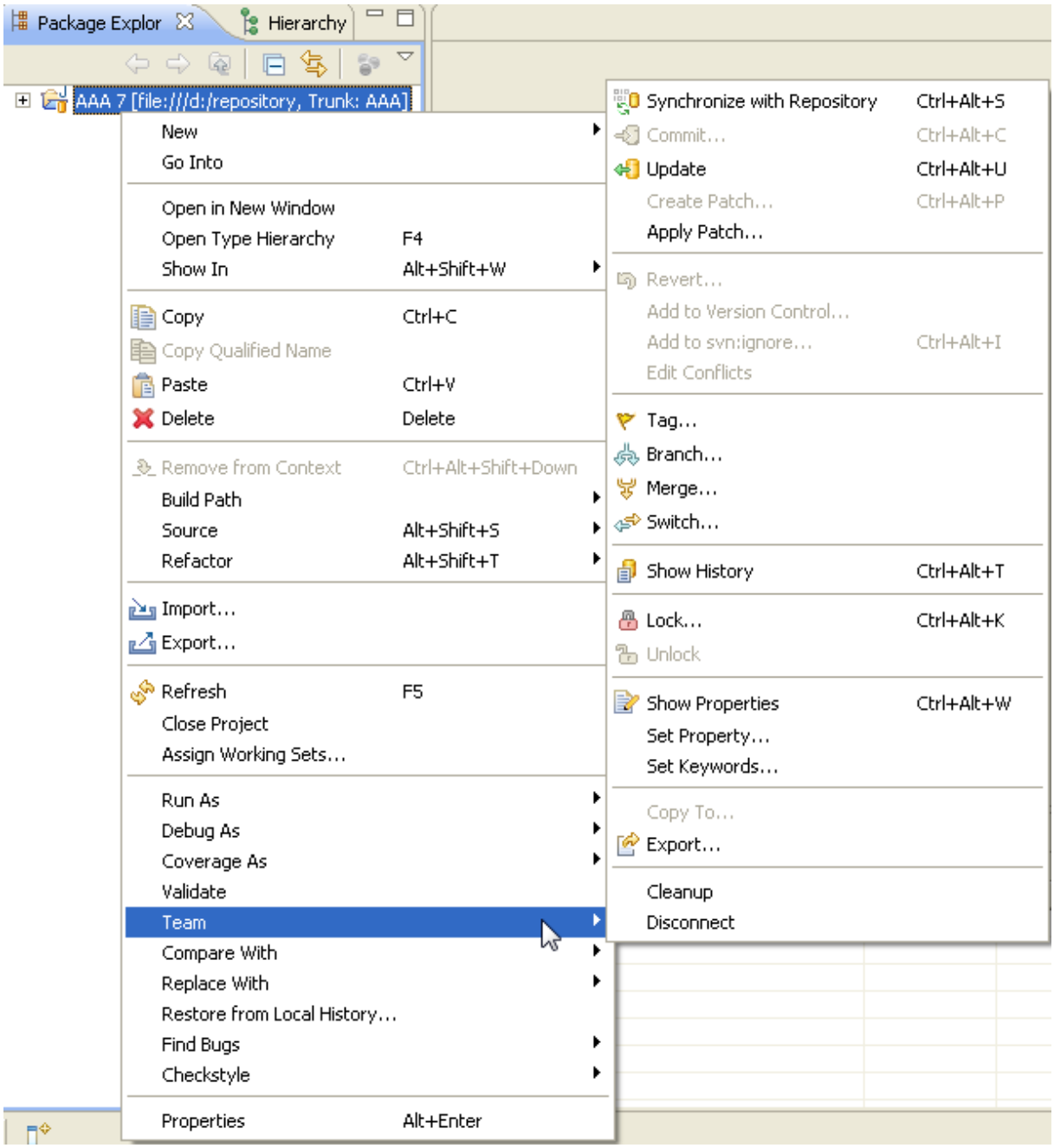
Es wird der zum Eclipse-Projekt gehörende Ordner ausgewählt und mit „Finish“ abgeschlossen.



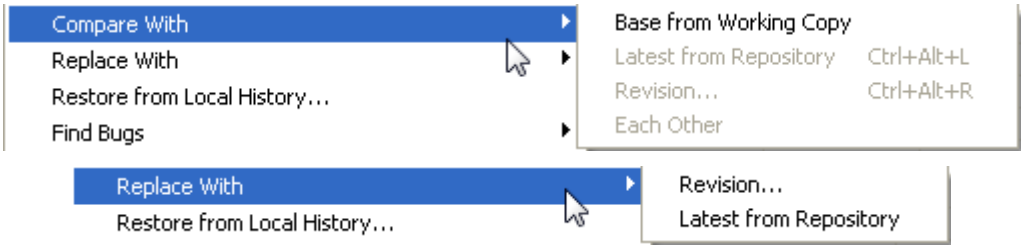
Man kann sich dann noch für einen anderen Projektnamen entscheiden, wobei die Übernahme des ursprünglichen Projektnamens meist sinnvoll ist, wozu trotzdem der Radio-Button auf „Check out as project with the name specified“ gewählt sein muss.

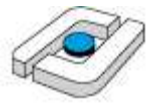


Bei der Bearbeitung des Projekts stehen die verschiedenen Subversion-Befehle von Eclipse aus zur Verfügung.

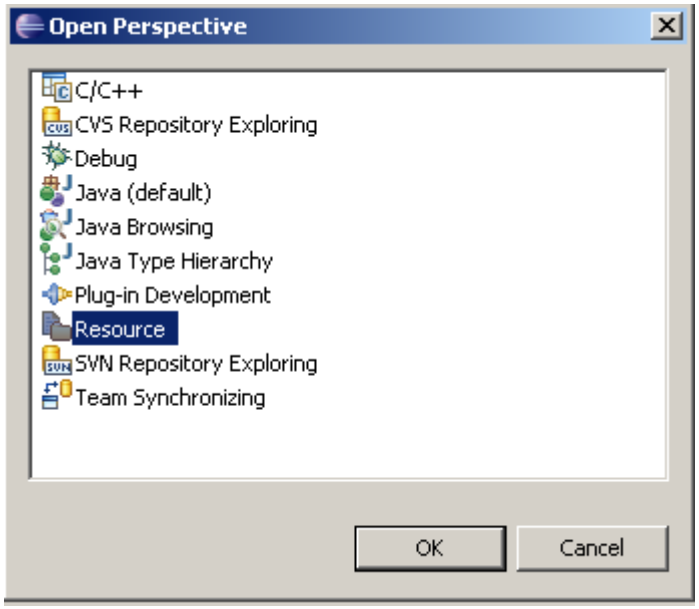


Die Punkte „Compare with“ und „Replace with“ sind besonders zu beachten, da hier u. a. Unterschiede zwischen den Versionen dargestellt werden.

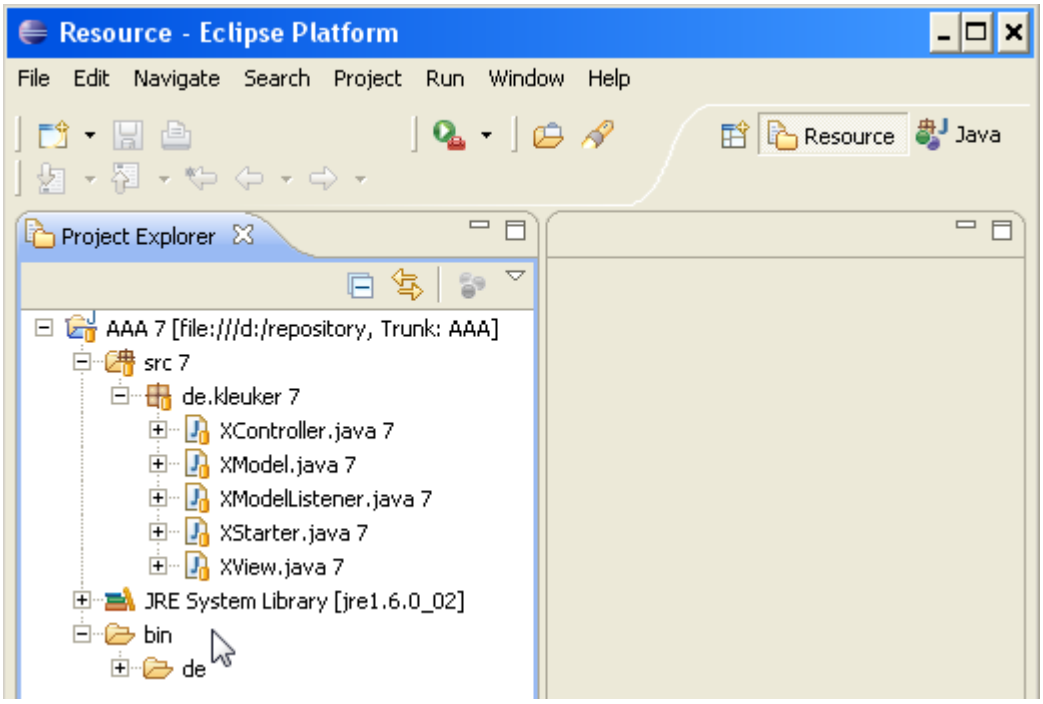




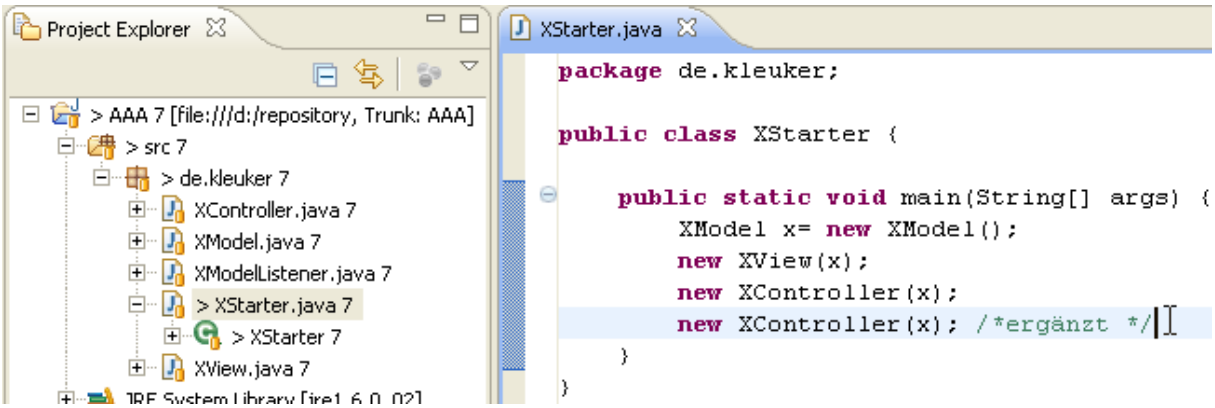
Da nicht alle Dateien in Java-Projekten angezeigt werden, kann es hier hilfreich sein, in die Resource-Perspektive zu wechseln, in der alle Dateien angezeigt werden. Alternativ kann man direkt Subversion von der Konsole aus nutzen oder das im Folgenden noch vorgestellte Werkzeug TortoiseSVN zur Synchronisation mit dem Repository einsetzen.



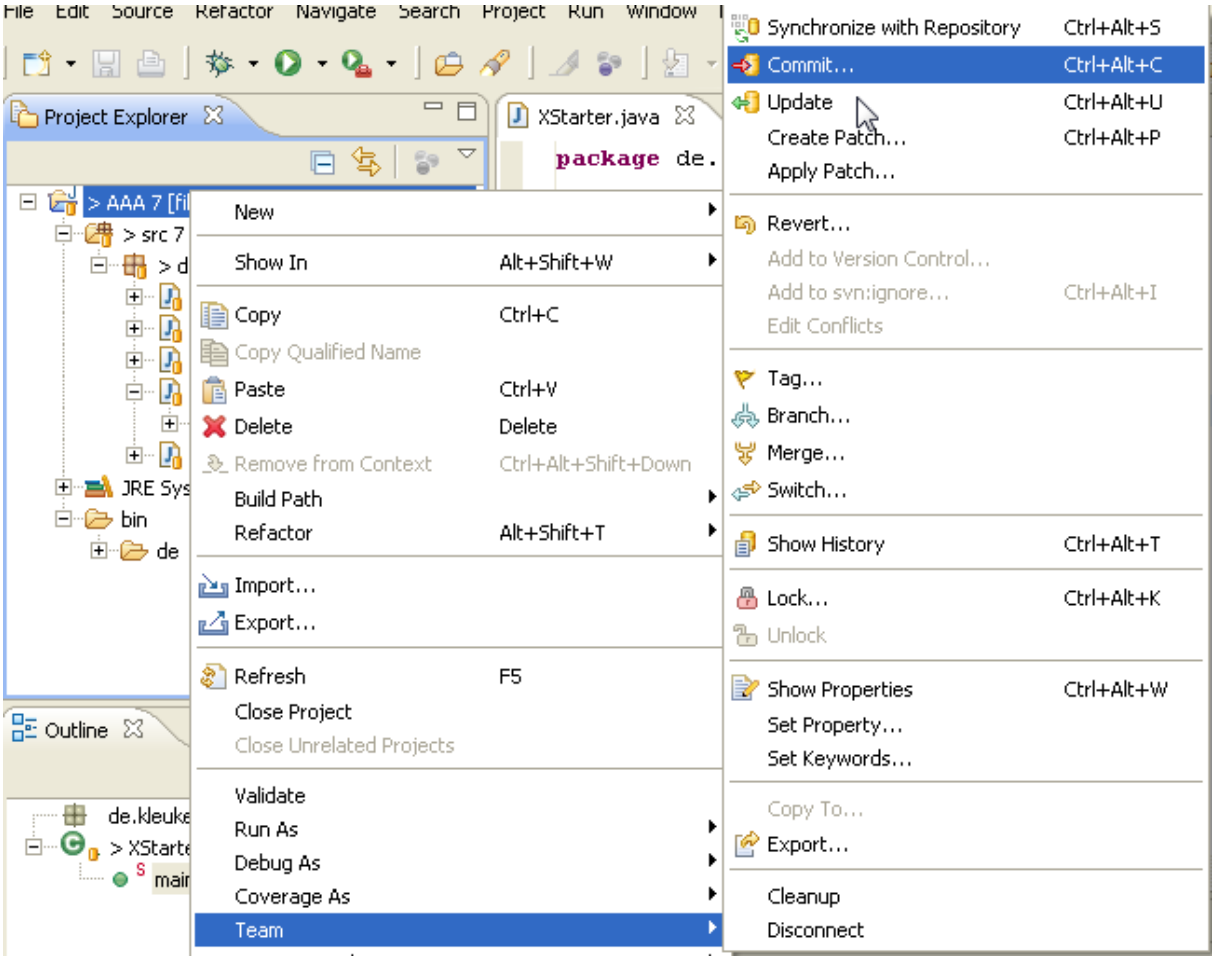
Man erkennt z. B., dass das gesamte bin-Verzeichnis nicht eingecheckt wird. Dies ist natürlich sinnvoll, da nicht jedes kompilierte Ergebnis eingecheckt werden soll. Grundsätzlich kann man von hier aus auch dieses Verzeichnis unter Versionskontrolle stellen. Dies wäre aber nur für ein leeres Verzeichnis sinnvoll, damit dieses beim Auschecken mit einem anderen Werkzeug garantiert angelegt wird.



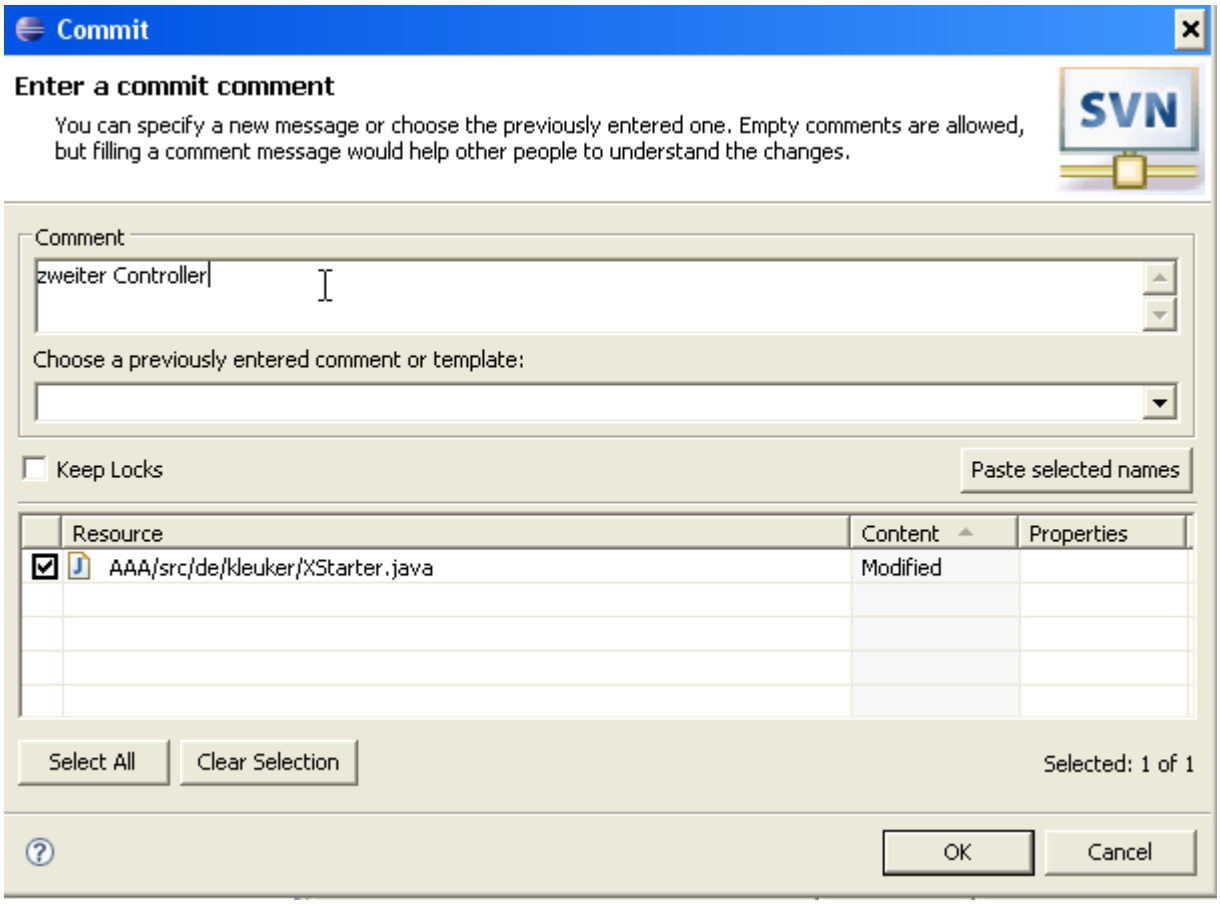
Verändert man jetzt eine Datei, so sieht man die Änderung gegenüber der Datei unter Versionskontrolle an einem „>“.



