Nutzungshinweise für Eclipse

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Thema:	Nutzung der SW-Entwicklungsumgebung Eclipse
Autor:	Prof. Dr. Stephan Kleuker
Version / Datum:	8.0.8 / 12.08.2021
Empfänger:	Teilnehmer der Lehrveranstaltungen im Bereich SW-Engineering

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 Praktika wird für konkrete Aufgaben auf einzelne Kapitel 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 wiederherzustellen.

0 KleukersSEU

Um Konflikte mit anderen Software-Paketen zu vermeiden, bietet Prof. Dr. Kleuker Teilnehmern seiner Veranstaltungen ein Verzeichnis an, dass alle in seinen Veranstaltungen benötigte Software beinhaltet. Dieses Verzeichnis ist auf den Hochschulrechnern installiert und kann auf eigene Rechner oder einfach einen USB-Stick kopiert werden. Es ist dabei zu beachten, dass Einstellungen, die im Nutzerkonto gespeichert werden, auf jedem Rechner neu einzurichten sind.

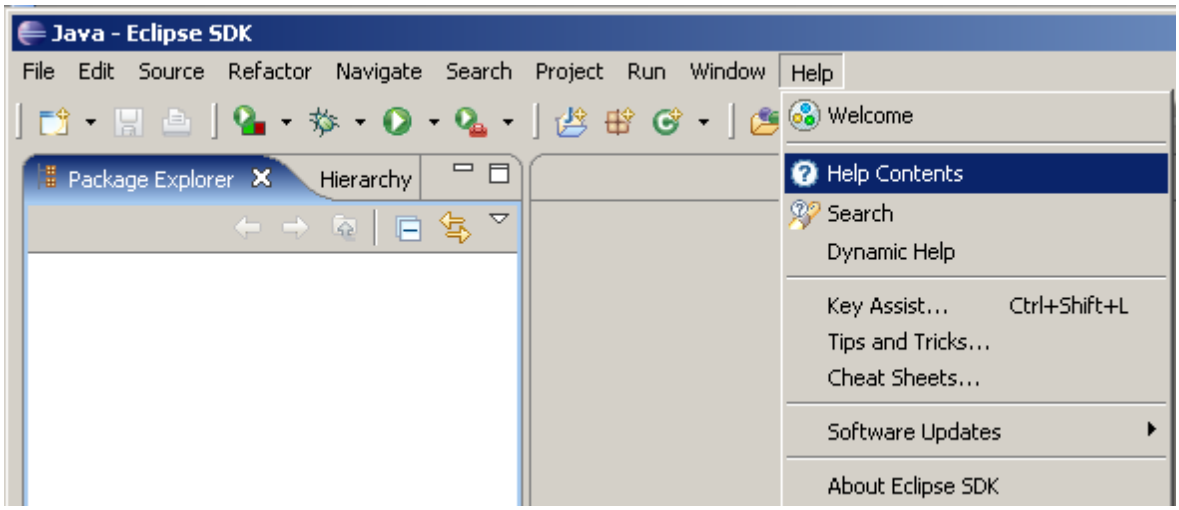
Weitere Hinweise und das Download-Paket können http://home.edvsz.hs-osnabrueck.de/skleuker/kleukersSEU/index.html entnommen werden.

1 Vorbemerkungen

In verschiedenen Lehrveranstaltungen wird Eclipse als durchgehendes Werkzeug genutzt, wobei verschiedene Plugins zur Erweiterung der Funktionalität genutzt werden. Man sollte nicht versuchen, das „ultimative“ Eclipse mit vielen Plugins zu bauen, da Eclipse dann deutlich langsamer wird. Es ist allerdings unproblematisch mehrere Eclipse-Versionen parallel auf einem Rechner zu nutzen.

Damit Eclipse funktioniert, ist vorher Java, genauer ein JDK ab der Version 6 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 im System (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. Abbildungen sind nicht benannt und nummeriert, da es sich hier um ein Arbeitspapier und keine wissenschaftliche Veröffentlichung handelt.

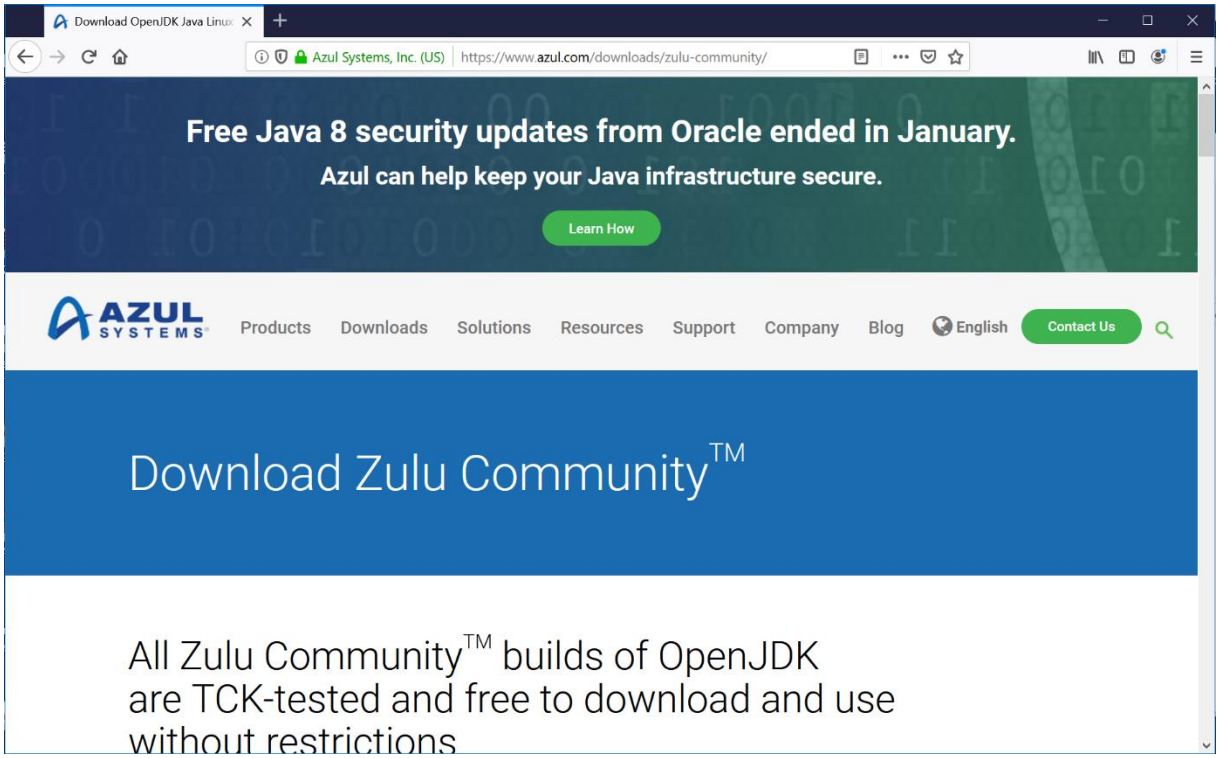
Insofern relevant, wird im Titel der Kapitel bzw. Unterkapitel die Version angegeben, auf die sich der Text bezieht. Dabei wurden nicht alle Bilder aktualisiert, wenn sich inhaltlich nichts verändert hat.

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.

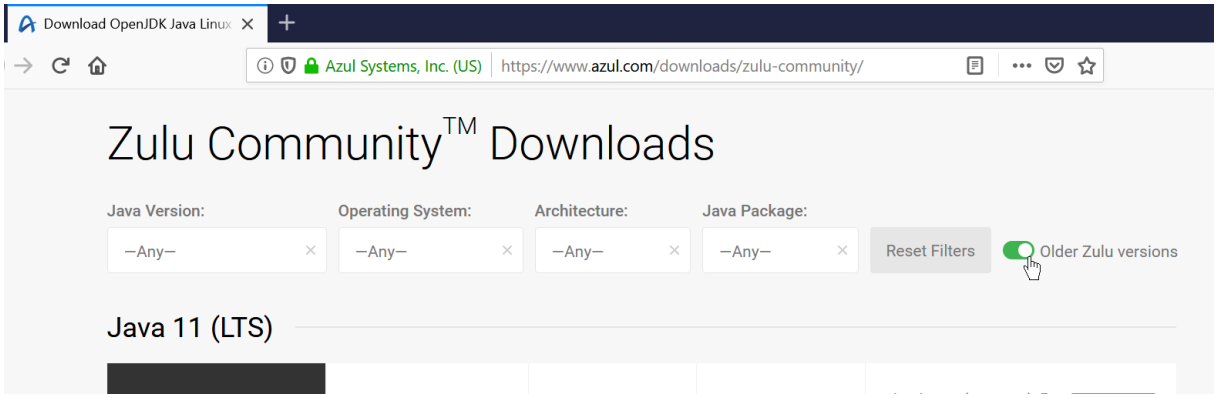
2 Installation von Java (JDK 11)

Vorbemerkung: Wie alle Programmiersprachen entwickelt sich Java weiter, dabei werden meist neue Programmbibliotheken ergänzt oder hinzugefügt, die Funktionalität anbieten, die bisher in Java noch nicht vorhanden war. Weiterhin werden Befehle ergänzt, die es erfahrenen Programmierern erleichtern, etwas kürze Programme zu schreiben. Bis einschließlich der Version 8 liefen (praktisch) alle in anderen Versionen von Java geschriebenen Programme auch in dieser Version. Dies ist ab der Version 9 nicht mehr ganz der Fall, was einige Vorteile (modularere Software-Entwicklung, Auslieferung ohne vollständige Java Run Time) hat, aber langfristig kritische strategische Auswirkungen haben kann. Andere Programmiersprachen wie PHP und einige JavaScript-Frameworks haben deutlich massiver gegen diese Abwärtskompatibilität verstoßen und nicht wesentlich in ihrer Beliebtheit eingebüßt. Trotzdem ist zu erwarten, dass viele Projekte auch weit über 2019 hinaus auf Java 8 basieren. Weiterhin bleibt anzumerken, dass bis Java 8 zwischen den Wechsel der Hauptversionsnummer, z. B. zwischen Java 7 und Java 8 oft mehrere Jahre lagen, nach einer neuen Release-Strategie diese Nummer aber durchaus mehrfach im Jahr verändert werden kann. Für Programmieranfänger sei angemerkt, dass alles Lehrmaterial welches ab Java 7 (teilweise Java 5) entstanden ist, relativ problemlos weiter genutzt werden kann.
Eine weitere Verwirrung um Java wurde in 2018 von Oracle initiiert, indem versucht wird kommerziell direkt mit der Sprache Geld zu verdienen, ein über 20 Jahre unnötiger Ansatz. Genauer müssen kommerzielle Kunden für die Nutzung der Java Virtual Machine (JVM) zahlen, die zur Ausführung von Java-Programmen benötigt wird. Für Studierende und Entwickler bleibt die Nutzung (zur Zeit ?) frei. Da der Java Code Open Source ist, kann jeder sein eigenes Java übersetzen und frei zur Verfügung stellen. Dies machen einige Anbieter, deren Java frei nutzbar auch für kommerzielle Nutzer bleibt. Um diesen Ansatz zu unterstützen wird hier eine solche Java-Variante installiert. Wichtig ist, dass die daraus resultierenden Programme unabhängig von der genutzten Java-Installation sind.

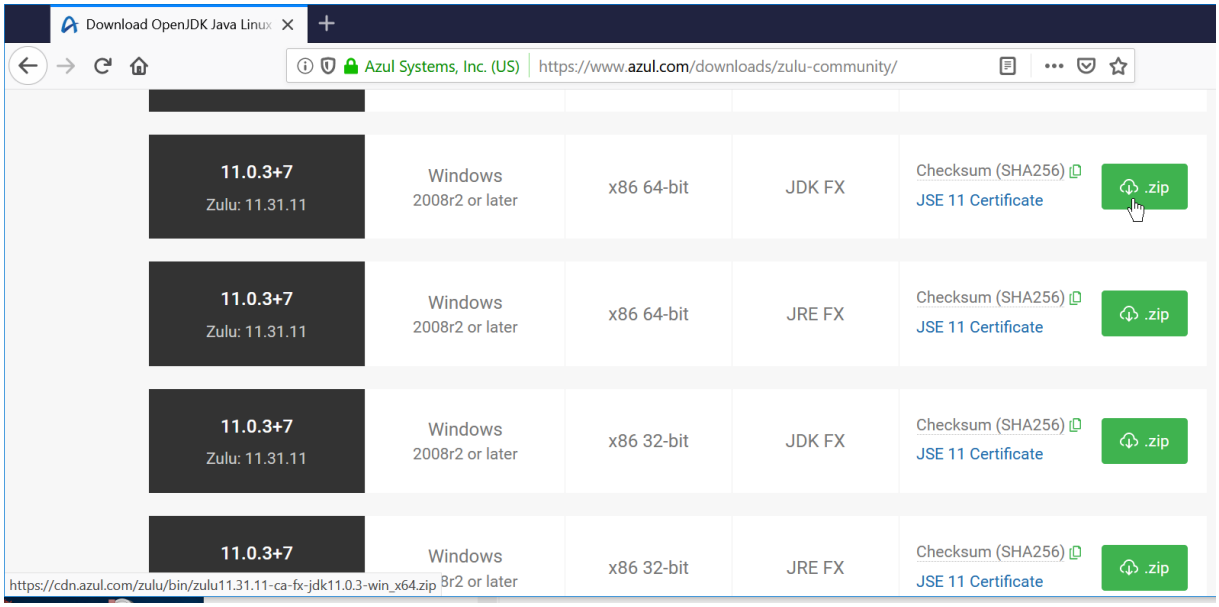
Java wird direkt von der Seite von Azul Systems https://www.azul.com/downloads/zulu-community/ heruntergeladen.



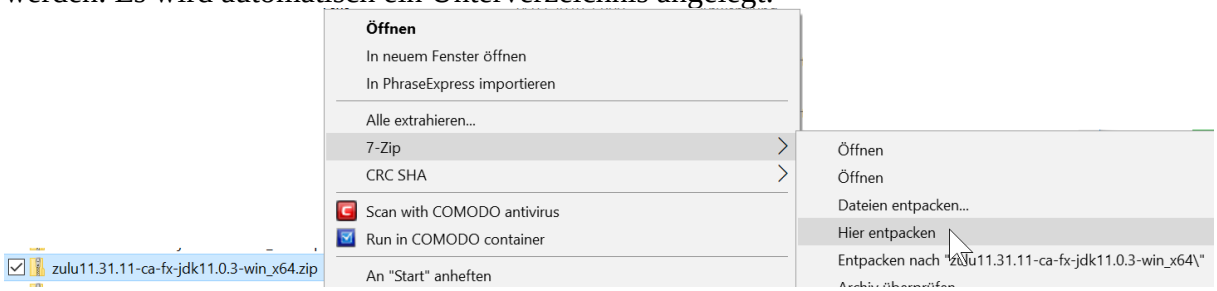
Es wird eine LTS (Long Term Service)-Version, hier 11, genutzt. Die Installation für andere Varianten verläuft identisch. Genauer soll eine Version mit bereits eingebauter Unterstützung von Java FX (jetzt JFX) geladen werden. Java FX ist eine Java-Bibliothek zur Entwicklung graphischer Oberflächen, die bis Java 8 Teil der Standard-Distribution war und bei anderen Versionen etwas aufwändig nachinstallieret werden muss. Java FX wird u. a. bei BlueJ und als Hilfsmittel in einigen Praktikumsaufgaben genutzt. Da Java FX nicht in allen Versionen von Zulu gepflegt wird, muss gegebenenfalls nach einer etwas älteren Version gesucht werden. Bei der betrachteten Webseiten-Variante ist dazu der Schieber „Older Zulu versions“ nach rechts zu setzen.



Es wird eine Java 11 LTS-Version mit JFX gesucht und heruntergeladen. Die Web-Seite zeigt auch, dass hier noch eine Version für 32-Bit-Systeme (x86 32-bit) angeboten wird. Eventuell funktioniert noch folgender Link: https://cdn.azul.com/zulu/bin/zulu11.31.11-ca-fx-jdk11.0.3-win_x64.zip. Wichtig ist, dass ein JDK, ein Java Development Kit, heruntergeladen wird, da nur damit die eigene Entwicklung möglich ist. Mit einer JRE, einer Java Runtime Environment, können nur fertige Java-Programme ausgeführt werden.

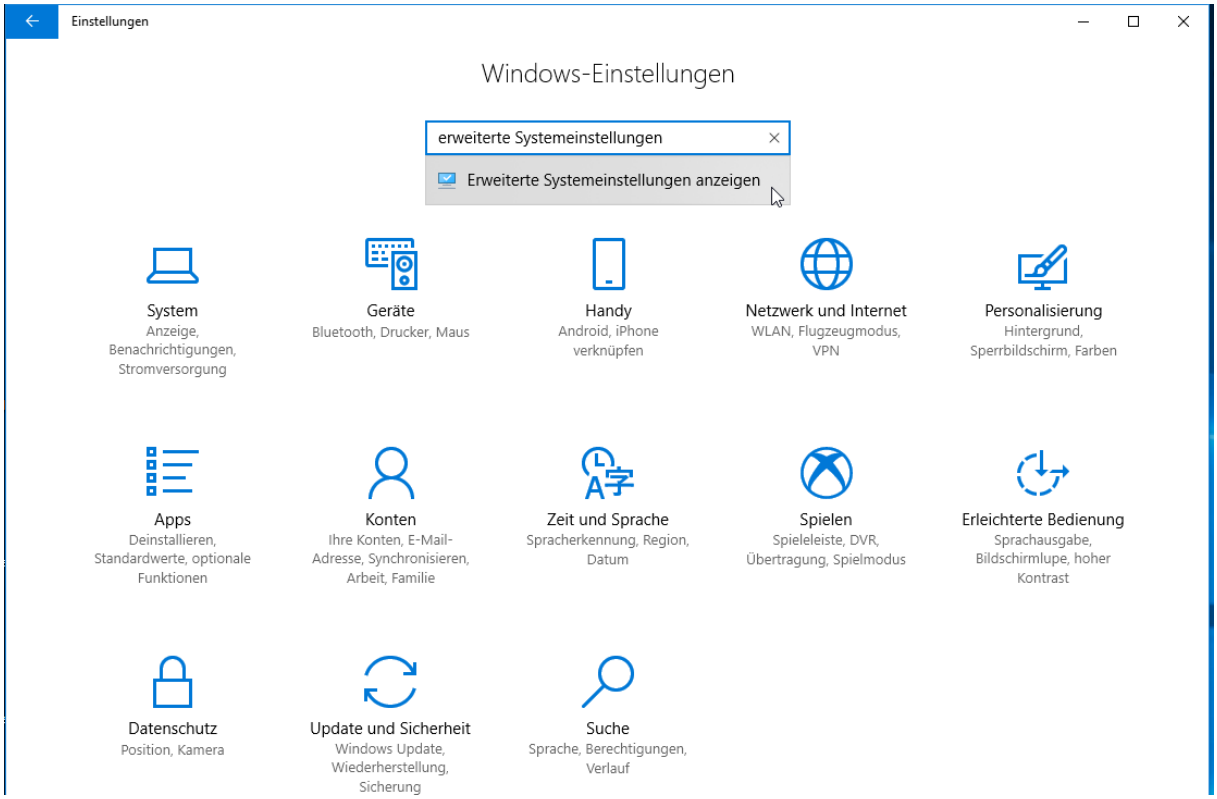


Die Zip-Datei kann an einem beliebigen Ort mit Schreibrechten mit „Extract here“ ausgepackt werden. Es wird automatisch ein Unterverzeichnis angelegt.