Danach kann der Nutzer z. B. die Änderung einfach einchecken (Team > Commit..).

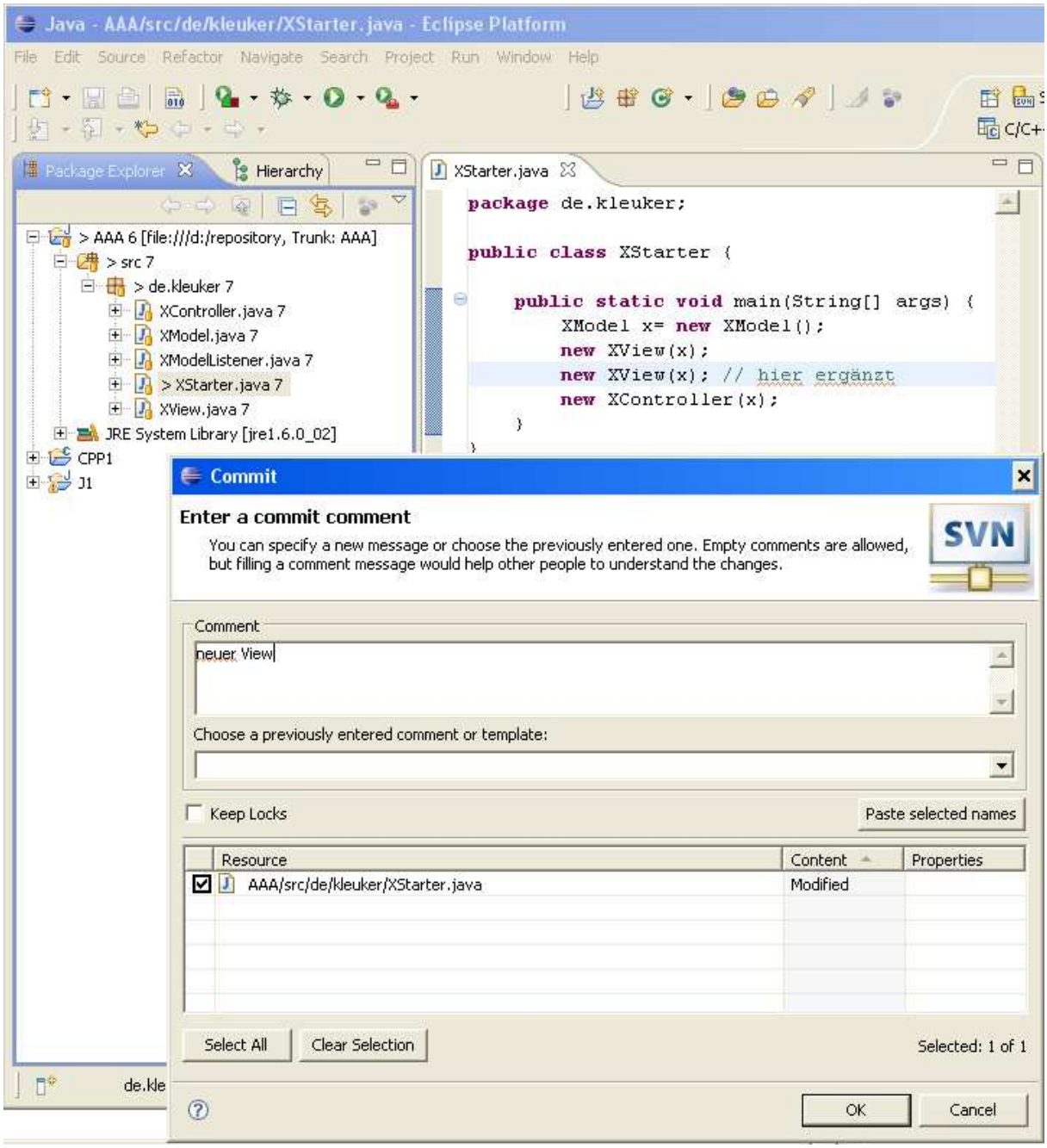


Man muss dann einen sinnvollen Kommentar zur durchgeführten Änderung eingeben. Hierzu sollte es eine Schablone geben, z. B. wer, wann, warum, auch wenn Teile dieser Informationen aus dem SVN-Repository direkt herausgelesen werden können.

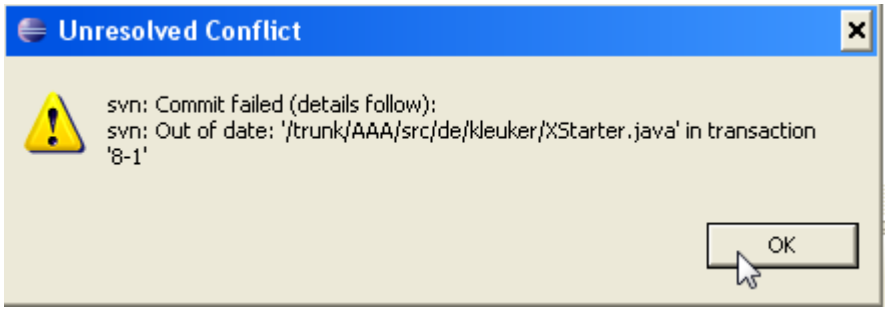


Gegebenenfalls muss man seine Nutzerdaten noch bestätigen.

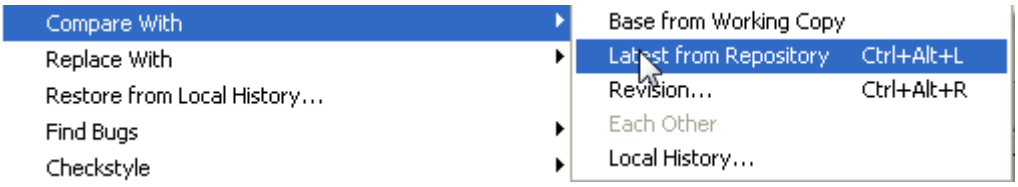
Falls ein anderer Nutzer vor einem Änderungen gemacht hat, sieht der Commit-Prozess zunächst genauso aus.



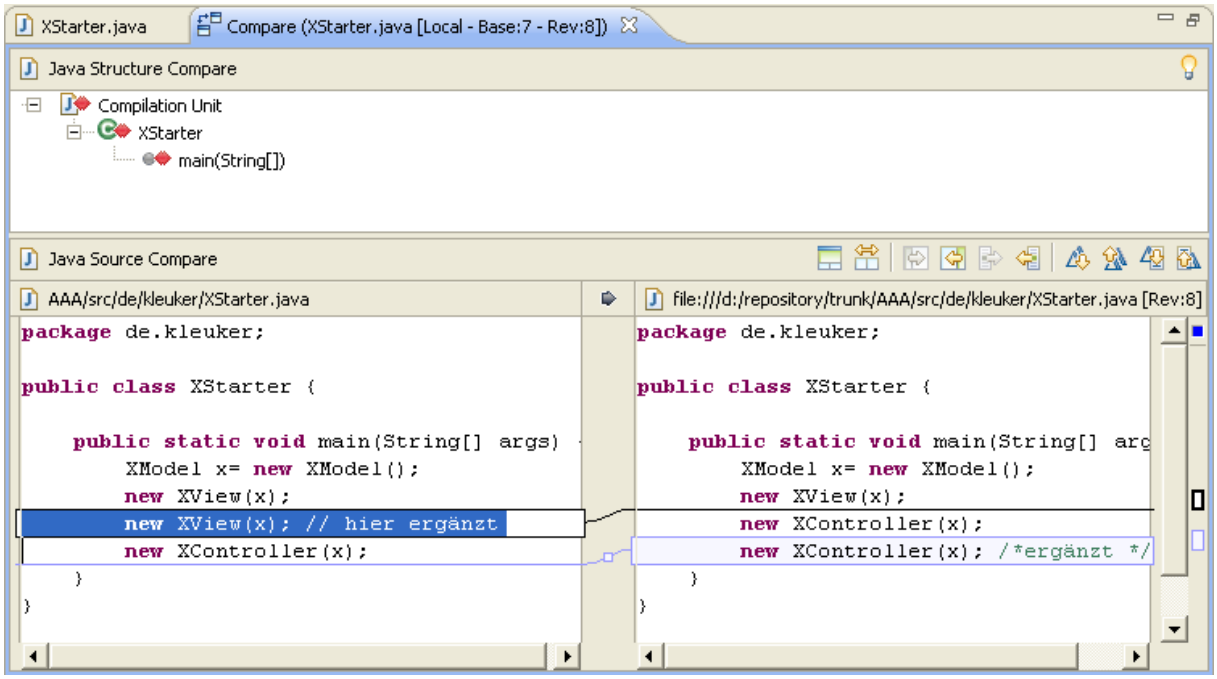
Danach wird man aber auf das Problem hingewiesen.



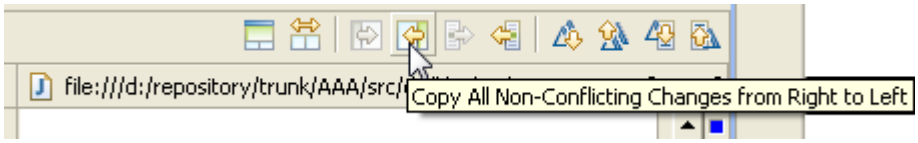
Der Entwickler hat jetzt die bekannten Möglichkeiten, z. B. den Vergleich mit der momentan gesicherten Version.



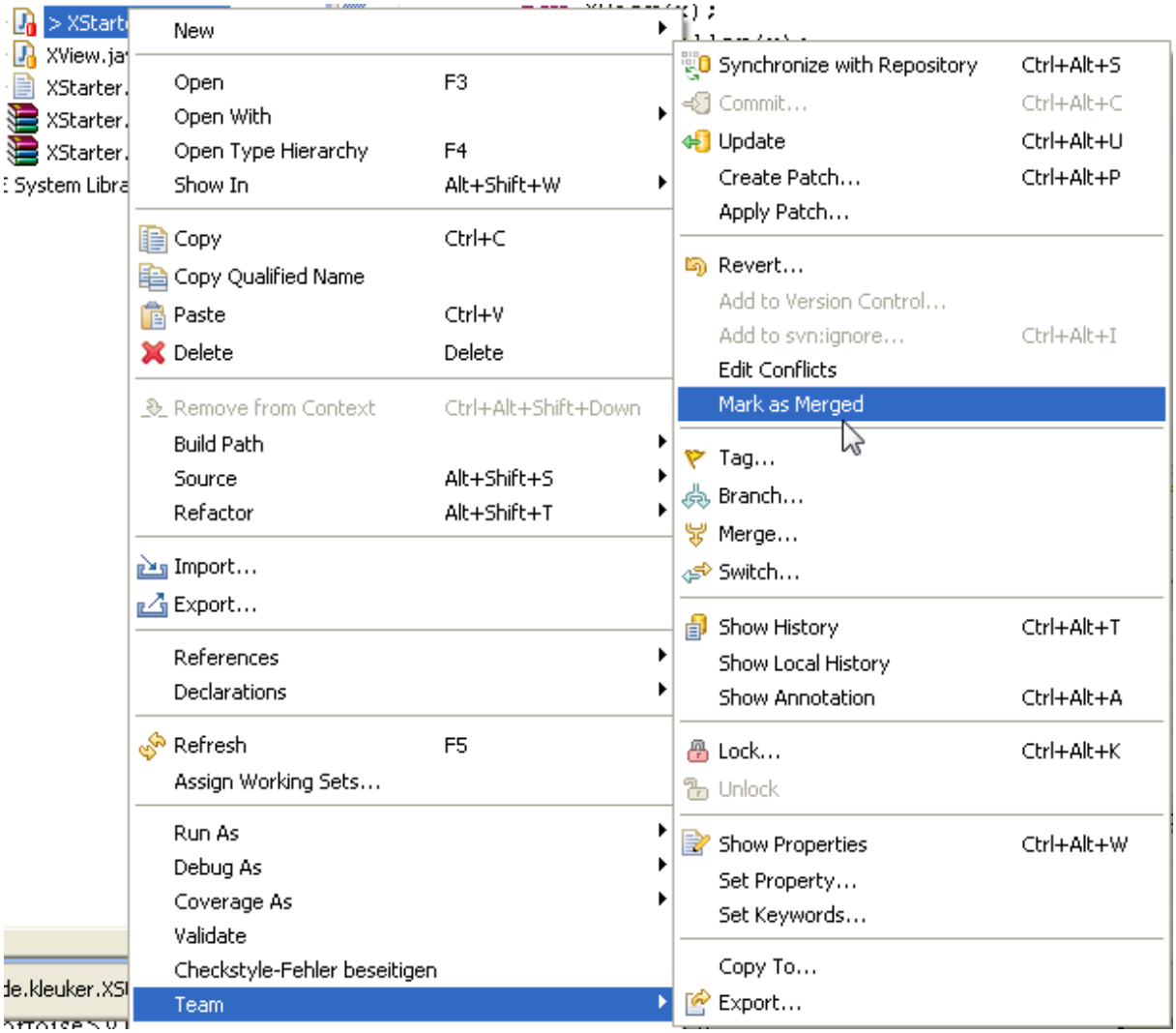
Man erhält eine Übersicht über die vorgenommenen Änderungen seit der genutzten Basisversion.



Dieser Editor bietet einige Arbeitsmöglichkeiten, die individuell genauer betrachtet werden sollten. Hilfreich ist z. B. folgender Punkt.



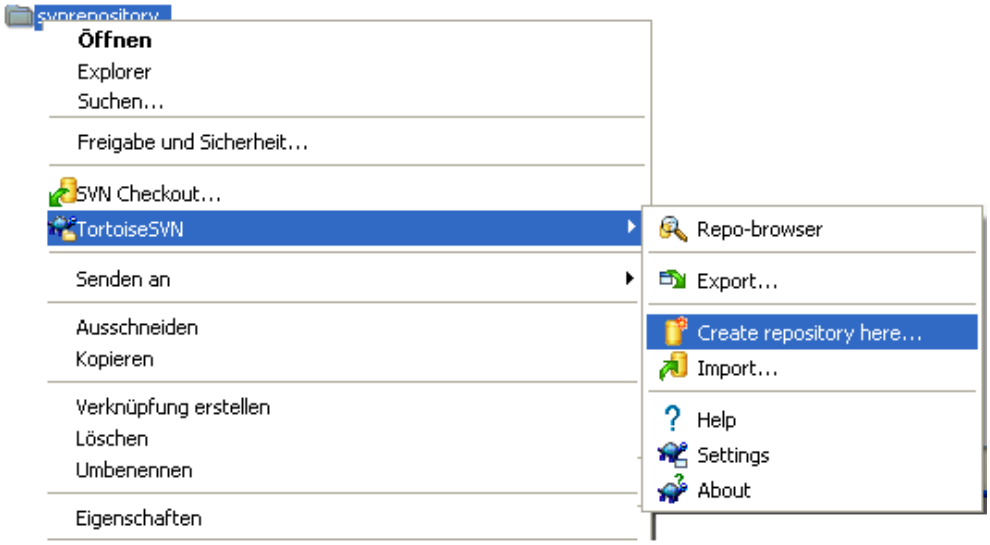
Weiterhin kann die Update-Funktionalität von SVN genutzt werden. Typischerweise steht erst nach einem Update die Funktionalität „Mark as merged“ zur Verfügung, wonach ein Commit problemlos möglich ist, wenn nicht zwischenzeitlich erneut ein Anderer schneller war. Im folgenden Bild sieht man weiterhin die rote Markierung, die anzeigt, dass ein Update einen Konflikt bemerkt hat.