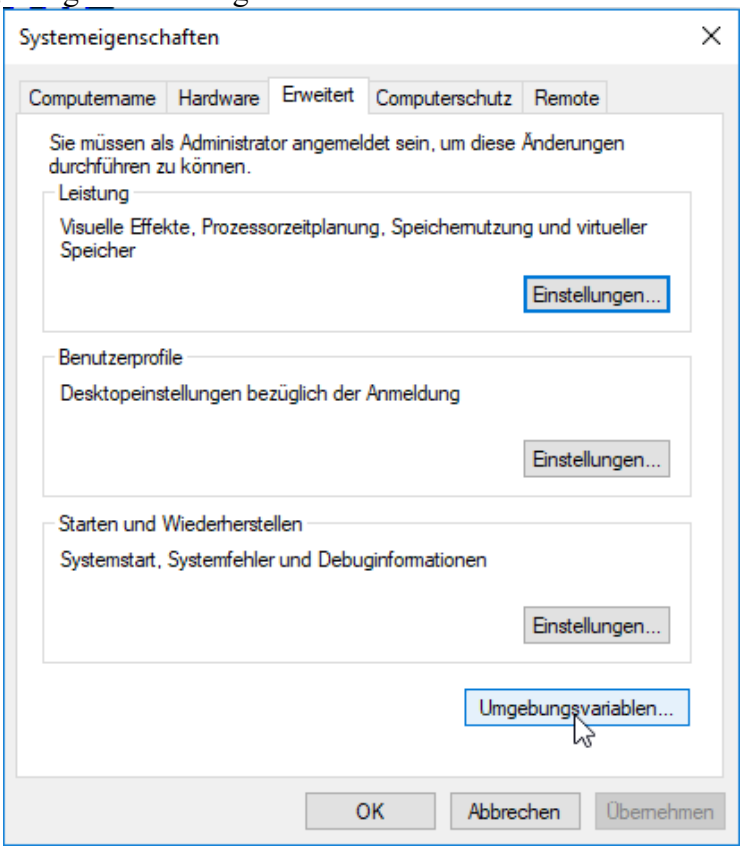


Zum Abschluss müssen einige Systemvariablen auf das Installationsverzeichnis gesetzt werden. Bei Windows 10 beginnt ein Weg mit einem Klick auf dem Start-Icon und der Auswahl „Einstellungen“.

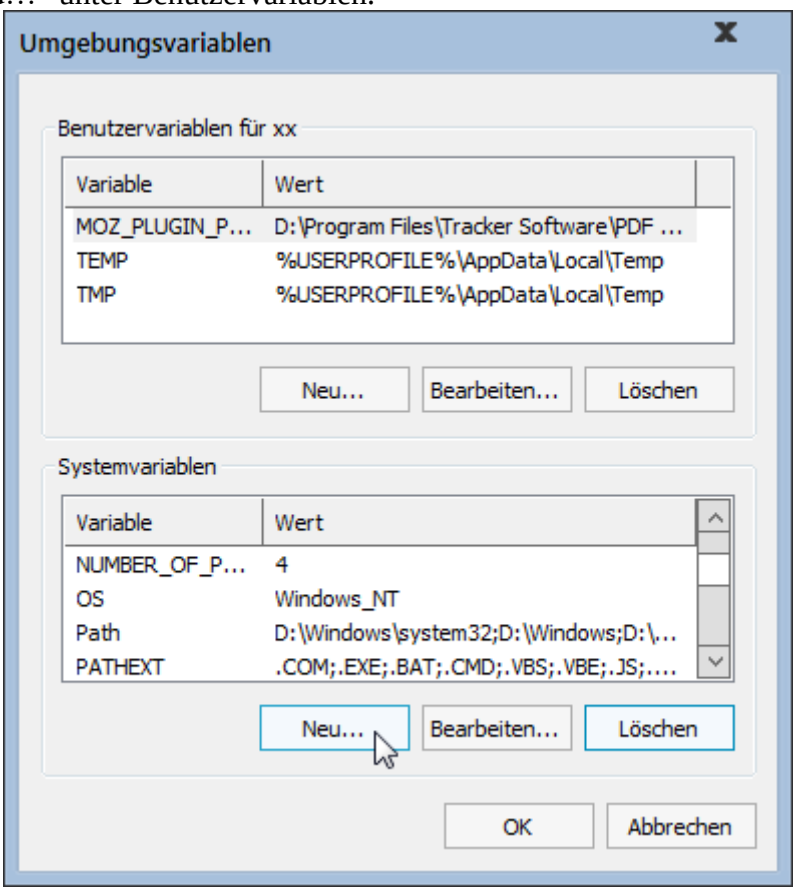
Im oberen Suchfenster wird „erweiterte Systemeinstellungen“ eingeben und darunter dann der Vorschlag mit einem Links-Klick ausgewählt.



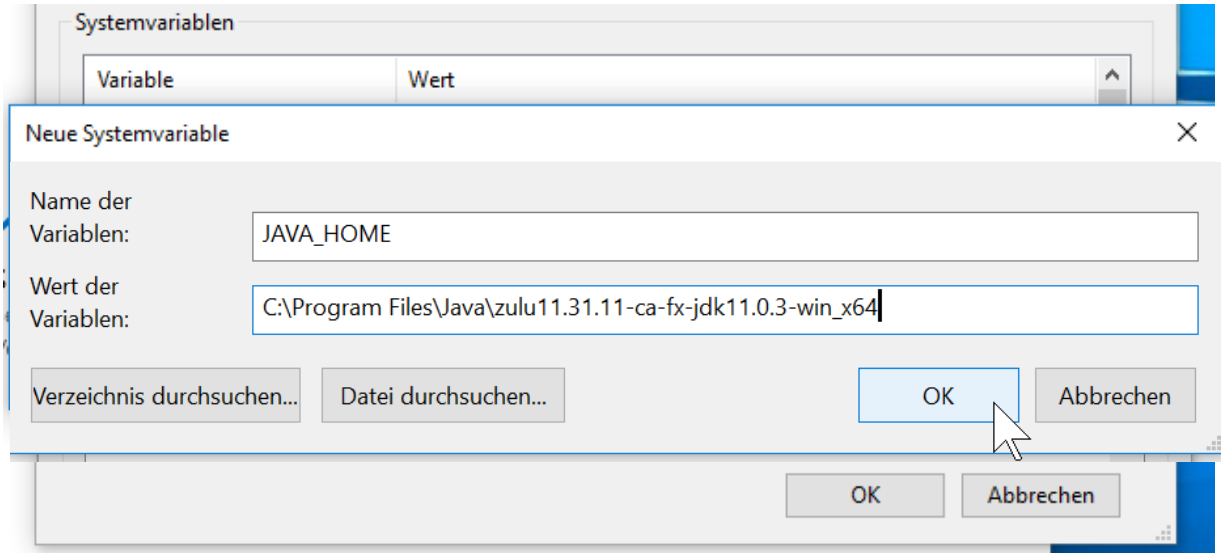
Es wird auf „Umgebungsvariablen“ geklickt.



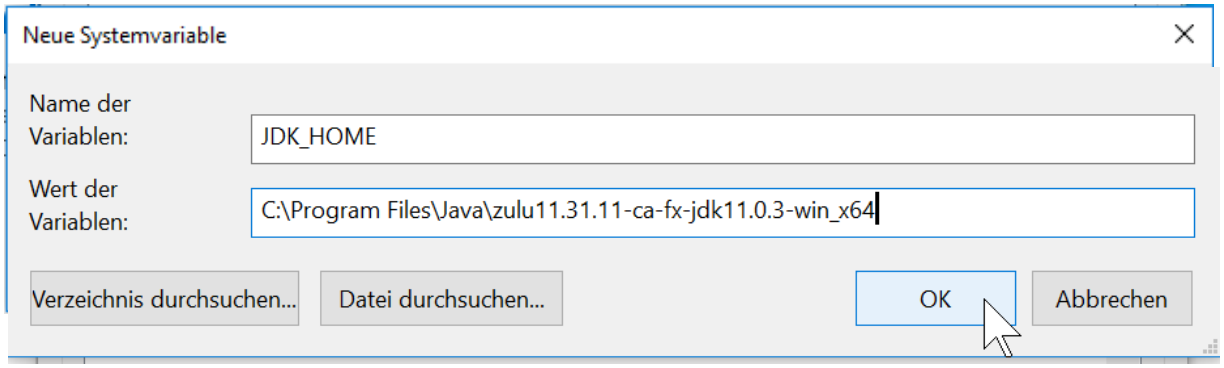
Administratoren des Systems nutzen den Knopf „Neu...“ unter Systemvariablen, andere Nutzer den Knopf „Neu…“ unter Benutzervariablen.