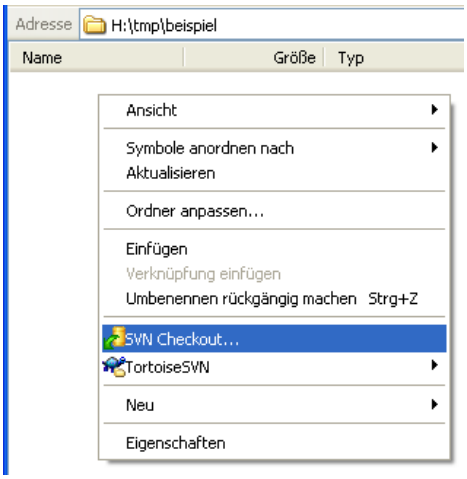
17.6 Nutzung von TortoiseSVN

Wer nicht gerne auf der Kommandozeile arbeitet, kann die Subversion-Erweiterung TortoiseSVN von der gleichen Web-Seite wie Subversion nutzen (hier TortoiseSVN-1.4.3.8645-win32-svn-1.4.3.msi). Dieses Programm integriert die Fähigkeiten von Subversion direkt in den Browser und kann sehr hilfreich beim Umgang mit Dateien sein, die nicht durch Eclipse, genauer Subversive, verwaltet werden. Da jeder nur mit Kopien aus dem Repository arbeitet, ist es grundsätzlich egal, ob die Steuerungen der Versionskontrolle aus Eclipse heraus oder mit TortoiseSVN oder direkt über die Kommandozeile erfolgen.

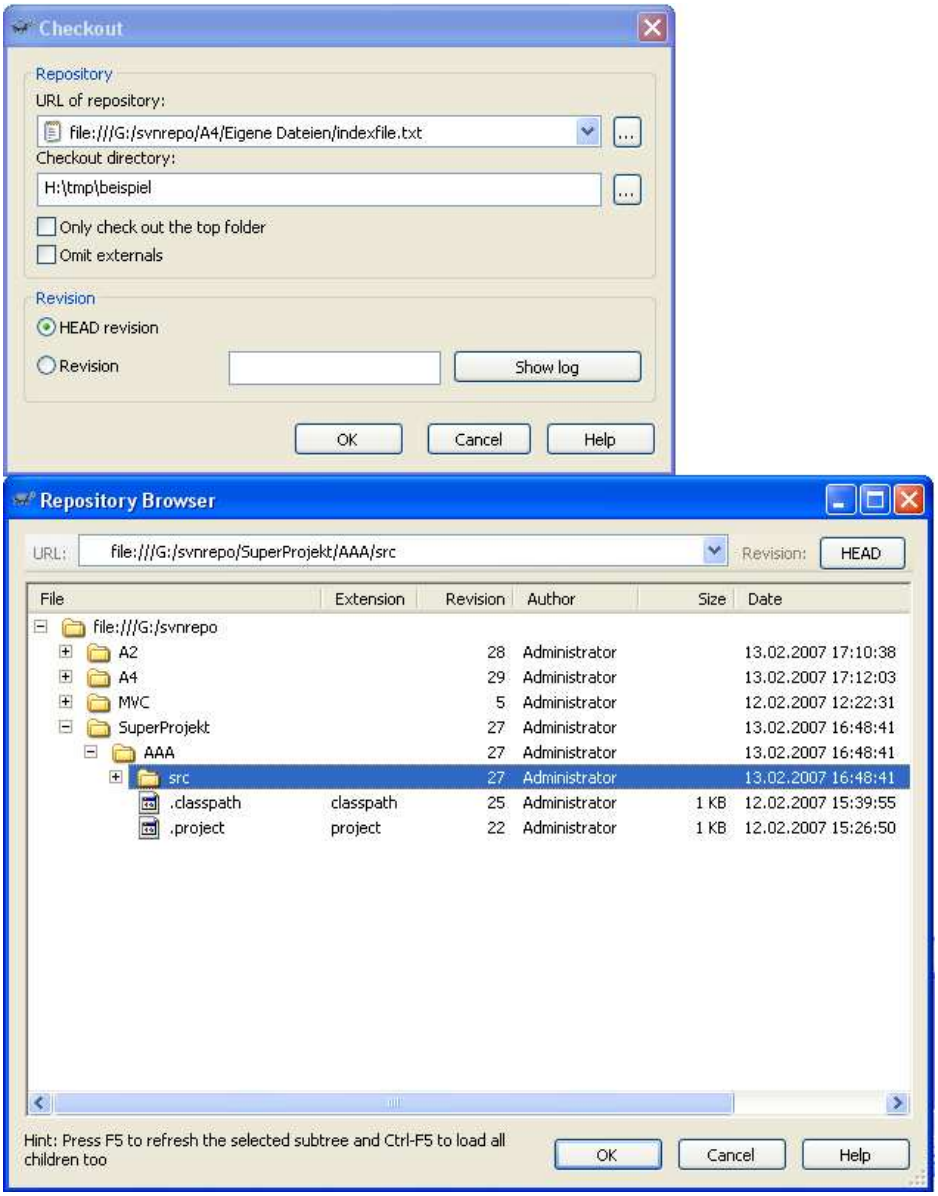
Unter anderem kann man mit Tortoise auch Repositories anlegen. Dazu wird ein leeres Verzeichnis erstellt und dann ein Rechtklick auf diesem Verzeichnis durchgeführt. Unter „TortoiseSVN > Create Repository here...“



Weiterhin ist interessant, dass man wie im vorherigen Bild gezeigt auch Dateien auschecken kann, was hier in einem Beispielordner H:/tmp/beispiel passiert. Bei einem Rechtsklick in einem Ordner kann der Punkt „SVN Checkout ...“ gewählt werden.

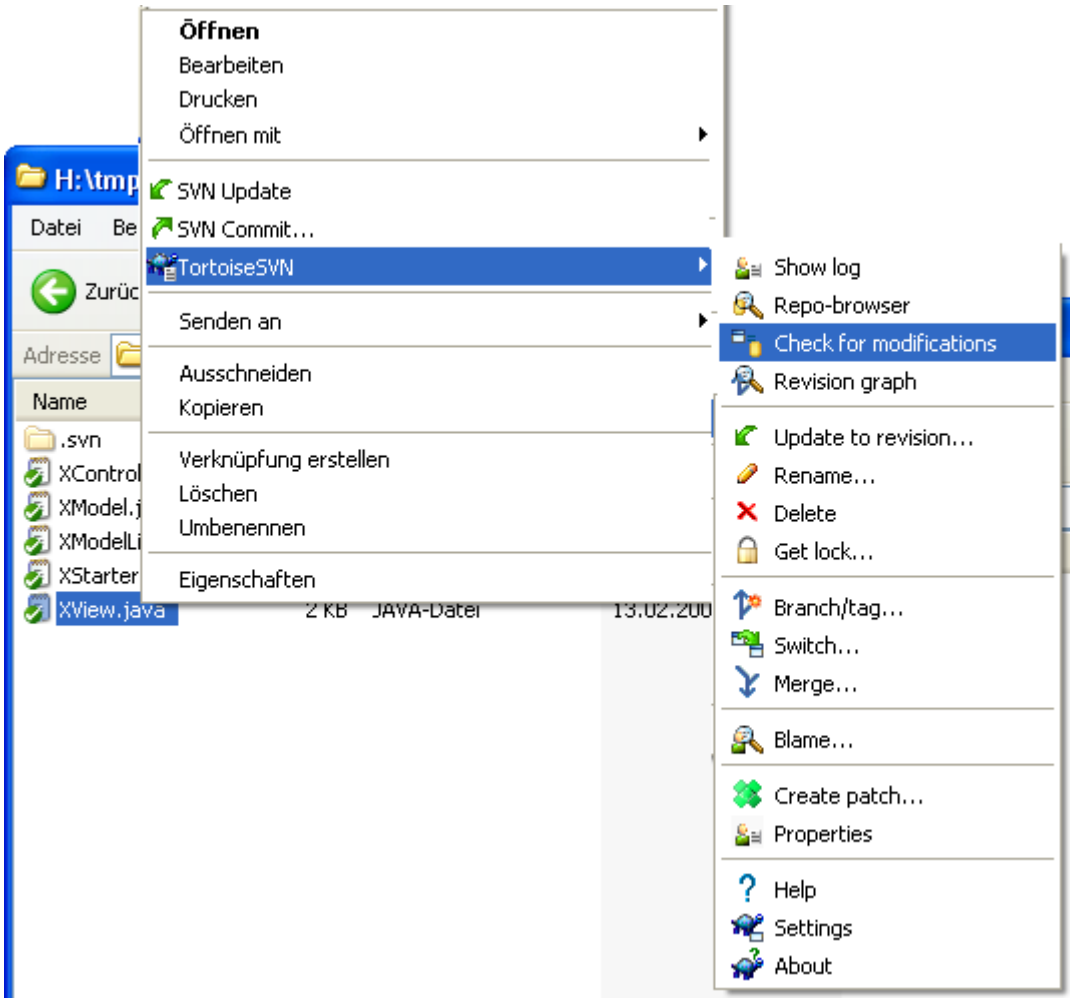


Danach öffnet sich ein Browser, in dem man zunächst den Pfad zu einem Repository angeben muss. Nachdem das Repository erkannt wurde (Eingabeauswahl mit dem oberen Knopf „...“), kann dieses durchsucht und der gewünschte Ordner zum Auschecken ausgewählt werden.

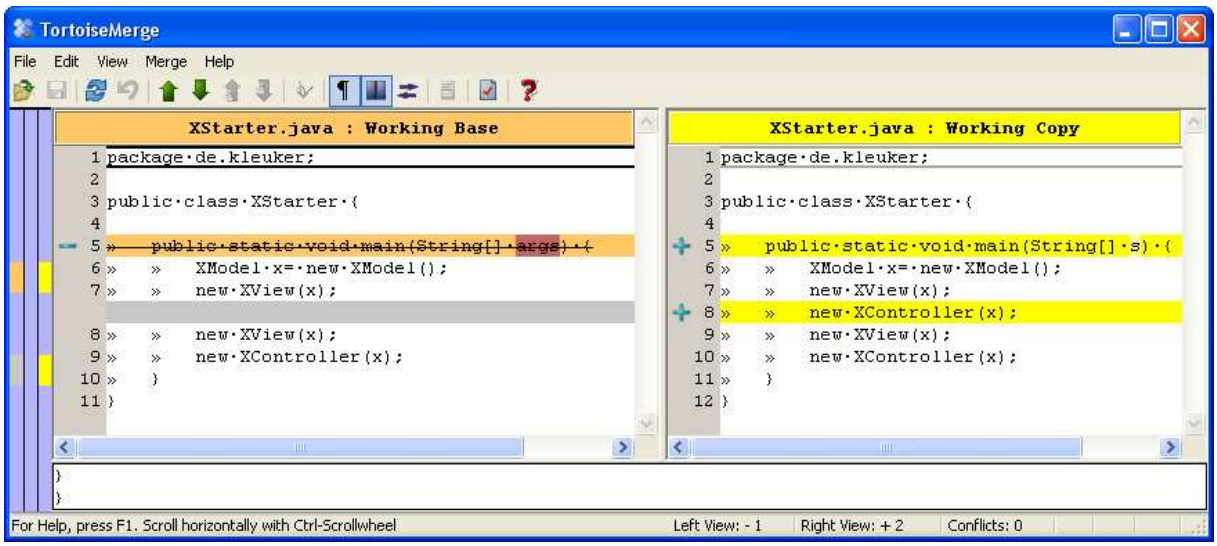


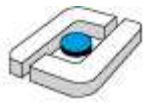
Danach erkennt man an den Icons der Dateien, dass sie unter Versionskontrolle stehen.

Durch einen Rechtsklick auf eine solche Datei stehen dann die von svn bekannten Funktionalitäten zur Verfügung.



Interessant ist weiterhin der in TortoiseSVN realisierte Browser für Differenzen. Das folgende Bild zeigt ein Beispiel mit der ursprünglichen Datei im Repository links und der geänderten Datei rechts.





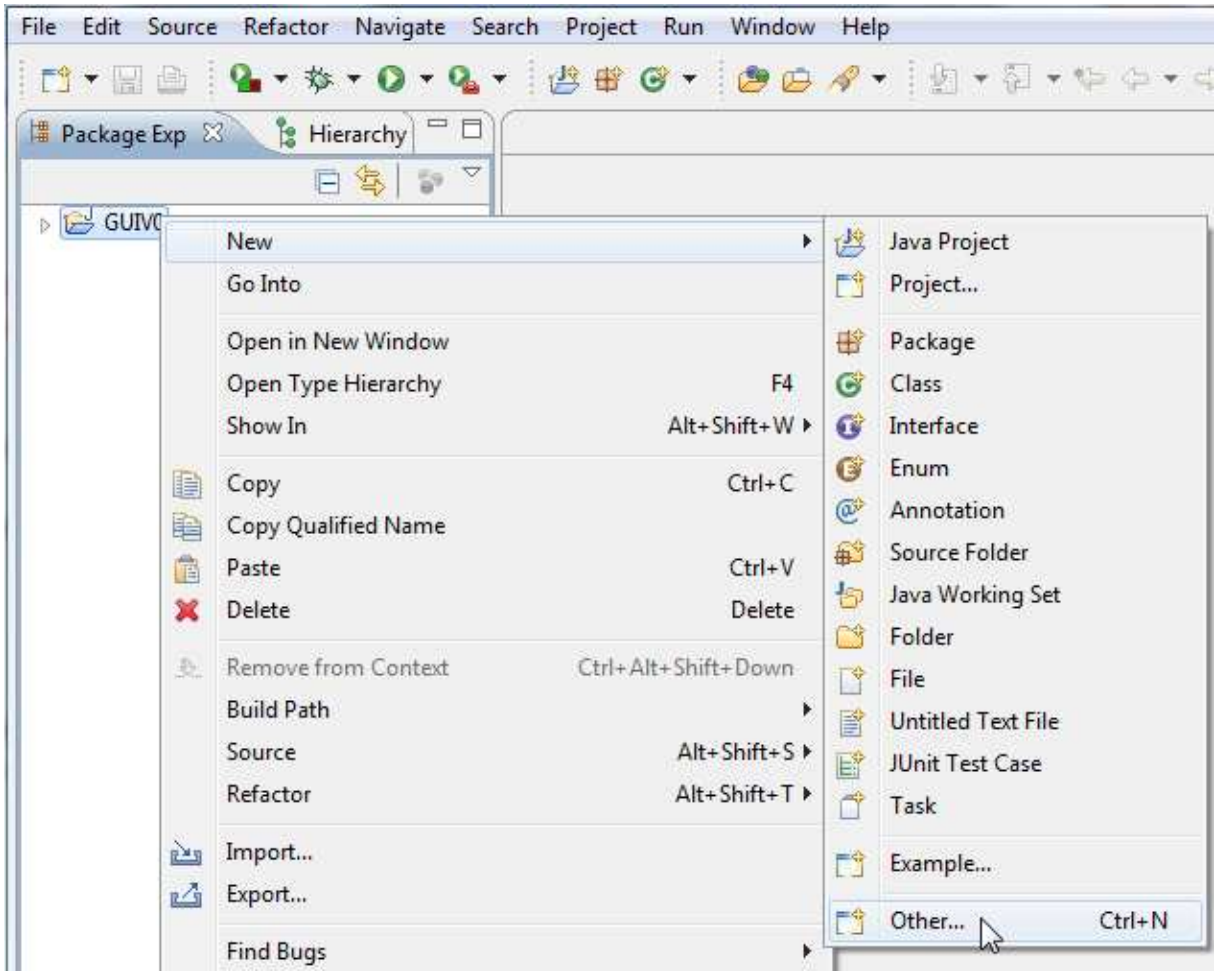
18 Nutzung eines GUI-Builders

GUI-Builder erlauben die schnelle Erstellung von graphischen Oberflächen, zumindest von Prototypen die mit Kunden diskutiert werden können. Generell gilt für alle GUI-Builder, dass man vor ihrer Nutzung verstanden haben muss, wie ein GUI sich zusammensetzt, also z. B. welche Konzepte hinter Swing mit seinen LayoutManagern und dem „Schächtelchen-in-Schächtelchen“-Prinzip der GUI-Entwicklung stehen.

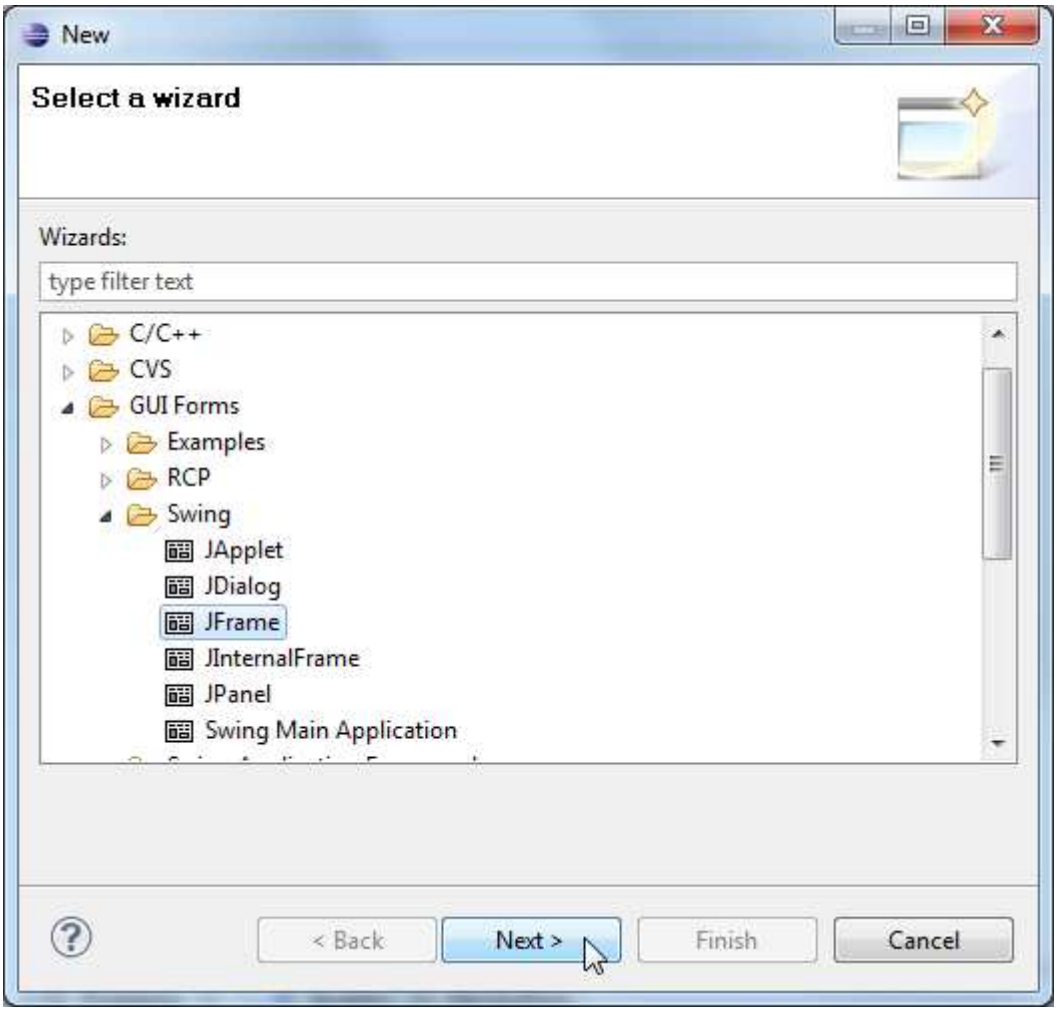
Beim Einsatz von GUI-Buildern ist immer zu beachten, dass man sich an die Kodierungsregeln des GUI-Builders halten muss, da nur dann eine kontinuierliche Weiterentwicklung eines GUIs ermöglicht wird. Im schlimmsten Fall kann ein GUI nicht mehr vom GUI-Builder bearbeitet werden, da man das Programm etwas umgestellt hat und weiterhin aber den Gesamtaufbau des Programms nicht versteht.

In diesem Kapitel wird nur ein erster Einblick in die freie Variante des GUI-Builders Jigloo (http://www.cloudgarden.com/jigloo/) gegeben. Entwickler sind aufgefordert, selbst mit dem Programm zu experimentieren und sich verschiedene Varianten von Änderungen zu überlegen, um dann heraus zu bekommen, wie diese Änderungen mit dem GUI-Builder umgesetzt werden können. Die Erklärungen des Werkzeugherstellers sind empfehlenswert.

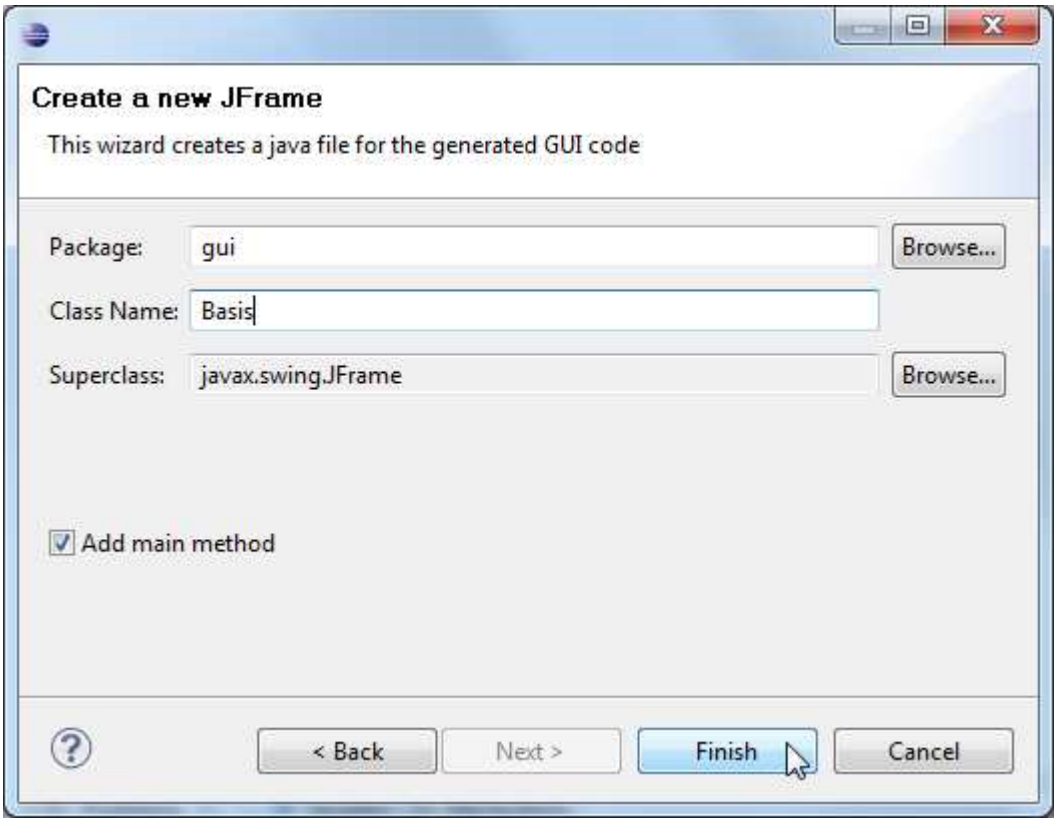
Um eine GUI in einem beliebigen Java-Projekt, hier „GUIV0“ zu bauen, wird über einen Rechtsklick auf dem Projekt dann „New->Other“ ausgewählt.



Hier wird dann „GUI Forms“ und dann „Swing“ aufgeklappt. Jetzt muss man sich für die Hauptart des Fensters entscheiden, die die Grundlage des GUIs sein soll, im Beispiel „JFrame“.

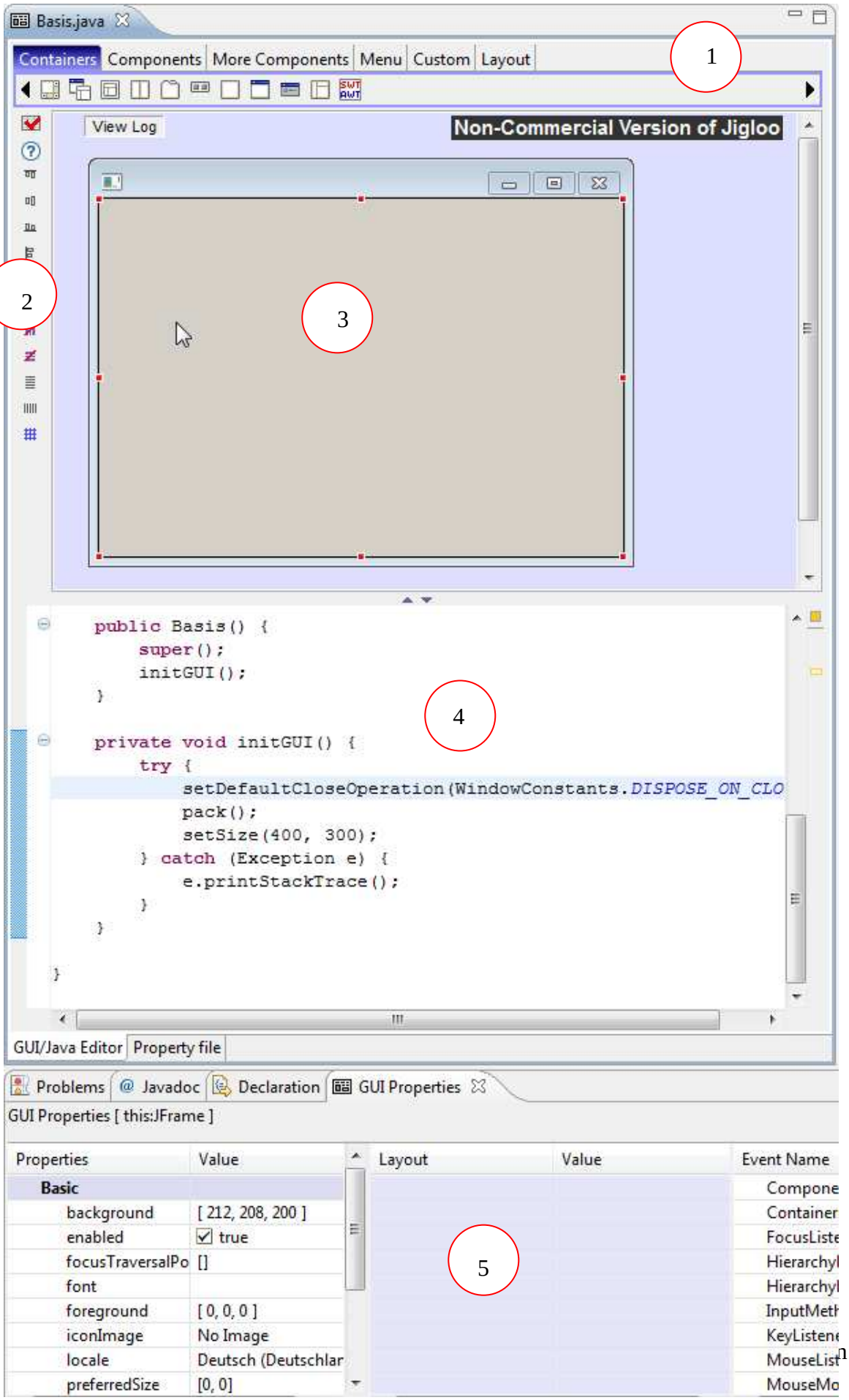


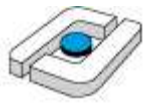
Nun muss ein Klassenname (Class Name) und kann ein Paket (Package) angegeben werden. Die Klasse wird dann im Paket nach Drücken von „Finish“ angelegt.



Zwischenzeitlich gibt es einen Hinweis, dass die Jigloo-Variante nur für nicht-kommerzielle Projekte genutzt werden darf, die bestätigt wird.

Im Wesentlichen steht dann eine Entwicklungsumgebung zur Verfügung, die aus folgenden Teilen besteht:


Basis.java
Containers
Components
More Components
Menu
Custom
Layout
1
View Log
Non-Commercial Version of Jigloo
2
3
4
public Basis() {
super();
initGUI();
}
private void initGUI() {
try {
setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLO
pack();
setSize(400, 300);
} catch (Exception e) {
e.printStackTrace();
}
}
}
GUI/Java Editor
Property file
Problems
Javadoc
Declaration
GUI Properties
GUI Properties [this:JFrame]
Properties
Value
Layout
Value
Event Name
Basic
background
[212, 208, 200]
enabled
true
focusTraversalPo
[]
font
foreground
[0, 0, 0]
iconImage
No Image
locale
Deutsch (Deutschlar
preferredSize
[0, 0]
5



1: verschiedene Reiter, mit deren Hilfe GUI-Elemente ausgewählt werden können, die dann in die Oberfläche eingefügt werden können

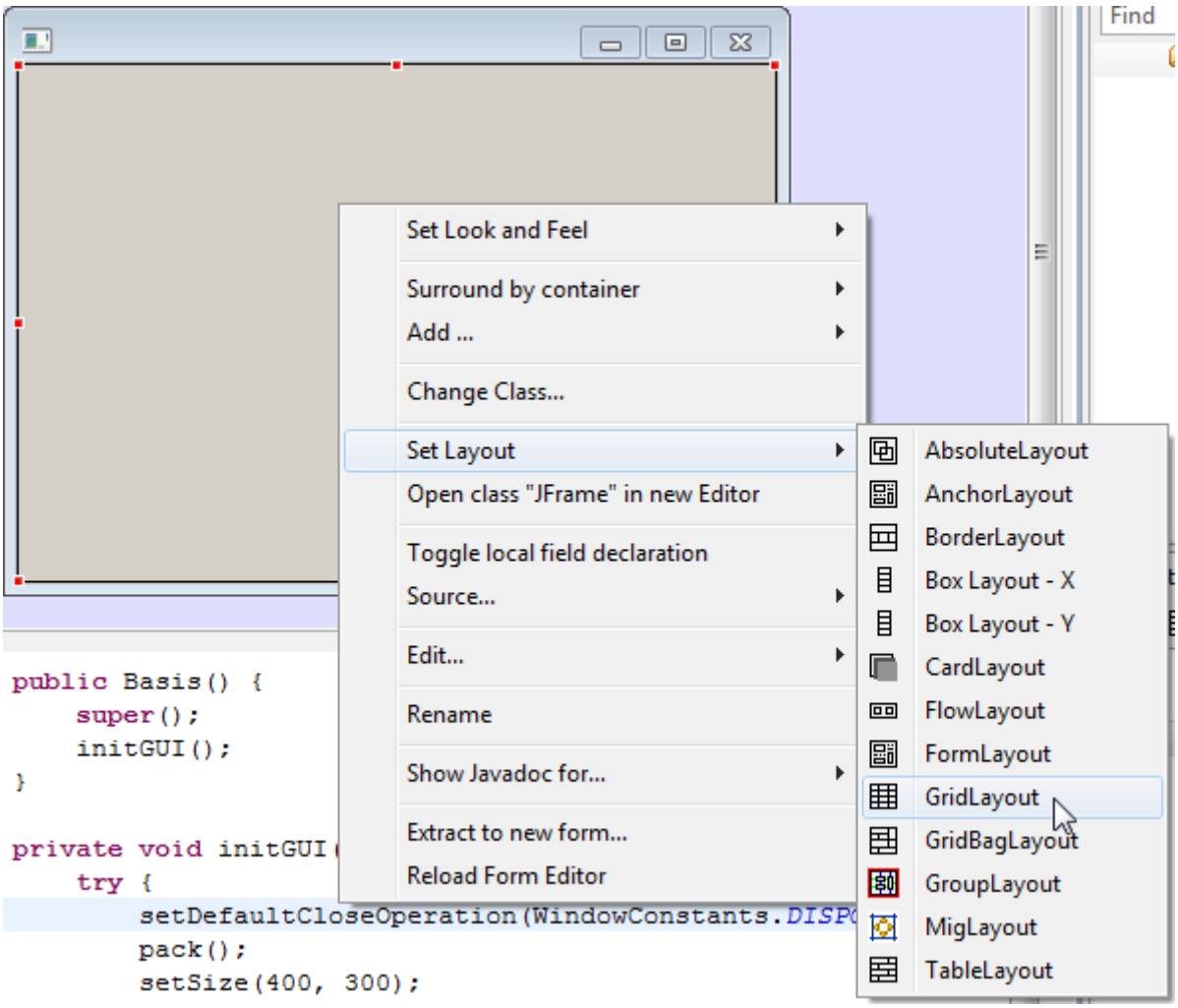
2: Detaileinstellungen, z. B. zur Ausrichtung mehrerer GUI-Elemente an einer Linie

3: das aktuell zu bearbeitende GUI, wobei das zurzeit genauer betrachtete Element rot umrahmt ist

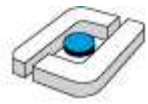
4: der zum ausgewählten Element gehörige Programm-Code

5: Reiter GUI-Properties, mit dem viele Detaileinstellungen für das ausgewählte Element vorgenommen werden können

Durch einen Rechtsklick auf das gewählte GUI-Element werden weitere Einstellungsmöglichkeiten, wie das Layout angezeigt.

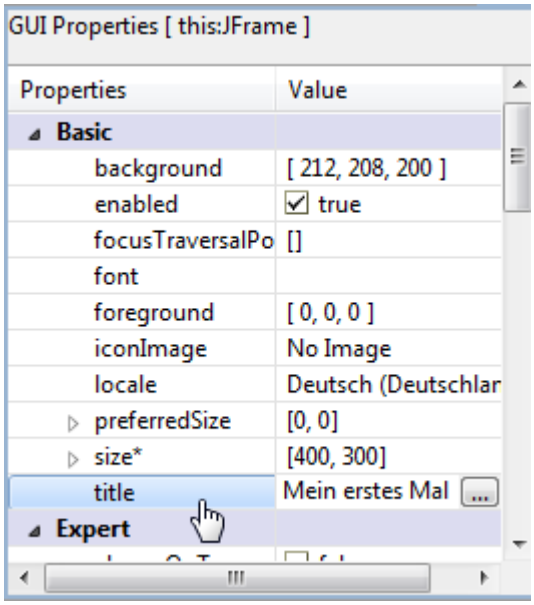


Detailanpassungen nimmt man typischerweise im Programm-Code vor.