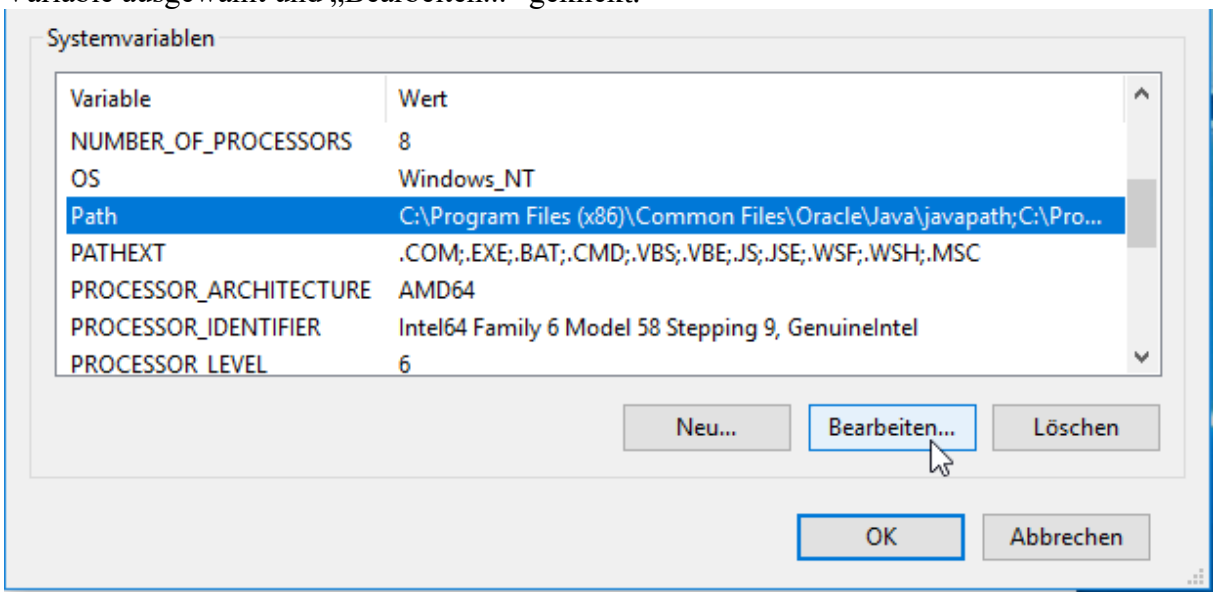
Hier werden der Variablenname sowie der Pfad zum installierten JDK eingetragen und die neue Variable mit „OK“ bestätigt. Es ist zu beachten, dass wahrscheinlich eine aktuellere Variante des JDK installiert wurde und so der letzte Ordnernamen etwas anders aussehen wird.



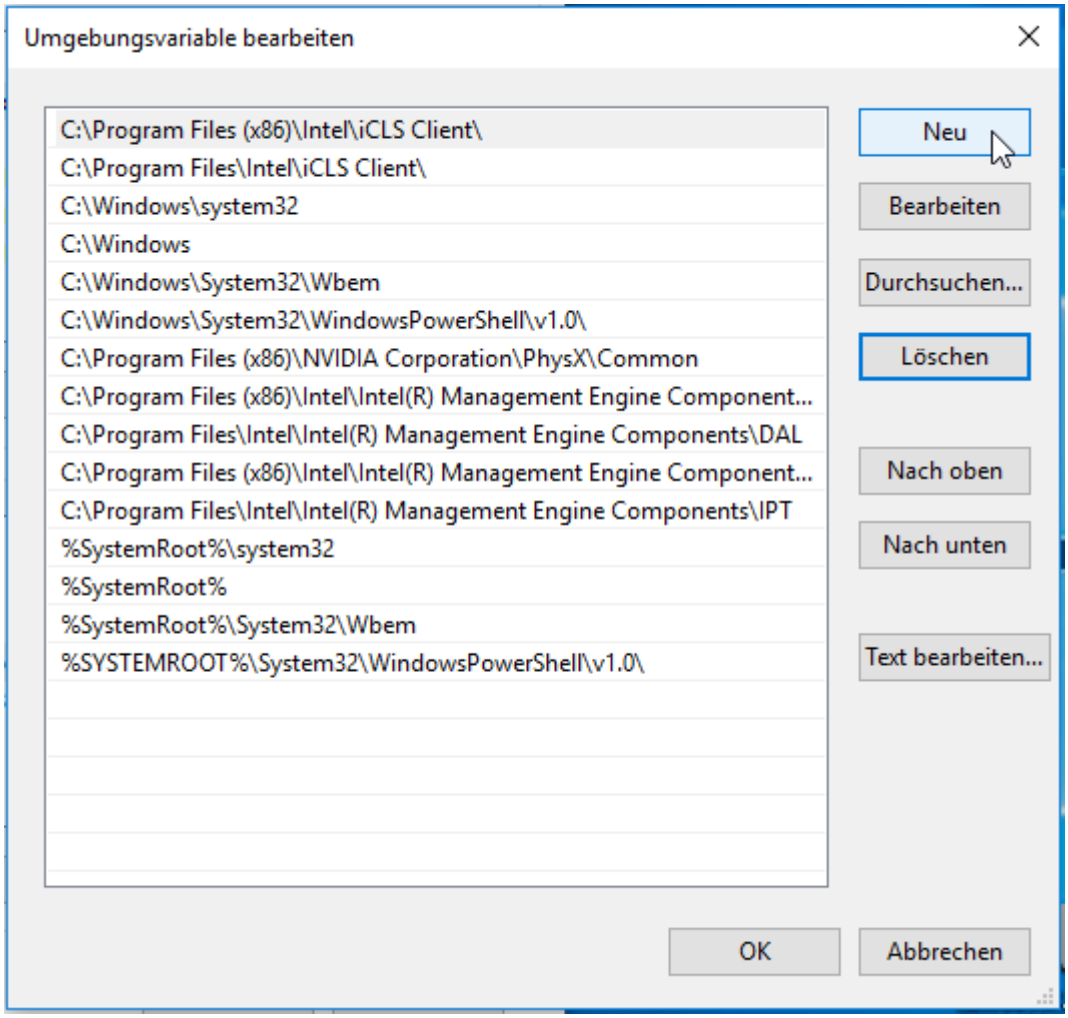
Weiterhin ist es sinnvoll, Variablen JDK_HOME und JRE_HOME mit genau dem gleichen Wert anzulegen.



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

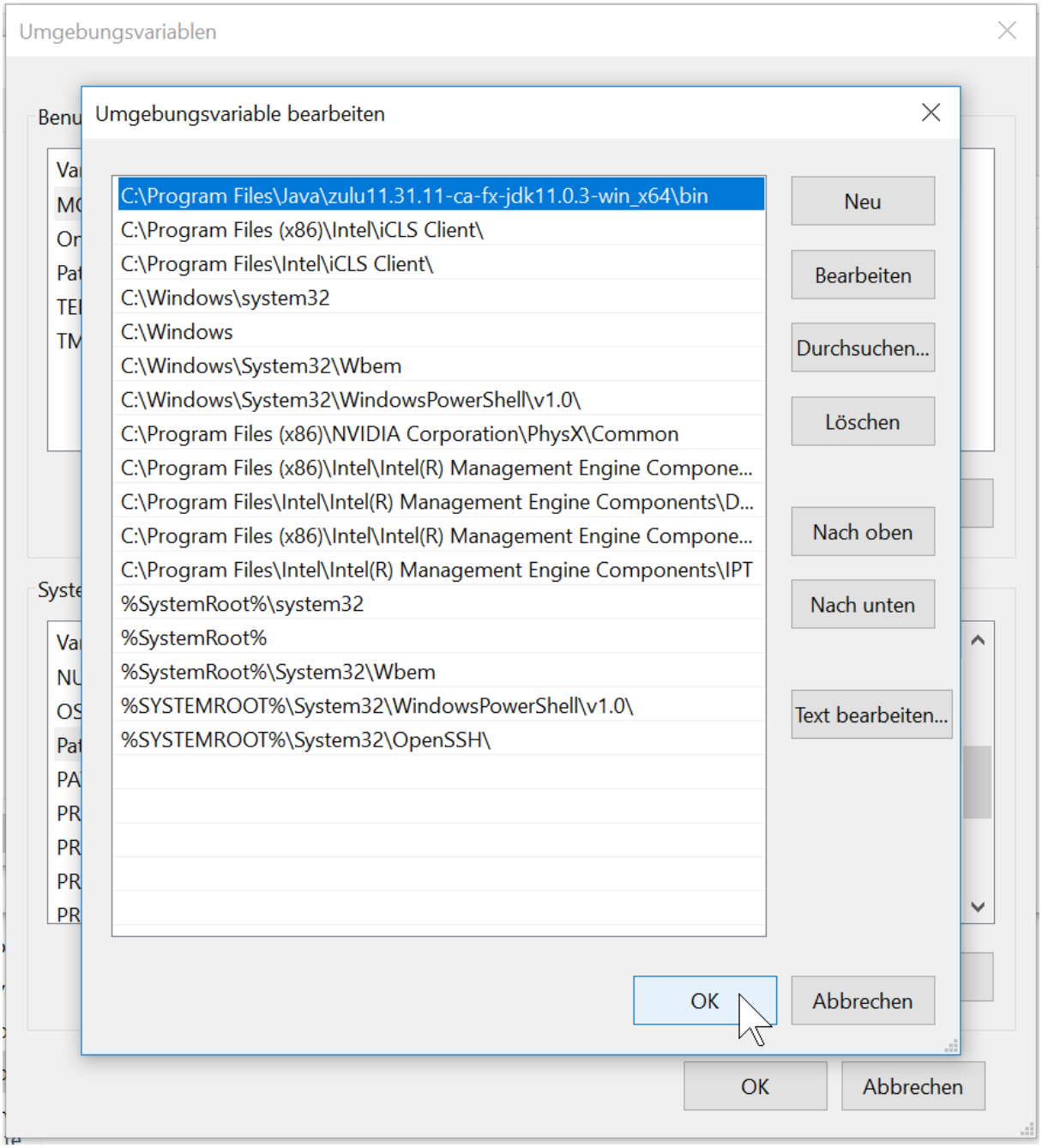


Es wird auf „Neu“ oben geklickt.