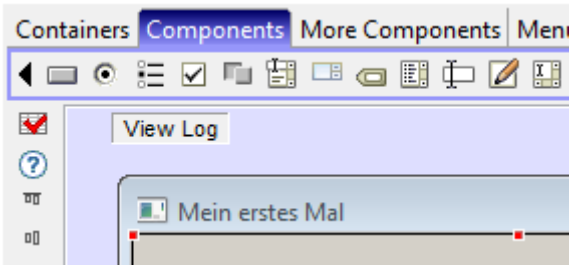


```
try {
    GridLayout thisLayout = new GridLayout(1, 1);
    thisLayout.setHgap(5);
    thisLayout.setVgap(5);
    thisLayout.setColumns(2);
    getContentPane().setLayout(thisLayout);
```

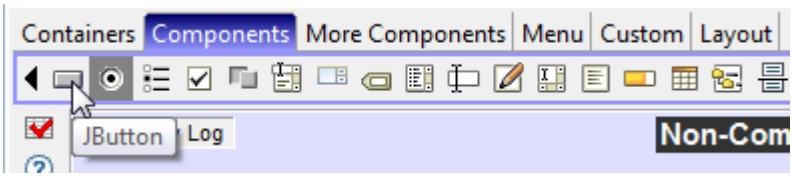
Den Titel des Fensters kann man z. B. über die GUI Properties einstellen, alternativ sind viele Textänderungen auch durch einen Doppelklick auf das zu ändernde Element bearbeitbar.



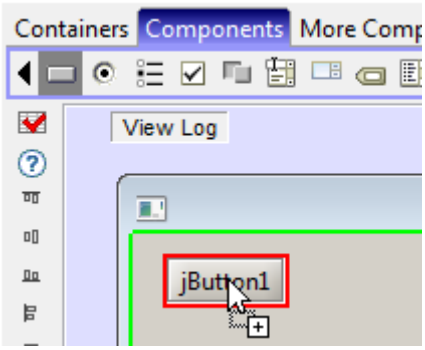
Der Titel wird auch unmittelbar angezeigt.



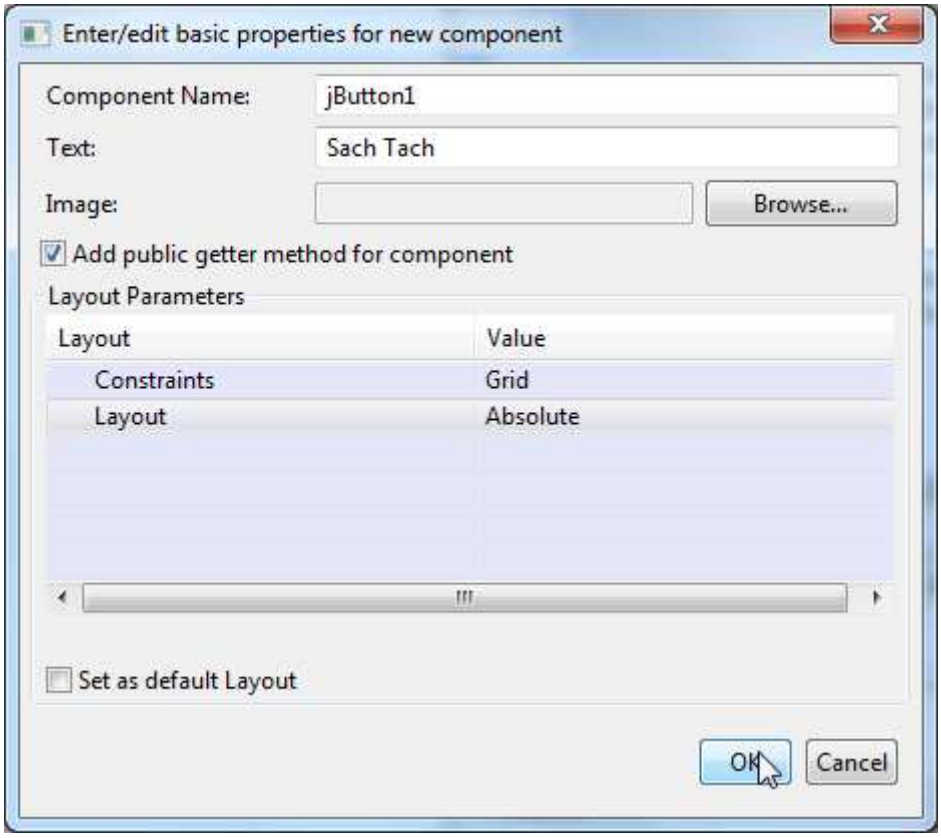
Möchte man z. B. einen Knopf zur Oberfläche hinzufügen, so wird zunächst im oberen Bereich unter „Components“ der zugehörige JButton gesucht.



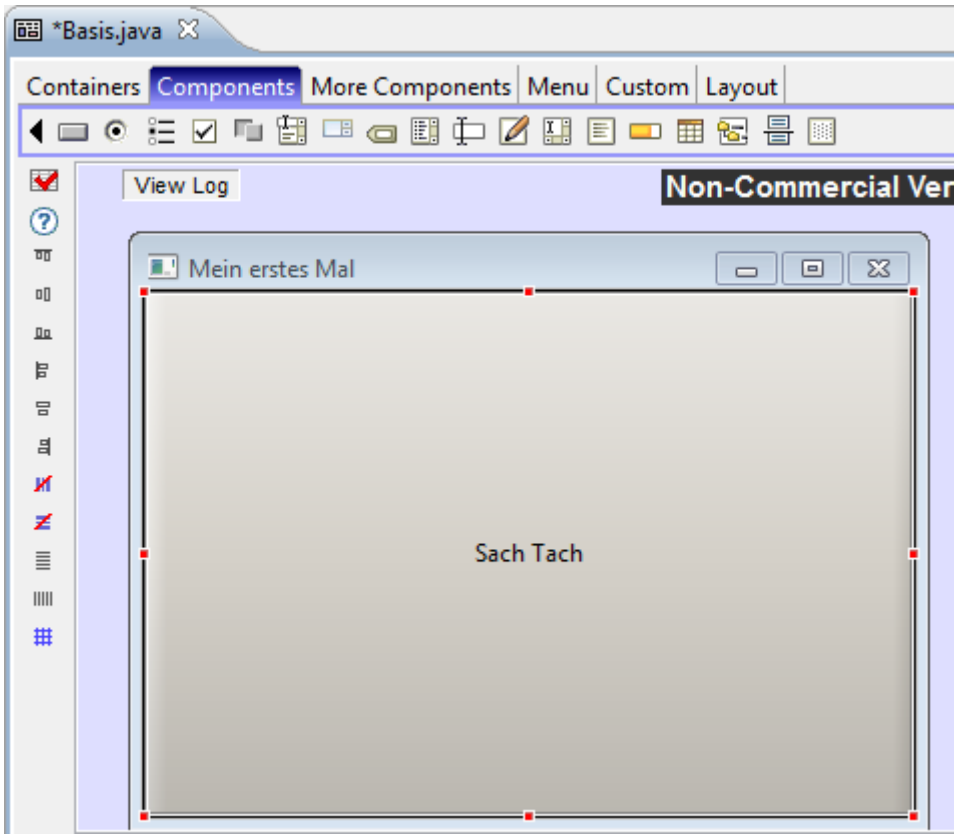
Wird dann dieses Element angeklickt, kann man es durch Schieben der Maus in der Oberfläche und erneutem Klicken platzieren.



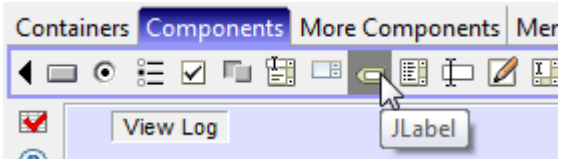
In dem folgenden Fenster können einige Eigenschaften des Knopfes festgelegt werden, weitere Eigenschaften und Änderungen können später jederzeit als Properties eintragbar. Der „Component Name“ wird intern benutzt und sollte auch „sprechend“ sein. Die eigentliche Knopfbeschriftung steht im Text.



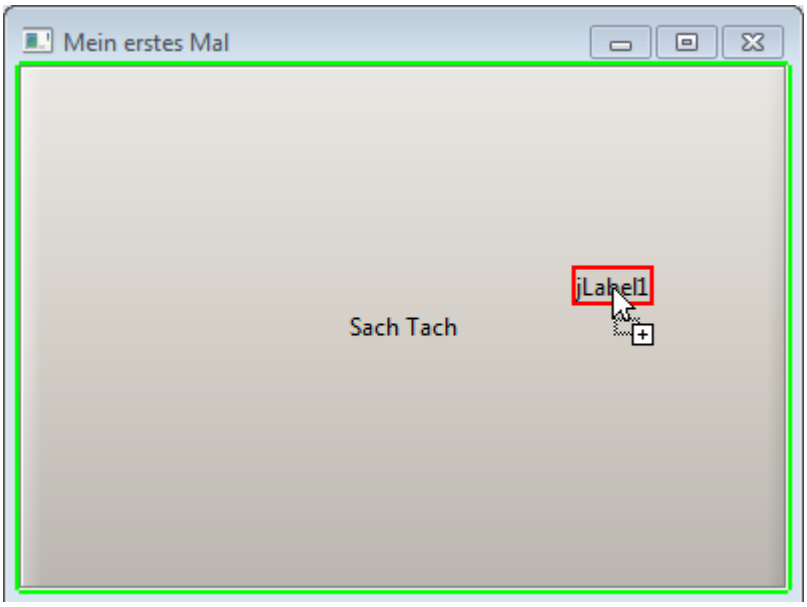
Der Knopf nimmt zunächst die gesamte Oberfläche ein, da keine weiteren Komponenten existieren.

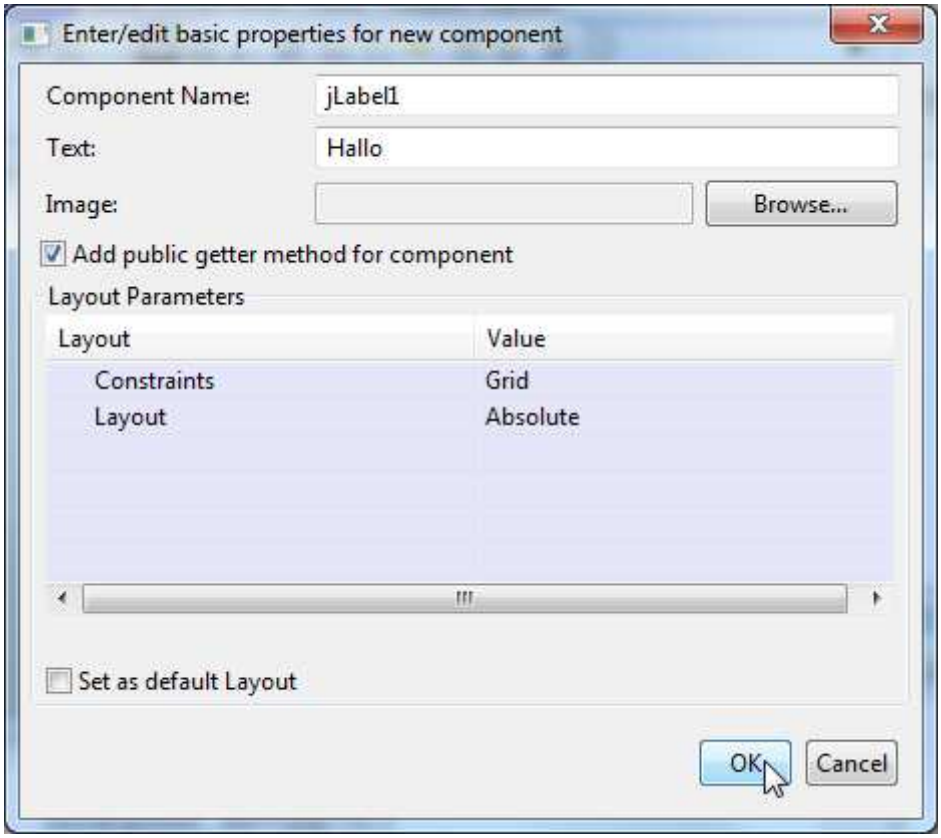


Als nächstes soll ein JLabel platziert werden, was analog zum JButton verläuft.

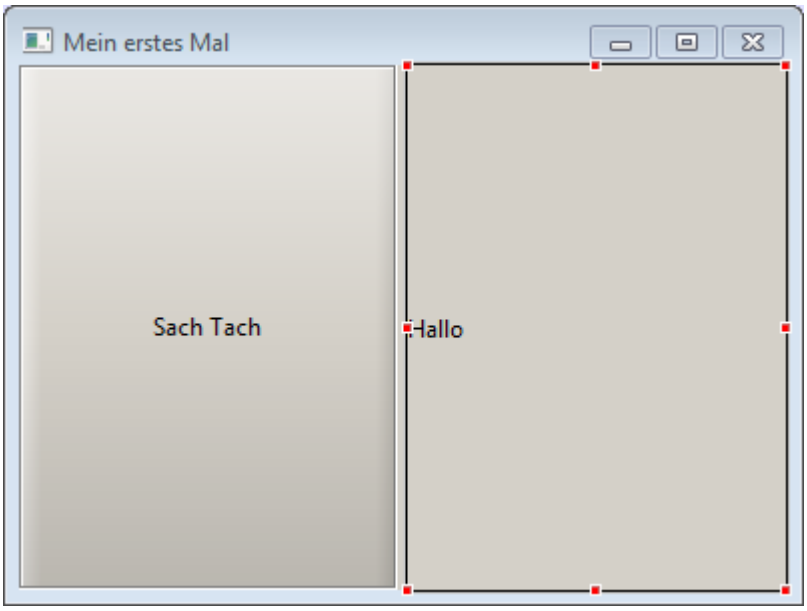


Beim platzieren können Elemente auch zunächst übereinander platziert werden. Die genaue Ordnung wird nach dem Einfügen unter Berücksichtigung des Layouts hergestellt.

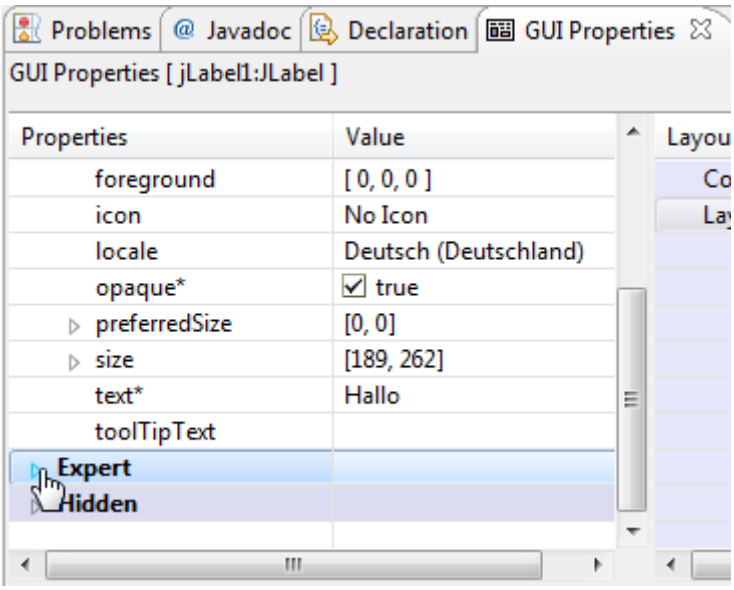




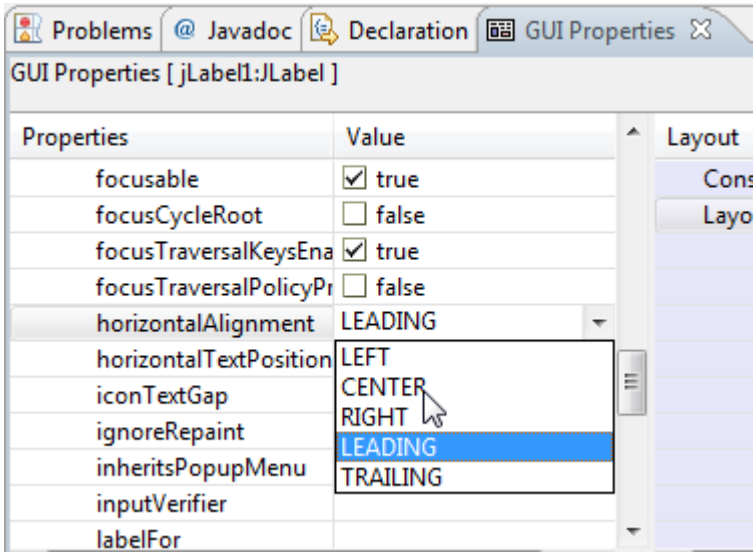
Das Zwischenergebnis sieht wie folgt aus.



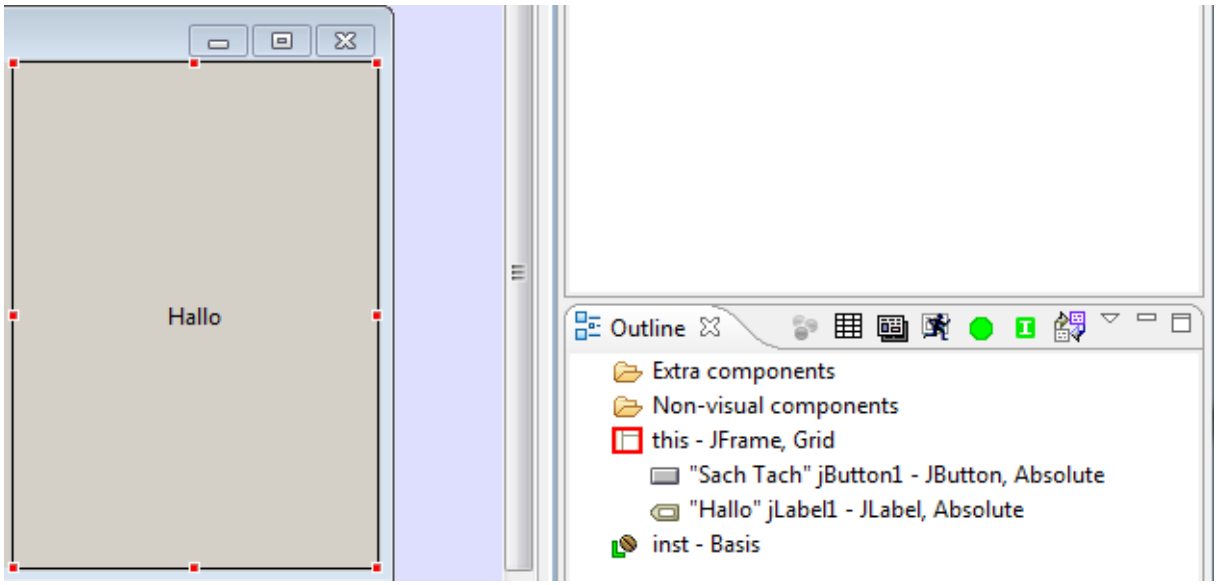
Um die unbefriedigende Platzierung des Textes zu ändern, muss man unten im Reiter „GUI Properties“ die Einstellungen unter „Expert“ aufklappen.



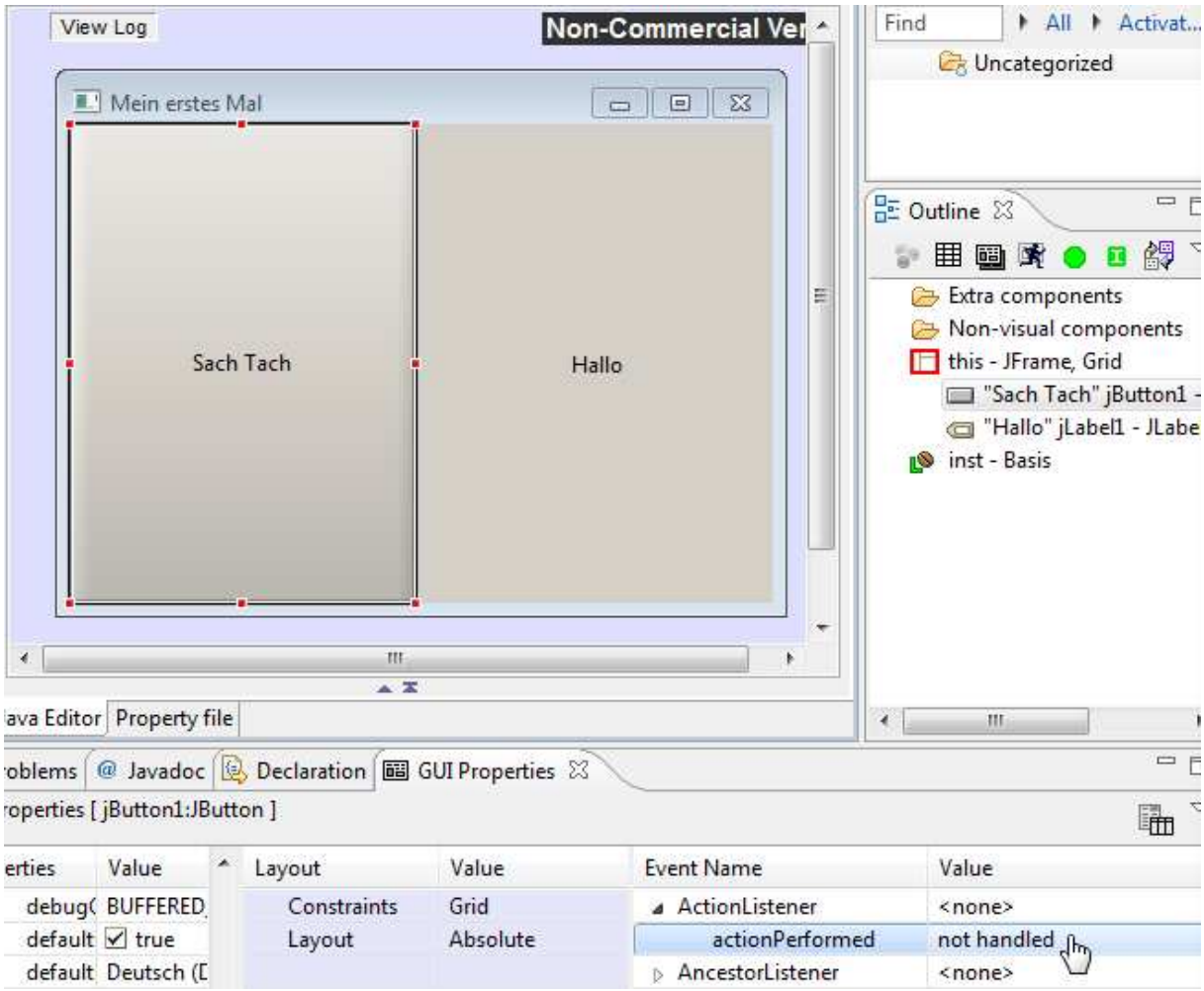
Hier kann jetzt die Textplatzierung geändert werden. Dazu klickt man in das zu ändernde Feld und entweder klappt ein Auswahlmenü auf, oder man kann direkt Werte eintragen. Im konkreten Beispiel wird der Wert „CENTER“ ausgewählt.



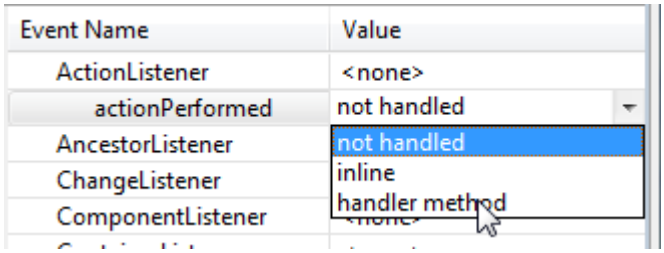
Generell ist zu beachten, dass der Aufbau des gesamten GUIs auch in dem Reiter „Outline“ sichtbar ist, der typischerweise am Anfang rechts neben dem Editorfenster platziert wurde. Durch das Klicken auf ein Element in „Outline“ wird dieses im Editor selektiert.



Nun soll der Knopf mit dem Label verknüpft werden. Dazu soll der Knopf ein ActionListener-Objekt erhalten. Dazu wird der Knopf oben im Erstellungsbereich ausgewählt und rechts-unten das Feld „ActionListener“ ausgeklappt. Hier steht, wie im folgenden Bild gezeigt, ursprünglich der Wert „not handled“.

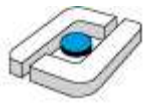


Klickt man in dieses Feld, hat man die Möglichkeit den Programm-Code entweder direkt als anonymes Objekt bei der Objekterstellung zu ergänzen (inline), oder die Bearbeitung in einer eigenen Methode durchzuführen (handler method). Nicht gezeigt ist die Möglichkeit, ein eigenes Objekt einer anzugebenden Klasse zu nutzen.



Bei der zweiten Variante wird bei der Knopf-Erstellung folgendes Programmstück ergänzt.

```
    {
        jButton1 = new JButton();
        getContentPane().add(getJButton1());
        jButton1.setText("Sach Tach");
        jButton1.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent evt) {
```



```
            jButton1ActionPerformed(evt);
          }
        });

    }
```

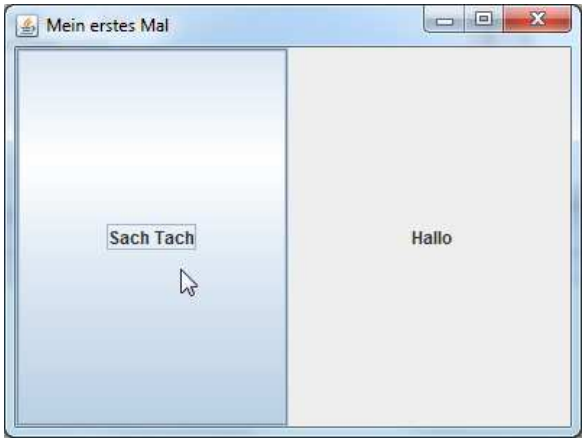
Weiterhin wird der Rumpf einer Methode angelegt, die der Entwickler dann ausprogrammieren muss. Bei einer Entwicklung muss man sich nur um diese Methode kümmern. Man spürt aber bereits bei diesem kleinen Beispiel, dass Änderungen das Programm schnell schwer pflegbar machen.

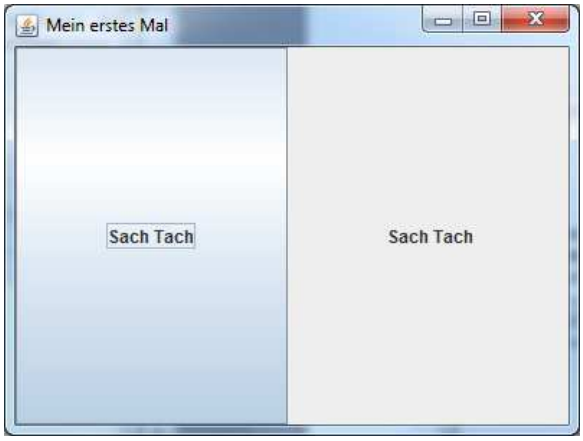
```
  private void jButton1ActionPerformed(ActionEvent evt) {
    System.out.println("jButton1.actionPerformed, event="+evt);
    //TODO add your code for jButton1.actionPerformed
  }
```

Die Methode wird wie folgt ergänzt.

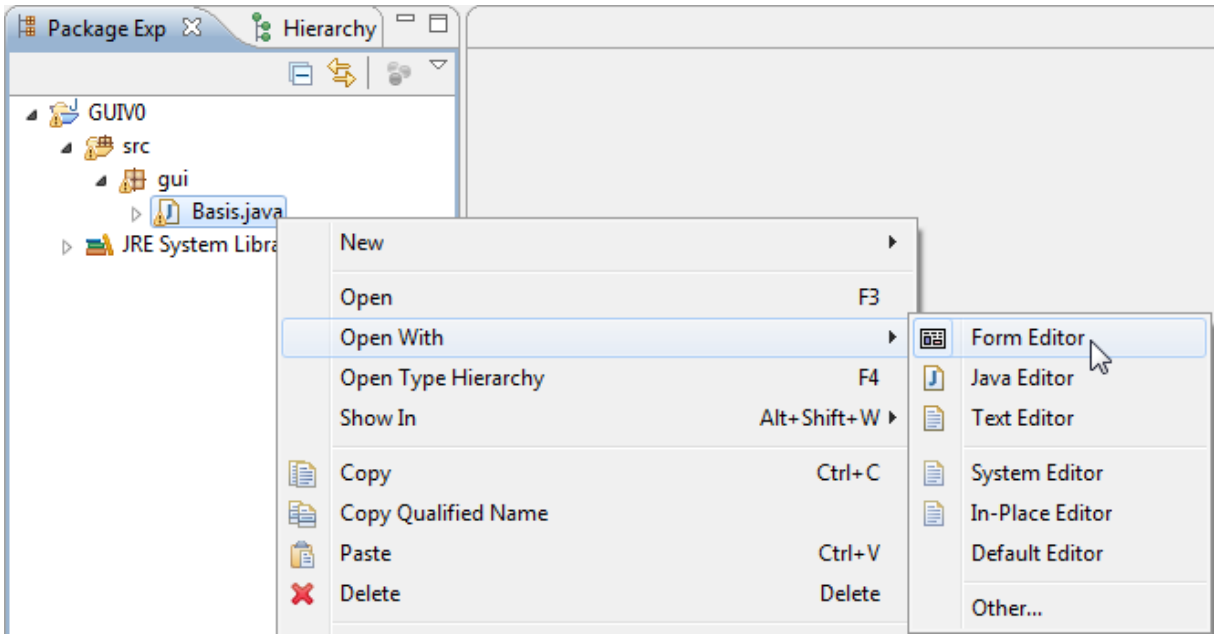
```
  private void jButton1ActionPerformed(ActionEvent evt) {
    getJLabel1().setText(getJButton1().getText());
  }
```

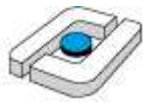
Generell ist es das Angenehme bei Jigloo, dass man das resultierende Programm immer laufen lassen kann. Die folgenden Bilder zeigen links das Programm nach dem Start und rechts nach dem Klicken des Knopfes.





Hat man das Programm beendet und möchte die GUI-Klasse wieder mit Jigloo öffnen, macht man einen Rechtsklick auf der Klasse und wählt „Open With -> Form Editor“.


Package Exp
Hierarchy
GUIV0
src
gui
Basis.java
JRE System Libra
New
Open F3
Open With
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Form Editor
Java Editor
Text Editor
System Editor
In-Place Editor
Default Editor
Other...



19 Ant-Installation

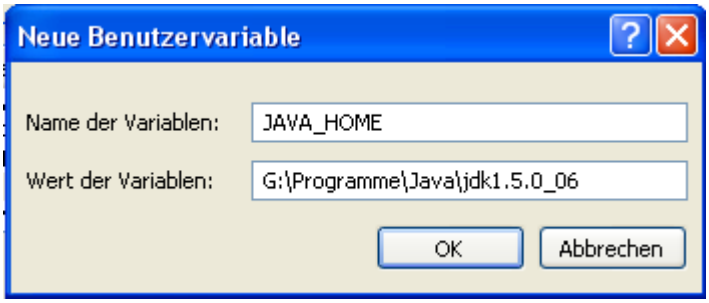
Vor dem Anfang sei u. a. auf die Ant-Dokumentation hingewiesen, die gut geschrieben und viel mehr nützliche Informationen als dieser kleine Einführungstext enthält. Das Verzeichnis ist H:\Programme\ant\docs\manual\index.html, wenn Ant unter H:\Programme ausgepackt wurde. Ant selber ist Teil von Eclipse und muss nur installiert werden, wenn es unabhängig von Eclipse zur Verfügung stehen soll.

Zur Installation wird antSoSe07.zip im Verzeichnis H:\Programme (natürlich auswählbar) ausgepackt. Der Pfad wird in der Systemvariablen ANT_HOME festgehalten.

Dieser Pfad zum bin-Verzeichnis muss in der PATH-Variablen eingetragen werden.

Wenn Sie die CLASSPATH-Variable nutzen, dann darf diese keine Anführungsstriche enthalten!

Weiterhin sollte die Variable JAVA_HOME auf dem aktuell genutzten Java-Verzeichnis stehen.

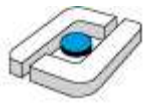


Hilfreich kann der Befehl

```
ant -diagnostics
```

sein, der recht viele Informationen über die genutzte Ant-Installation liefert.

Weiterhin hilfreich kann auch

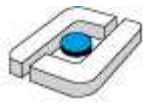


```
ant -help
```

sein, was folgendes liefert.

```
G:\>ant -help
ant [options] [target [target2 [target3] ...]]
Options:
  -help, -h              print this message
  -projecthelp, -p       print project help information
  -version               print the version information and exit
  -diagnostics           print information that might be helpful to
                         diagnose or report problems.
  -quiet, -q             be extra quiet
  -verbose, -v           be extra verbose
  -debug, -d             print debugging information
  -emacs, -e             produce logging information without adornments
  -lib <path>            specifies a path to search for jars and classes
  -logfile <file>        use given file for log
    -l     <file>                ''
  -logger <classname>    the class which is to perform logging
  -listener <classname>  add an instance of class as a project listener
  -noinput               do not allow interactive input
  -buildfile <file>      use given buildfile
    -file    <file>              ''
    -f       <file>              ''
  -D<property>=<value>   use value for given property
  -keep-going, -k        execute all targets that do not depend
                         on failed target(s)
  -propertyfile <name>   load all properties from file with -D
                         properties taking precedence
  -inputhandler <class>  the class which will handle input requests
  -find <file>           (s)earch for buildfile towards the root of
    -s  <file>           the filesystem and use it
  -nice  number          A niceness value for the main thread:
                         1 (lowest) to 10 (highest); 5 is the default
  -nouserlib             Run ant without using the jar files from
                         ${user.home}/.ant/lib
  -noclasspath           Run ant without using CLASSPATH
  -autoproxy             Java1.5+: use the OS proxy settings
  -main <class>          override Ant's normal entry point
G:\antspiel>
```

Ant arbeitet typischerweise eine Datei build.xml ab, in der die einzelnen Befehle stehen.