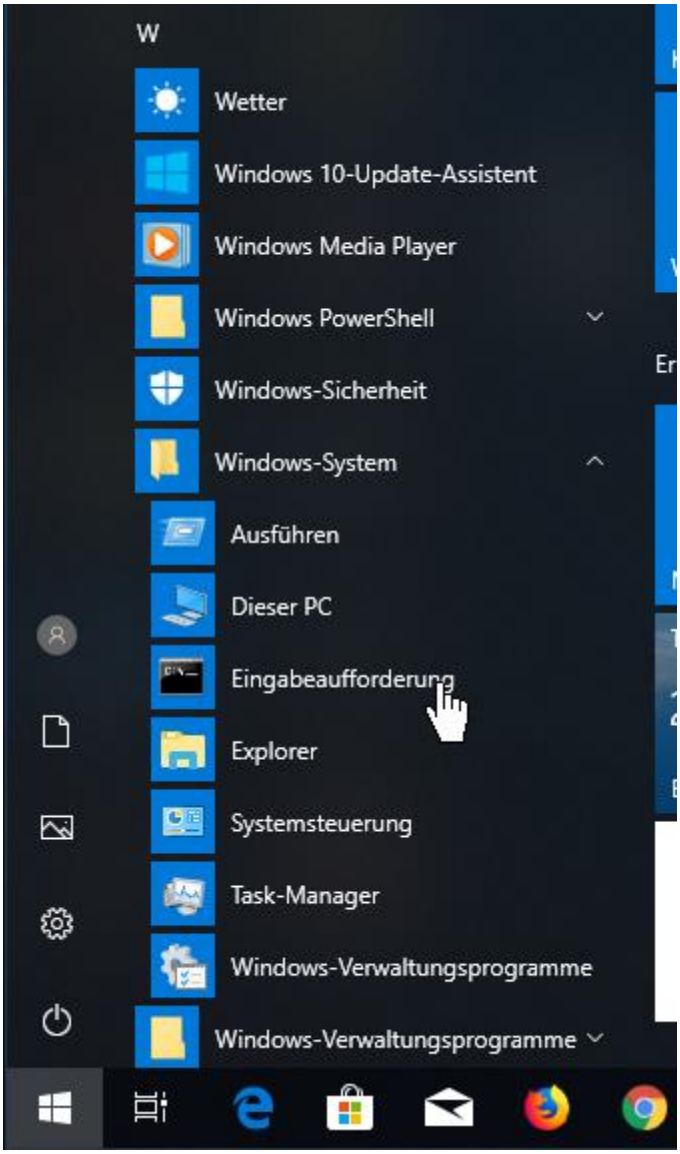


Es wird der Pfad zum bin-Verzeichnis C:\Program Files\Java\zulu11.31.11-ca-fx-jdk11.0.3-win_x64\bin eingetragen.

Danach kann der ausgewählte Pfad durch mehrfaches Klicken von „Nach oben“ in die obere Zeile geschoben werden, was allerdings für die Nutzung nicht relevant ist. Die Eintragung wird jeweils mit „OK“ bestätigt.



Zur Überprüfung der Installation kann ein Konsolenfenster genutzt werden. Dies ist z. B. über einen Klick auf das Windows-Icon links-unten, dem Scrollen zum Buchstaben „W“ bei den installierten Programmen, dem Aufklappen von „Windows-System“ und der Auswahl von „Eingabeaufforderung“ erreichbar.

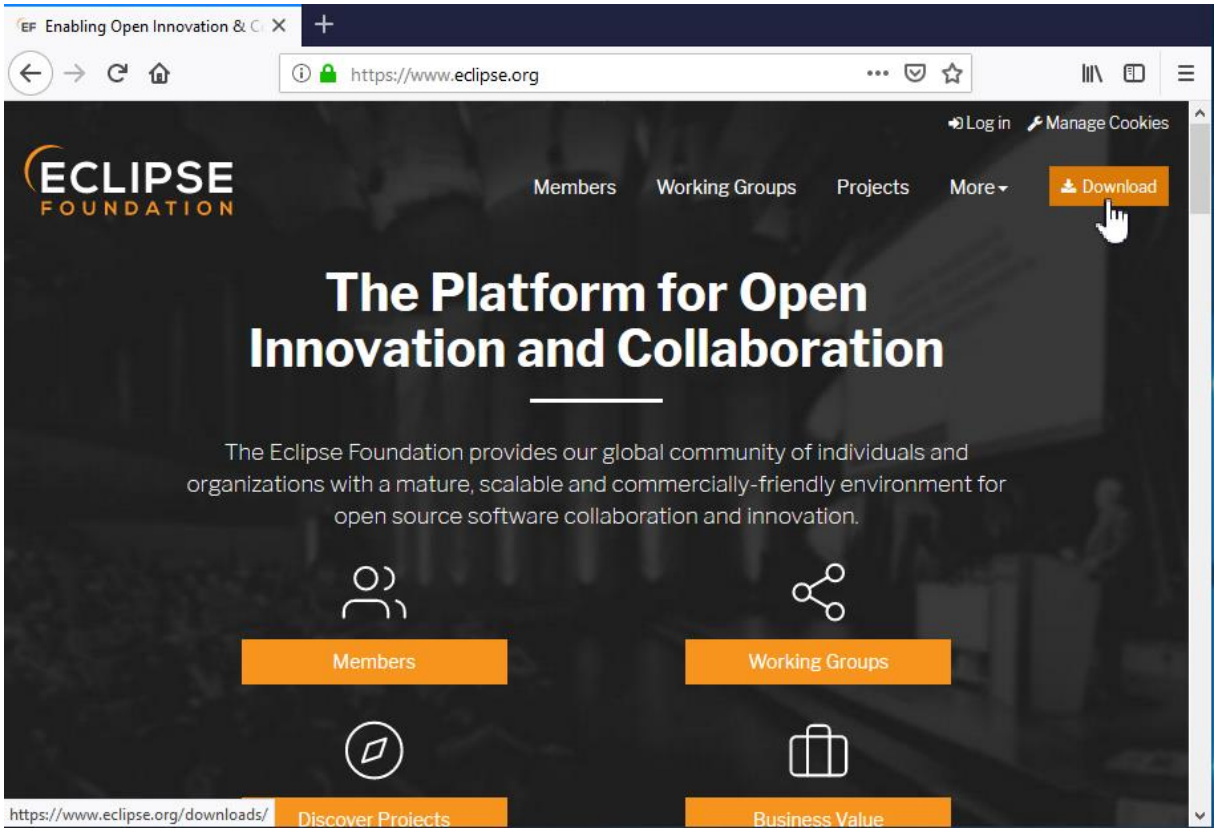


In dem Fenster wird javac –version eingegeben, was zur angezeigten Ausgabe führen muss. Java ist dann korrekt installiert.

```
Administrator: Eingabeaufforderung
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\x>javac -version
javac 11.0.3

C:\Users\x>_
```

3 Einstieg in Eclipse (4.11.0)

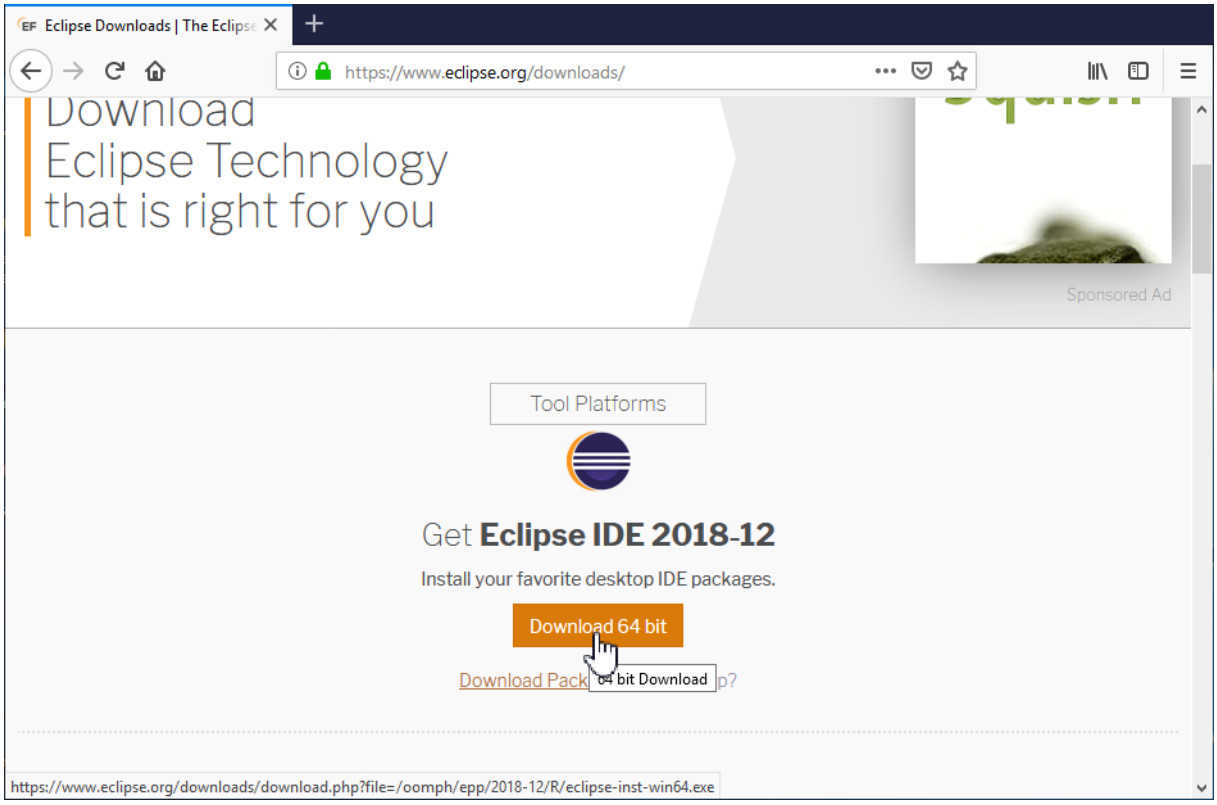
Der Download erfolgt von der Eclipse-Seite http://www.eclipse.org/ über den Download-Button, der sich abhängig von der aktuellen Seitengestaltung meist rechts-oben befindet.



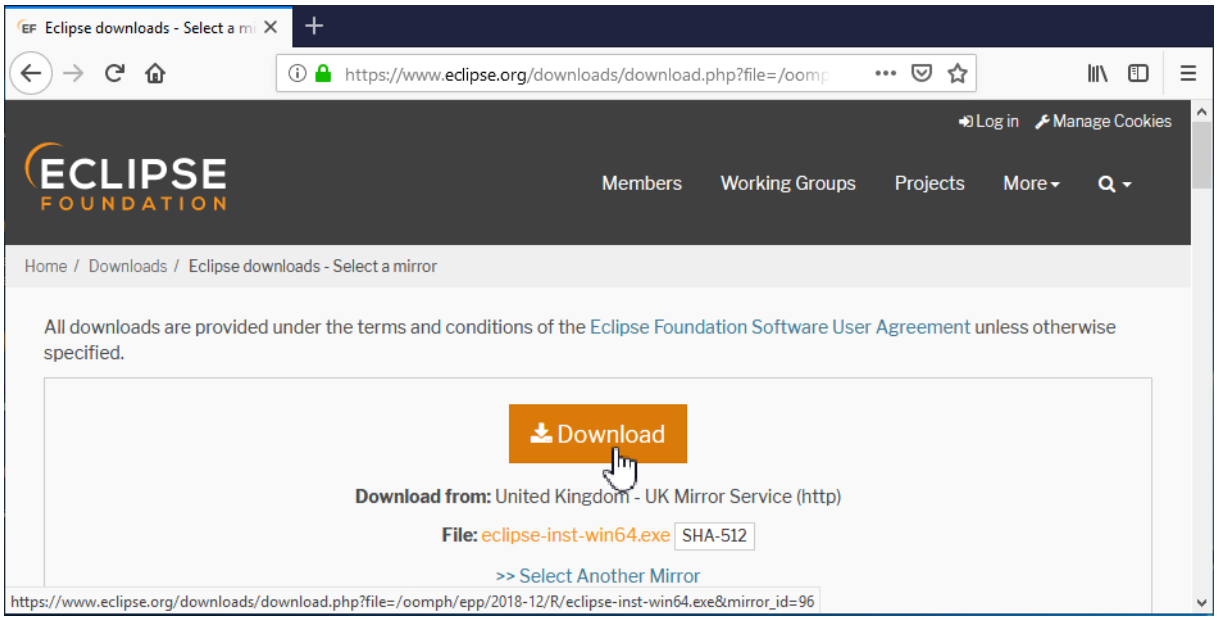
Man muss die passende Eclipse-Version auswählen, wobei weitere Funktionalität einfach durch Plugins später ergänzt werden kann. Eclipse bietet dazu einen Installer an, der spätere Änderungen etwas vereinfacht. Weiterhin ist eine einfache Zip-Datei erhältlich. In beiden Fällen können Erweiterungen nachträglich auf mehreren Wegen ergänzt werden. Hier wird zunächst der Installer betrachtet und danach auf die Zip-Version eingegangen. Die Zip-Version hat den Vorteil ohne Administrationsrechte an allen Orten ausführbar zu sdein, an denen der Nutzer Dateien nutzen darf. Dies ermöglicht z. B. die Nutzung auf einem USB-Stick. Generell isdt aber zu beachten, dass jedes Plugin die Ausführungs- und Reaktionszeit von Eclipse verlangsamen kann.

3.1 Nutzung des Eclipse-Installers

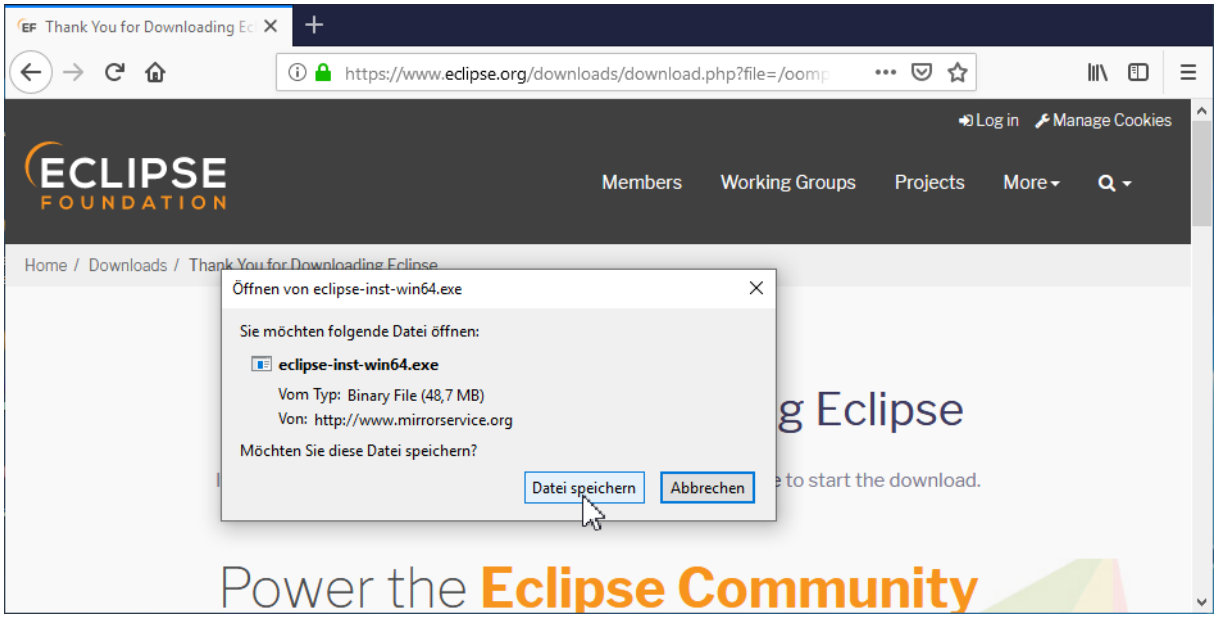
Es kann einfach der Download-Button geklickt werden.



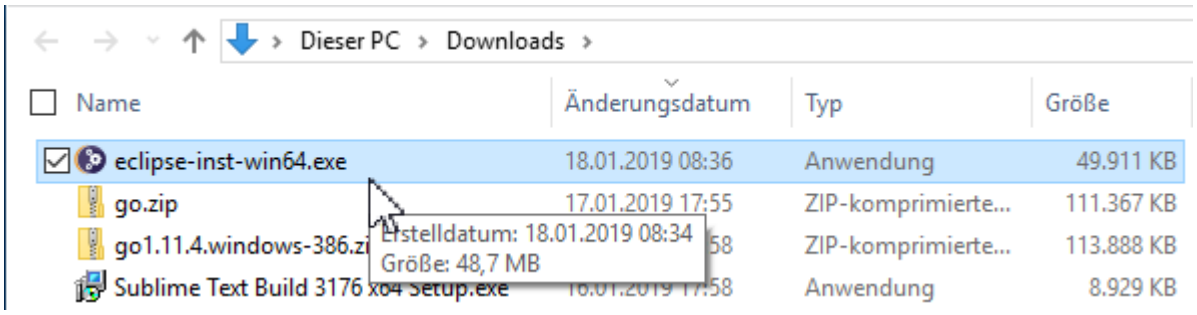
Es wird wieder der Download-Button geklickt.



Der Download sollte dann starten.

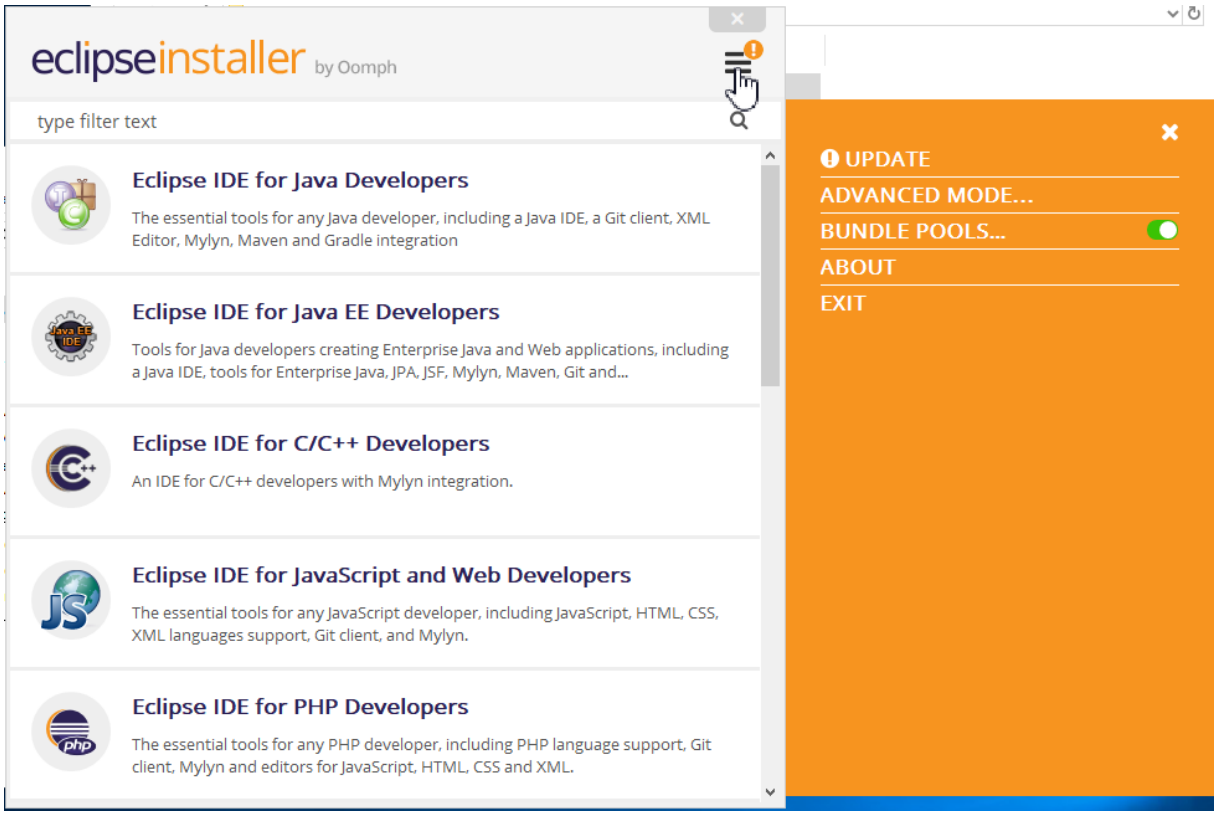


Die heruntergeladene exe-Datei wird ausgeführt.