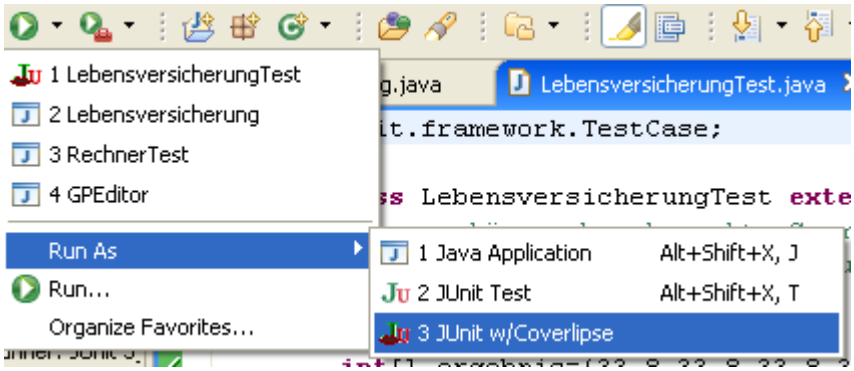


20 Kurzhinweise für weitere Werkzeuge in Eclipse

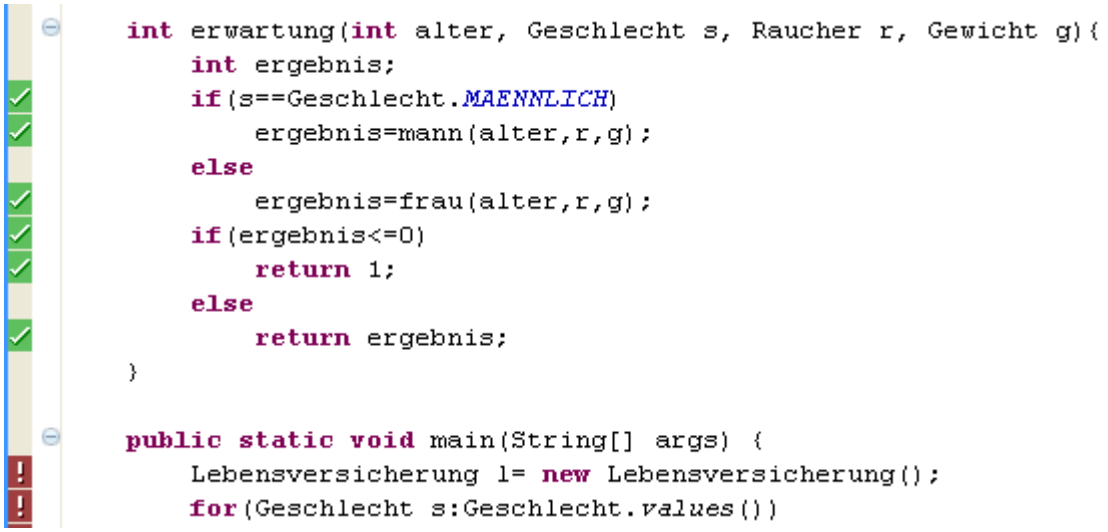
Die folgenden Werkzeuge können, wenn in Eclipse installiert, eventuell während eines SW-Projekts hilfreich sein. Sie sind nur als ein Angebot zum Ausprobieren zu sehen. Generell ist es sinnvoll, über weitere Werkzeuge nachzudenken, die dann aber auf individuellen Rechnern installiert werden müssen. Weiterhin gilt, dass teilweise notwendige individuelle Einstellungen der Werkzeuge nicht zentral vorgenommen werden können und man sich so überlegen muss, ob und wie eine Werkzeugnutzung möglich wird.

20.1 Coverlipse

Coverlipse ist ein alternatives Überdeckungswerkzeug für C0-Tests, das als Alternative zu EclEmma zur Verfügung steht, da EclEmma relativ neu für Eclipse ist (Emma gibt es schon länger). Nur JUnit kann zusammen mit Coverlipse ausgeführt werden und liefert dann Informationen, ob eine Programmzeile ausgeführt wurde oder nicht. Coverlipse wird zusammen mit JUnit ausgeführt. Dazu gibt es unter „Run As“ bei der Testausführung den Unterpunkt „JUnit w/Coverlipse“.



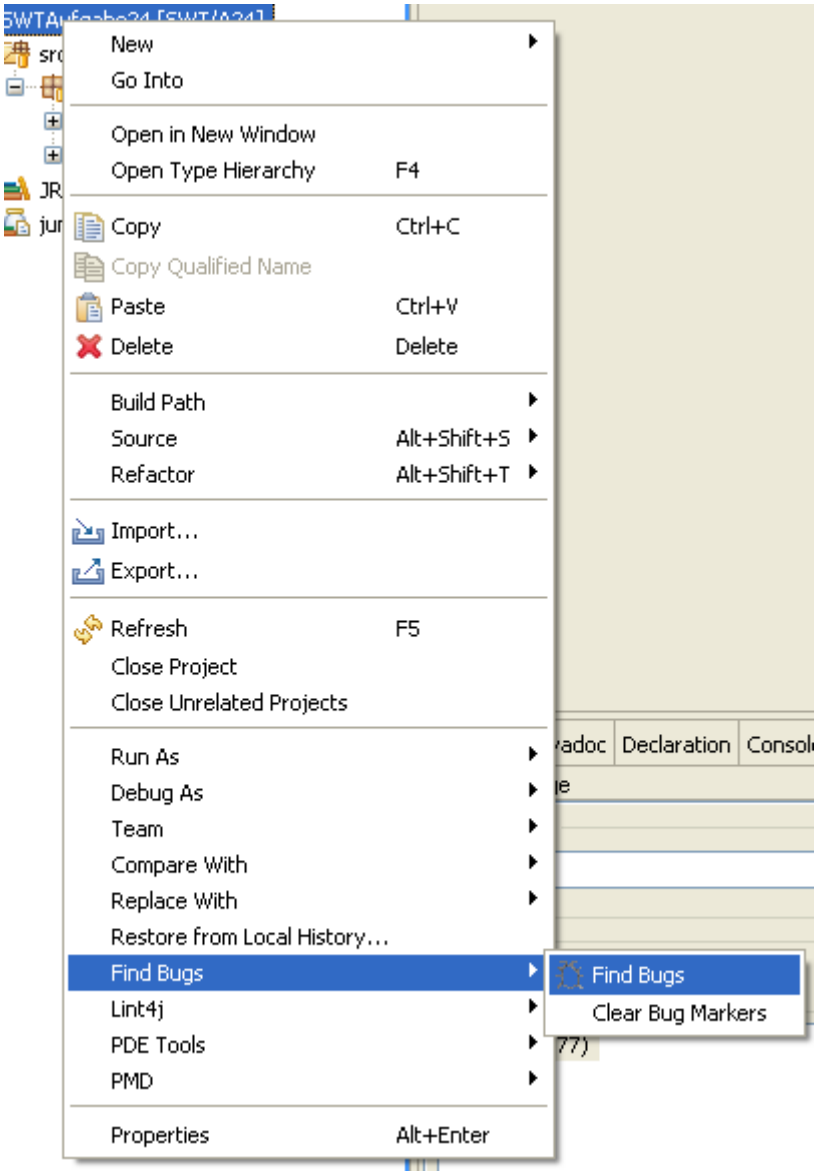
Im Java-Quellcode werden dann Markierungen gesetzt, die zeigen, ob die Zeile ausgeführt wurde.



```
int erwartung(int alter, Geschlecht s, Raucher r, Gewicht g){
    int ergebnis;
    if(s==Geschlecht.MAENNLICH)
        ergebnis=mann(alter,r,g);
    else
        ergebnis=frau(alter,r,g);
    if(ergebnis<=0)
        return 1;
    else
        return ergebnis;
}

public static void main(String[] args) {
    Lebensversicherung l= new Lebensversicherung();
    for(Geschlecht s:Geschlecht.values())
```

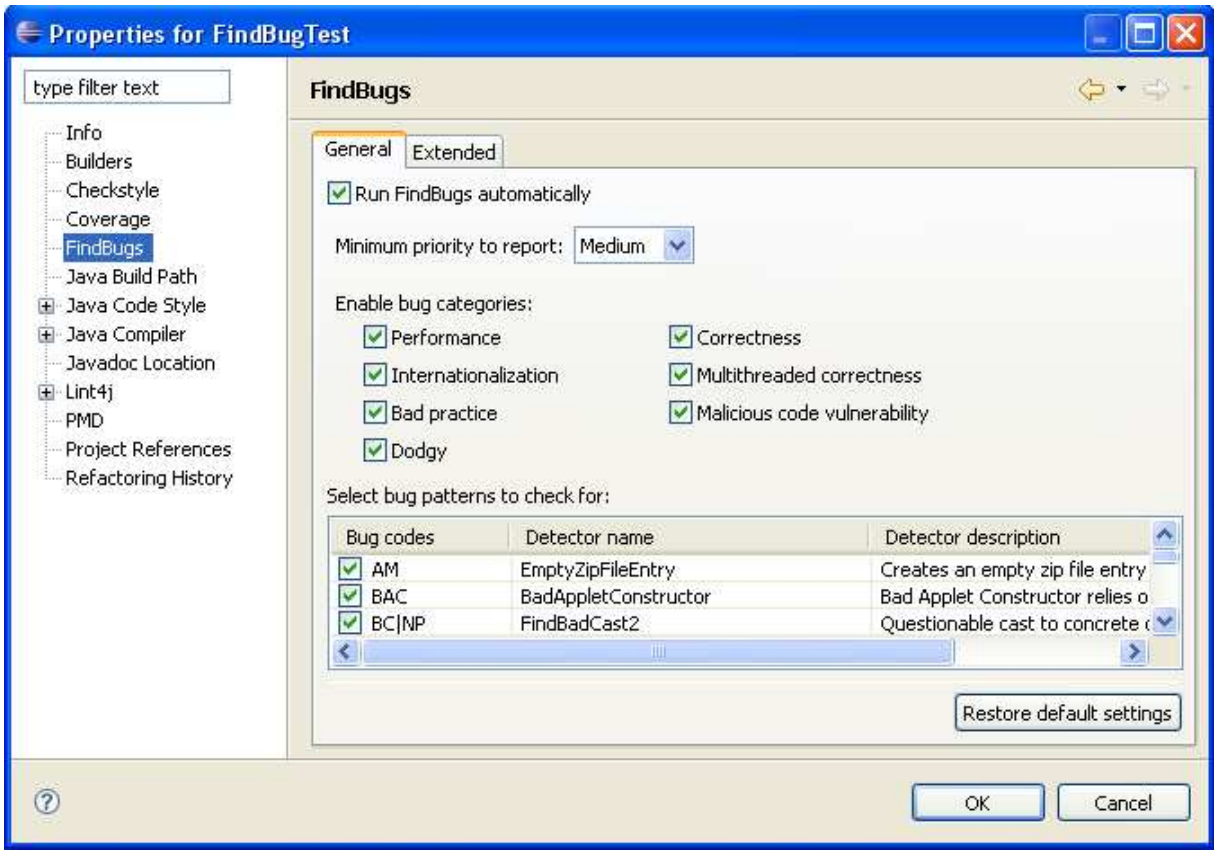
20.2 FindBugs

Mit FindBugs wird der Quellcode nach möglichen Fehlern durchsucht. Wichtig ist dabei, dass es sich um potenzielle Problemfälle handelt, die nicht alle bearbeitet werden müssen. Nützlich sind z. B. Hinweise, dass eine Variable eine null-Referenz enthalten könnte und dass dann keine Methode für die Variable ausgeführt werden kann. FindBugs enthält eine Menge von Regeln, die man für das jeweilige Projekt konfigurieren kann.

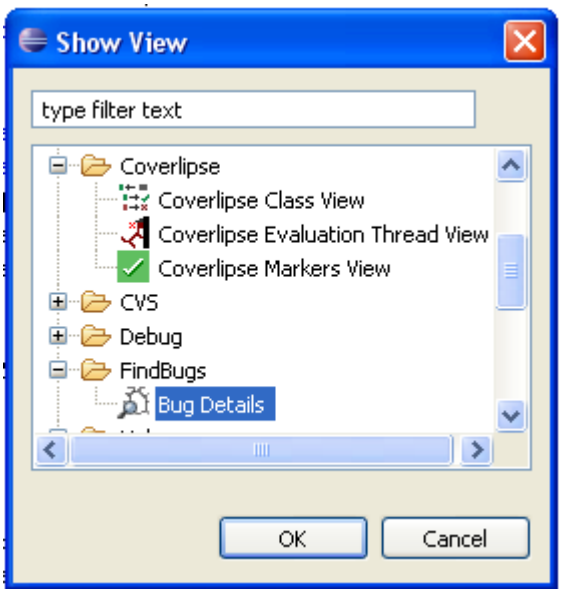


FindBugs wird mit einem Rechtklick auf dem Projekt unter „Find Bugs“ mit „Find Bugs“ gestartet. Potenzielle Fehler werden als Warnung markiert. Um diese Markierungen wieder zu entfernen, wird der Punkt „Clear Bug Markers genutzt“.

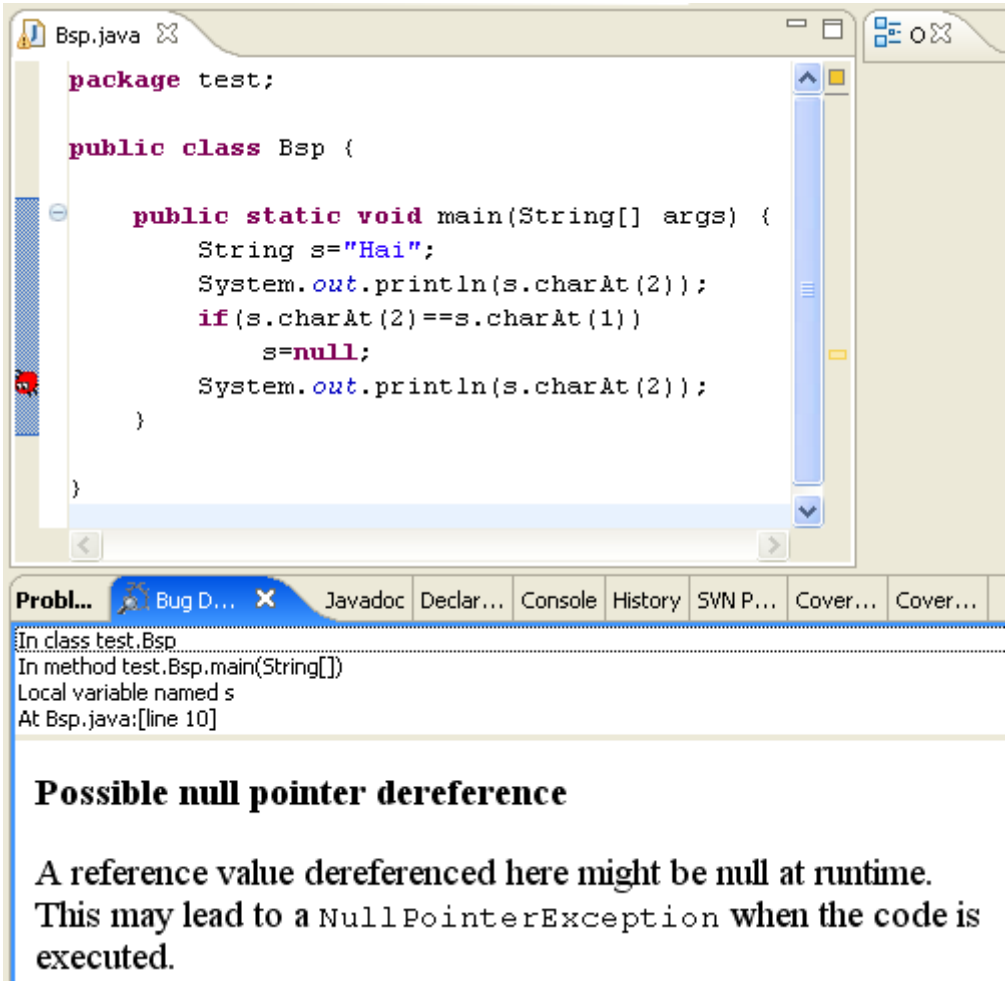
Die genaue Einstellung, was FindBugs untersuchen soll, wird unter „Properties" eingestellt. Zur Einstellung steht der folgende Dialog, hier für ein Beispielprojekt FindBugTest, zur Verfügung.



Unter „Window > Show View > Others“ steht ein View zur Verfügung, der einen generellen Kommentar zum potenziellen Bug liefert.