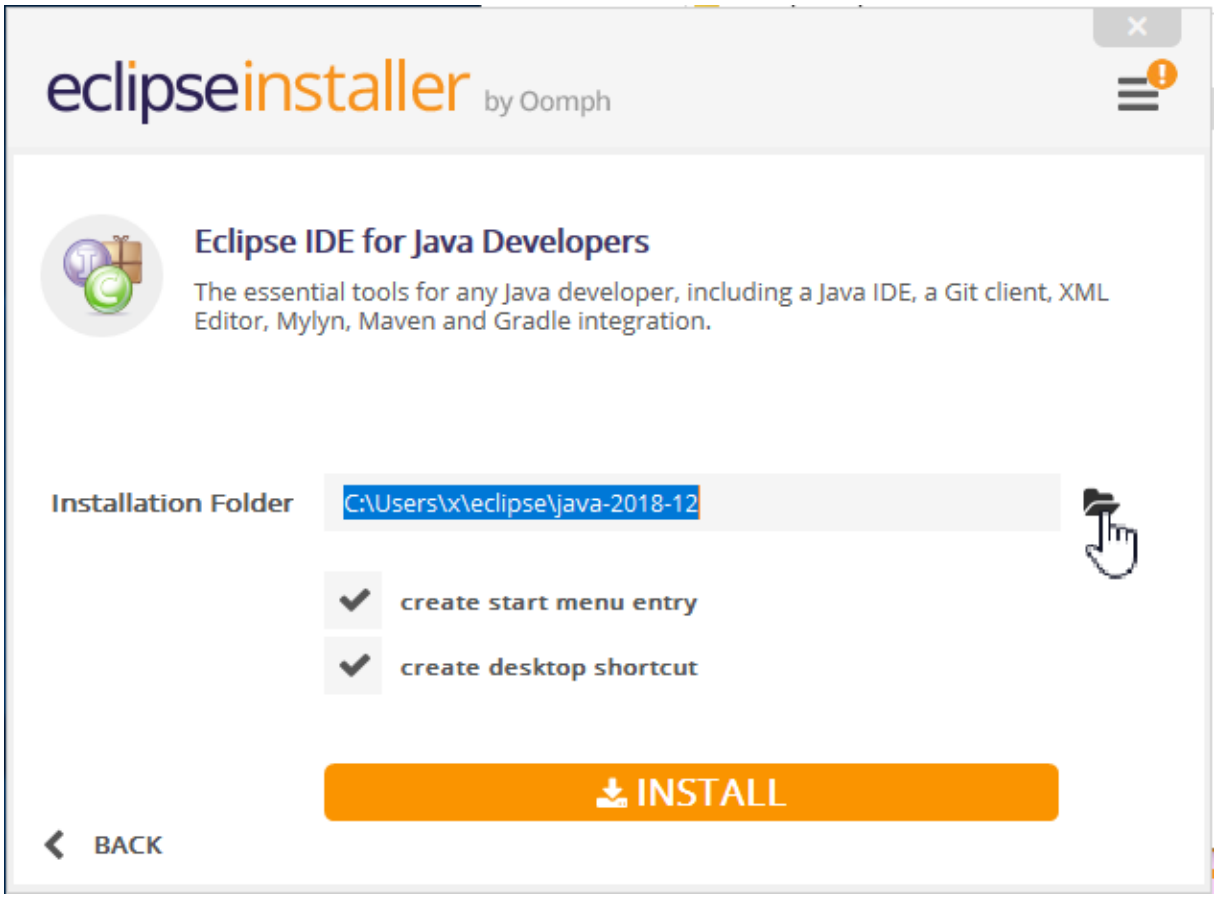
Der Start kann etwas dauern.

Es stehen einige Möglichkeiten zur Verfügung, in der folgenden Abbildung sind einige Varianten aufgezeigt, die über das Menü rechts-oben eingeblendet werden können.

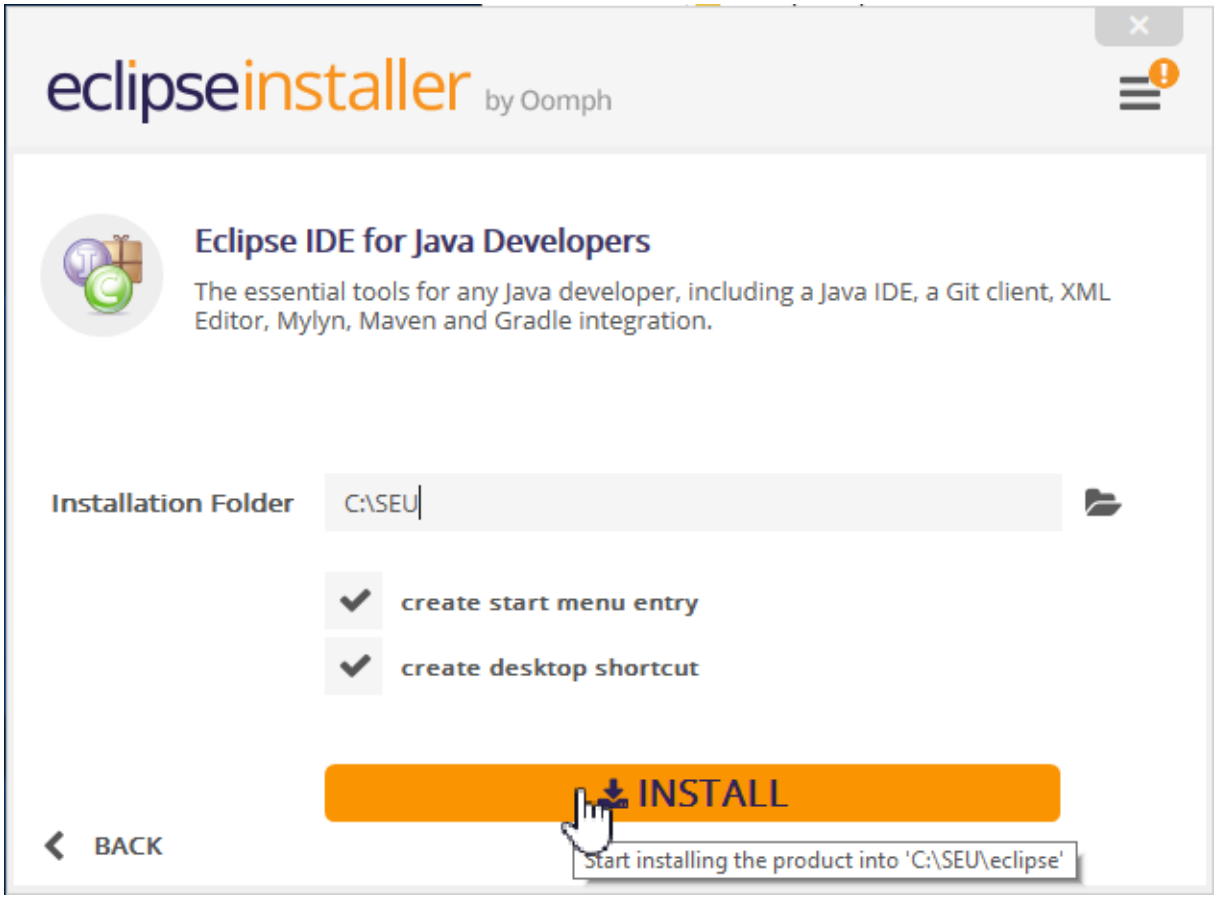


Für die erste Basisinstallation wird auf die „Eclipse IDE for Java Developers“ geklickt.

Der vorgeschlagene Installationsordner wird meist geändert. Dies erfolgt durch ein Klick rechts neben dem vorgeschlagenen Ordner. Es ist zu beachten, dass im ausgewählten Ordner ein Unterordner eclipse erstellt weird.