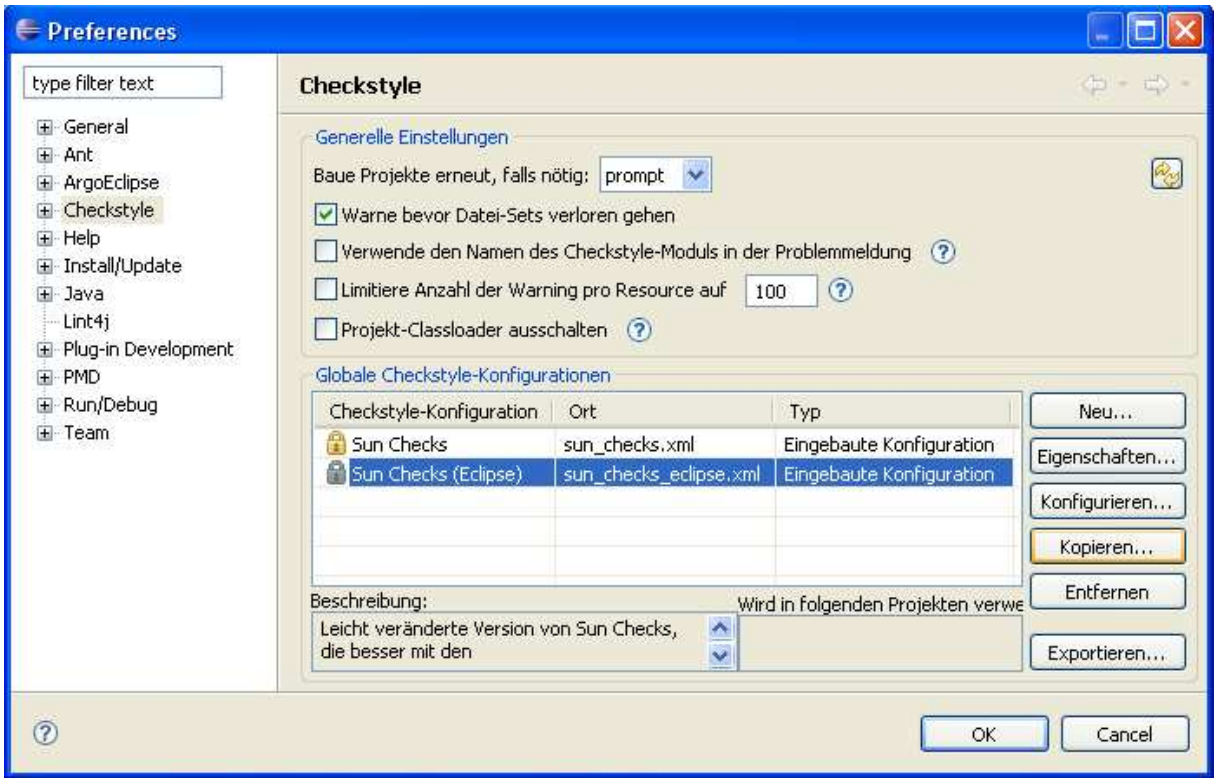


Das folgende Bild zeigt einen auf der linken Seite von FindBugs markierten potenziellen Bug und den dazugehörigen Kommentar.

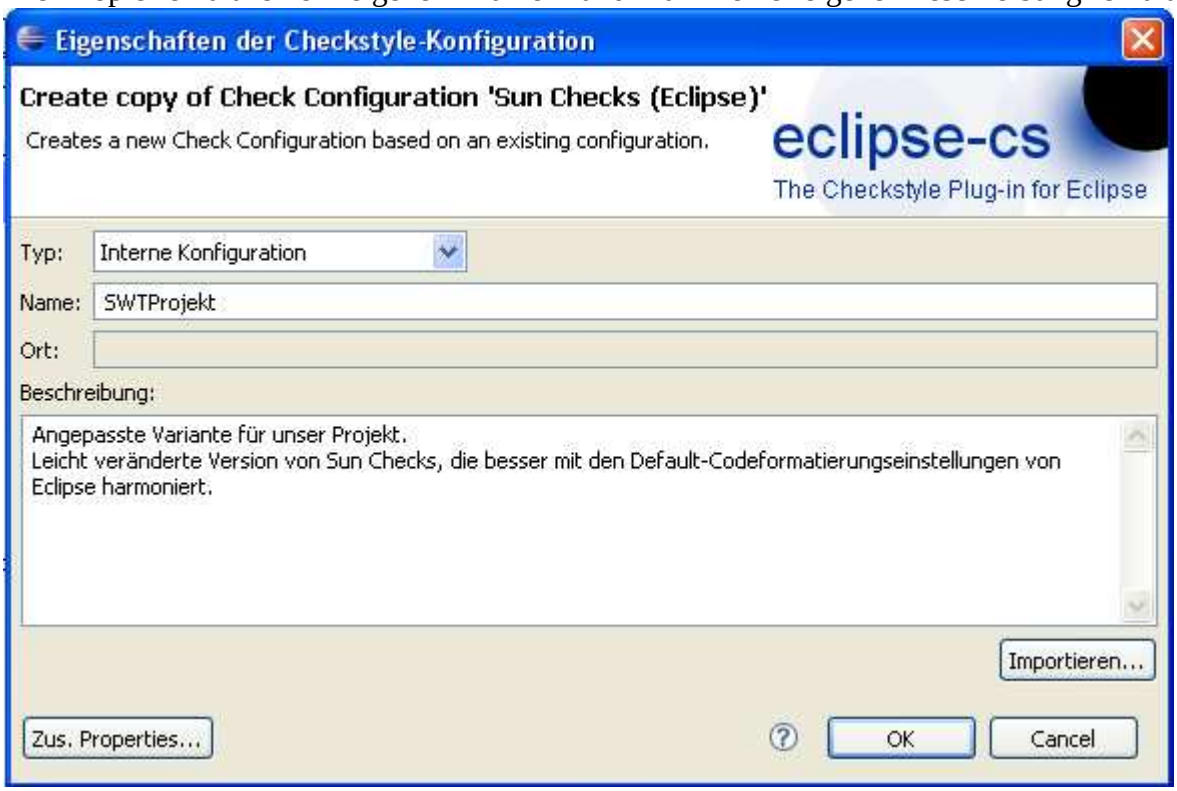


20.3 Checkstyle

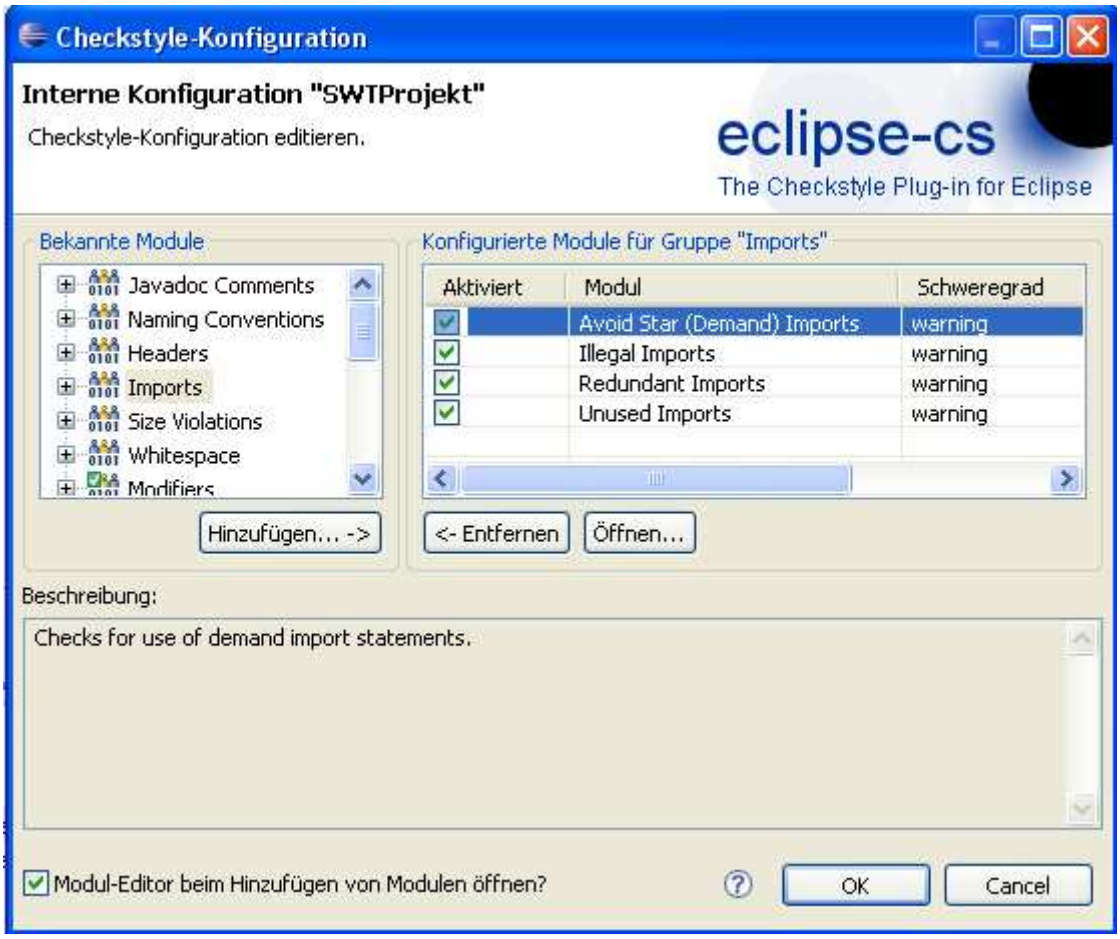
Wenn man sich auf Coding-Guidelines geeinigt hat, dann müssen diese überprüft werden. Diese Überprüfung ermöglicht Checkstyle. Dabei steht eine sehr große Auswahl von Regeln zur Verfügung, so dass eine projektindividuelle Konfiguration unbedingt vorgenommen werden muss. Diese Regelmenge wird unter „Window > Preferences“ vorgenommen. Als Ausgangspunkt ist es sinnvoll, „Sun Checks (Eclipse)“ zu nehmen und auf der rechten Seite auf „Kopieren...“ zu drücken.



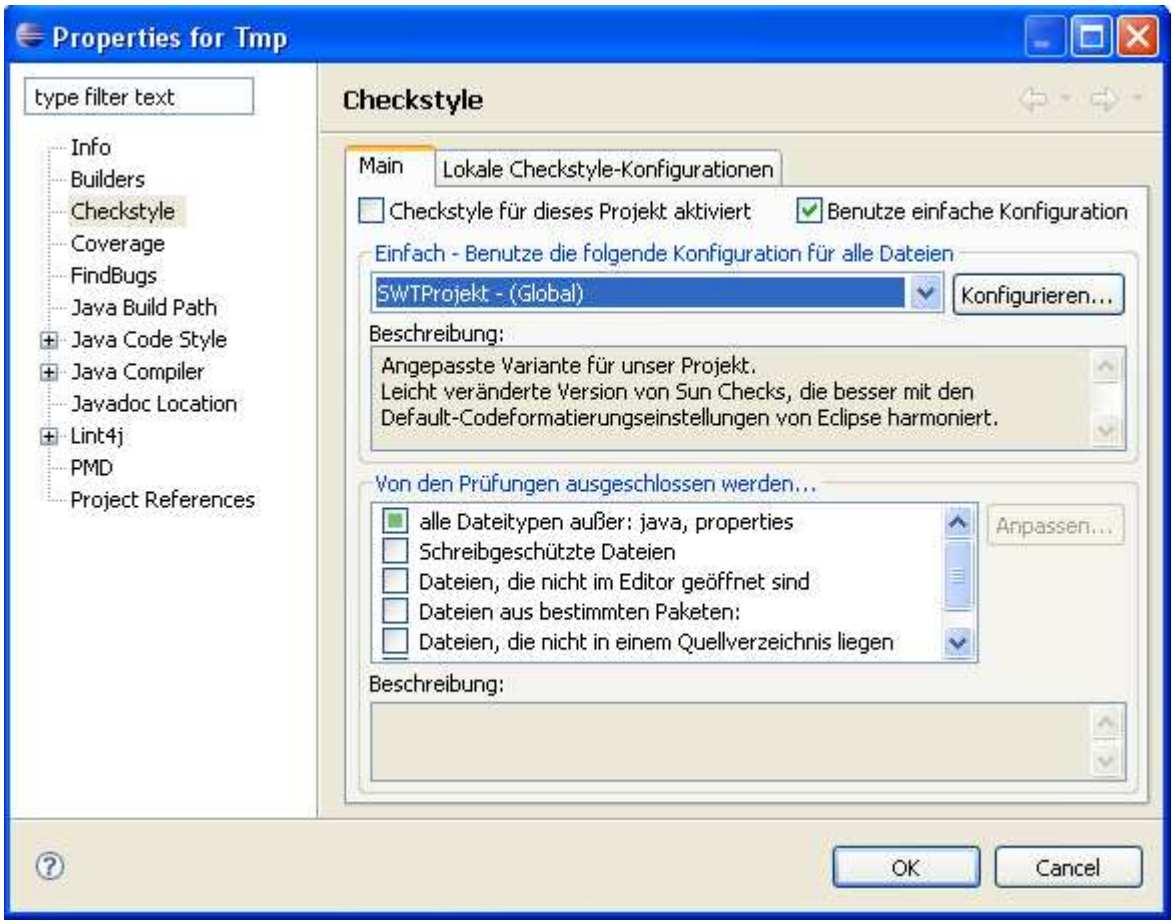
Die Kopie erhält einen eigenen Namen und kann eine eigene Beschreibung erhalten.



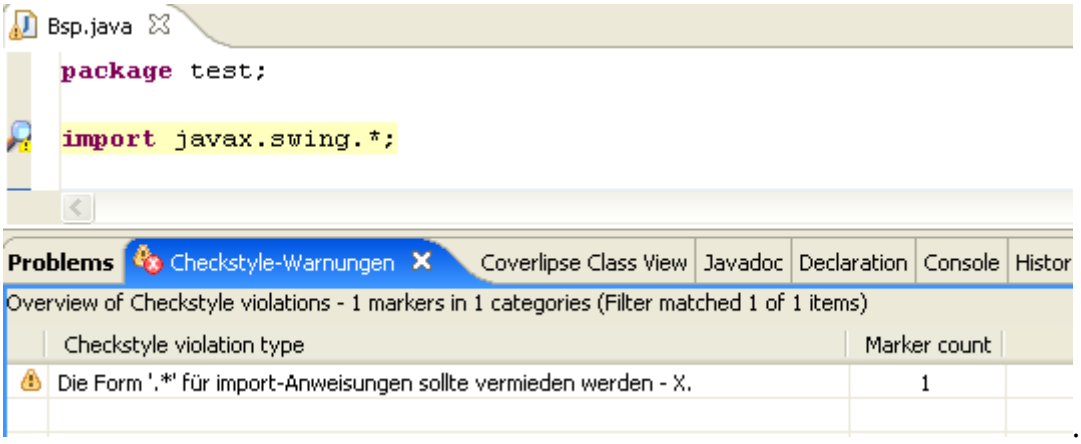
Nach einem „OK“ steht die Konfiguration zur Verfügung und kann unter „Konfigurieren...“ individuell eingestellt werden. Ein Ausschnitt aus dem vorhandenen Regelkatalog sieht wie folgt aus.



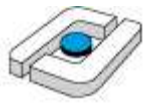
Checkstyle muss über die Properties des Projekts aktiviert werden. Weiterhin kann eingestellt werden, ob Checkstyle automatisch ausgeführt werden soll. Die neu erstellte Konfiguration muss dabei ausgewählt werden.



Probleme werden dann mit einem gesonderten Symbol angezeigt, weiterhin gibt es unter „Window > Show View > Other“ die Möglichkeit, einen View für Checkstyle-Warnungen auszuwählen.



Falls man viele Regeln aktiviert hat, ist es sinnvoll, CheckStyle während der Bearbeitung auszuschalten und zu Prüfungen wieder zu aktivieren.



21 Kurzhinweise für weitere Werkzeuge

Die folgenden Werkzeuge könnten auf dem FH-Rechner installiert und eventuell während eines SW-Projekts hilfreich sein. Sie sind nur als ein Angebot zum Ausprobieren zu sehen. Generell ist es sinnvoll, über weitere Werkzeuge nachzudenken, die dann aber auf individuellen Rechnern installiert werden müssen.

21.1 ArgoUML

ArgoUML (ArgoUML-0.24.zip) ist ein mächtiges freies UML-Modellierungswerkzeug, mit dem man auch Quellcodegerüste generieren kann. Die Handhabung ist etwas gewöhnungsbedürftig, wobei sich gerade der untere Teil der Diagramme zum Manövrieren und Ergänzen eignet. Der Start findet durch einen Doppelklick auf argouml.jar im Ordner C:\programme\argouml statt. Falls Sie irrtümlich ein Entpackprogramm mit *.jar assoziiert haben, sollten Sie diese Verknüpfung lösen. Alternativ können Sie in einer DOS-Box **`java -jar argouml.jar`** ausführen. Die Nutzbarkeit für größere Projekte kann aber wegen größerer Detailprobleme eventuell angezweifelt werden.

21.2 Abbot

Abbot ist ein sogenanntes Capture-and-Replay-Werkzeug zum Testen von SW ausgehend von Ihrer Oberfläche. Aktionen des Nutzers können aufgezeichnet und wieder abgespielt werden. In die Aufzeichnungen können Zusicherungen eingebaut werden, die bei erneuten Abläufen geprüft werden. Man findet die Dateien unter added/abbot im Eclipse-Verzeichnis.

21.3 Jester

Jester ist ein JUnit-Testtester, mit dem die Genauigkeit von Junit 3-Tests analysiert werden kann. Man findet die Dateien unter added/jester137 im Eclipse-Verzeichnis.

21.4 Ganttproject

Ganttproject (Installation: ganttproject-2.0.3.exe) kann bei der Projektplanung hilfreich sein, da man Projekte auf Phasen aufteilen, Abhängigkeiten zwischen den Phasen spezifizieren und den Phasen Ressourcen zuordnen kann. Das Werkzeug ist bei Weitem nicht so mächtig wie andere Managementwerkzeuge, aber zum Planen kleiner Projekte und zum Einstieg in die Nutzung eines solchen Werkzeugs kann Ganttproject hilfreich sein.