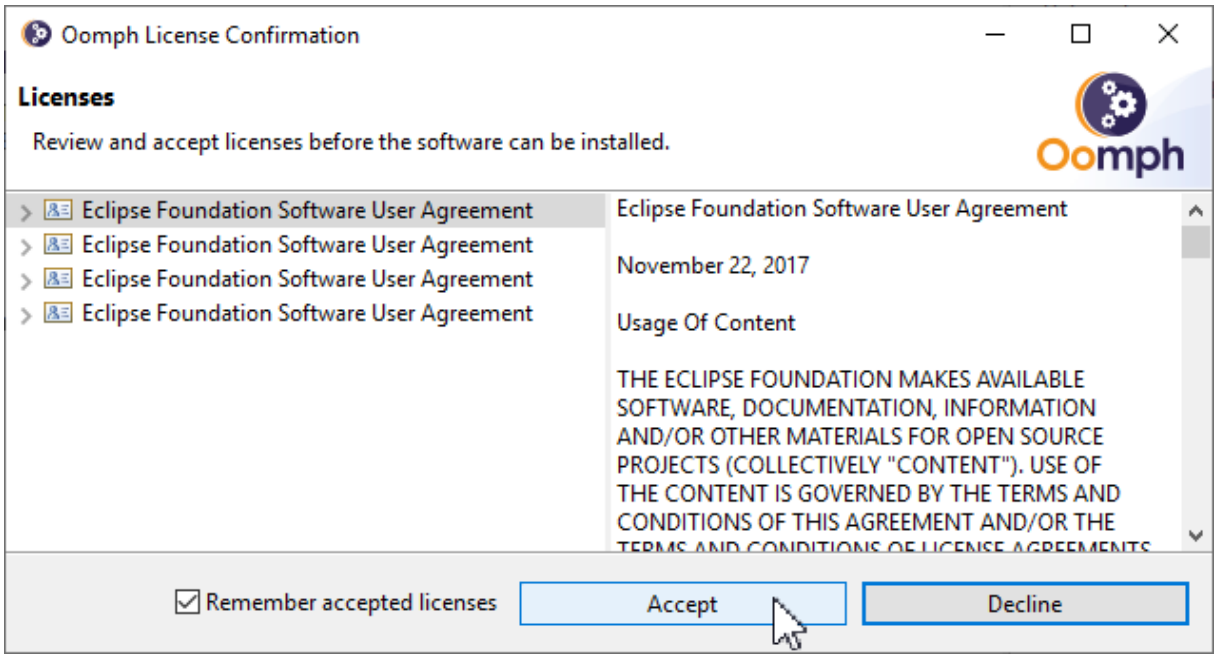


Nachdem ein Ordner ausgewählt, eventuell vorher eingerichtet, wurde, wird der „Install“-Knopf geklickt.

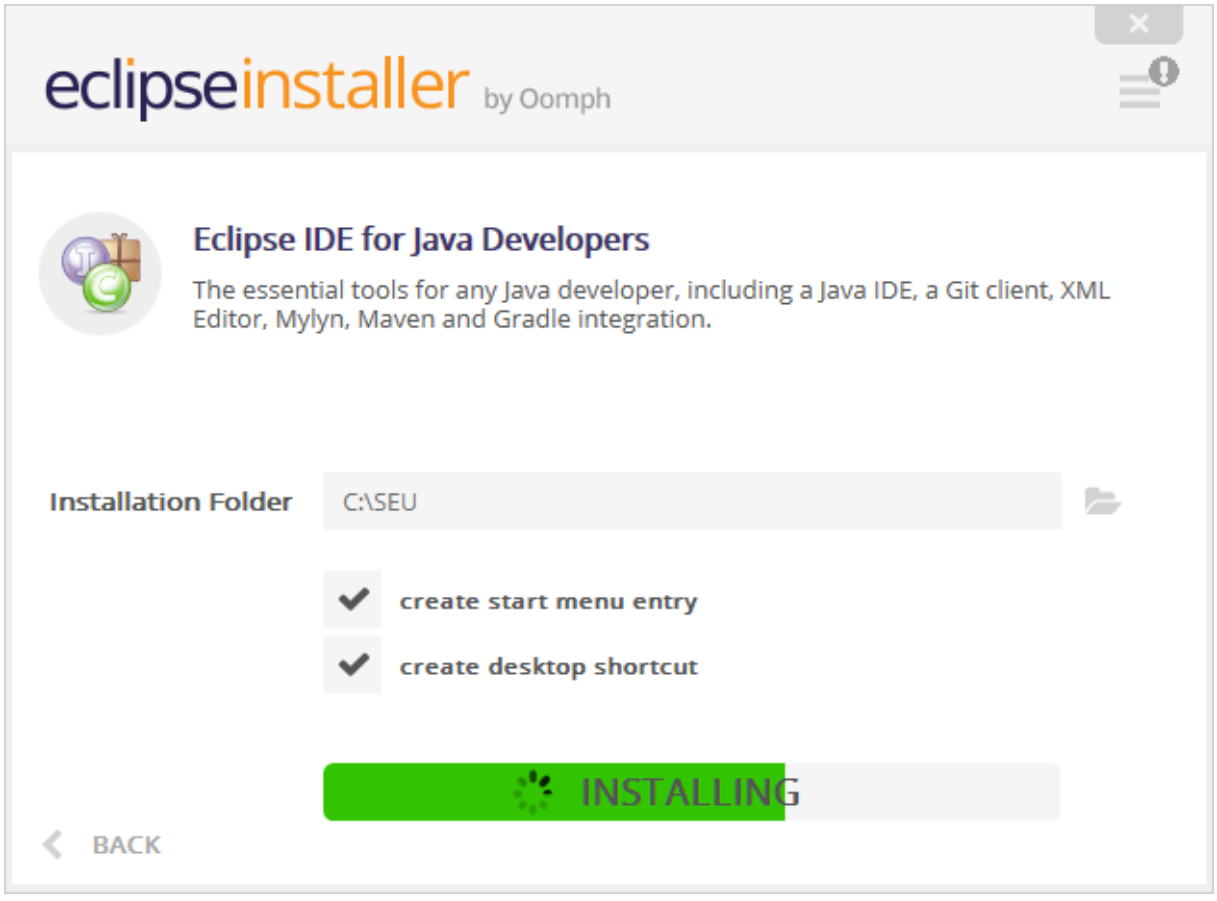


Die Lizenz wird gelesen und mit „Accept Now“ angenommen oder auf die Installation verzichtet.

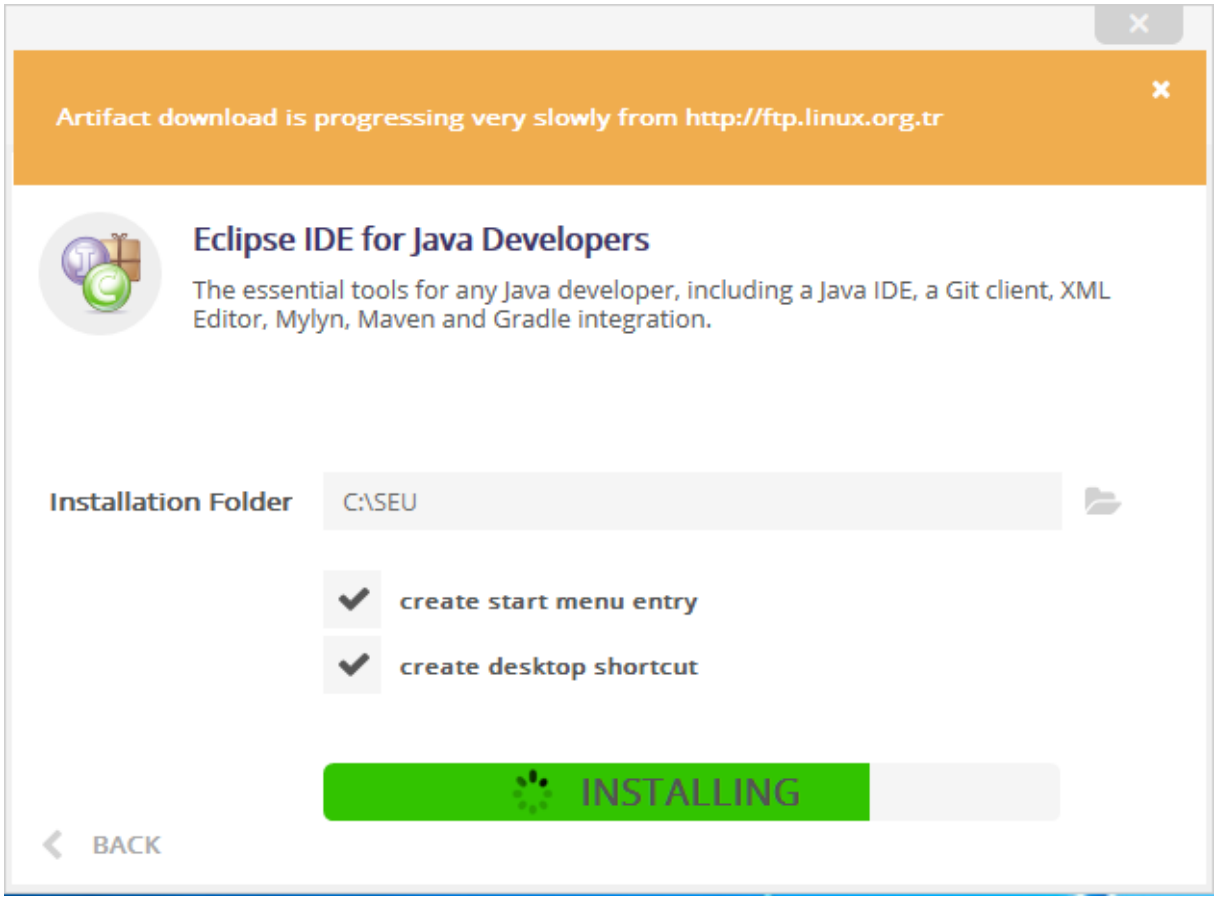
Es folgt eine weitere Lizenzabfrage, da verschiedene Komponenten verschiedene Varianten von Lizenzen nutzen. Bevor der „Accept“-Knopf geklickt wird, ist es deshalb sinnvoll, einen Haken bei „Remember accepted licenses“ zu setzen.



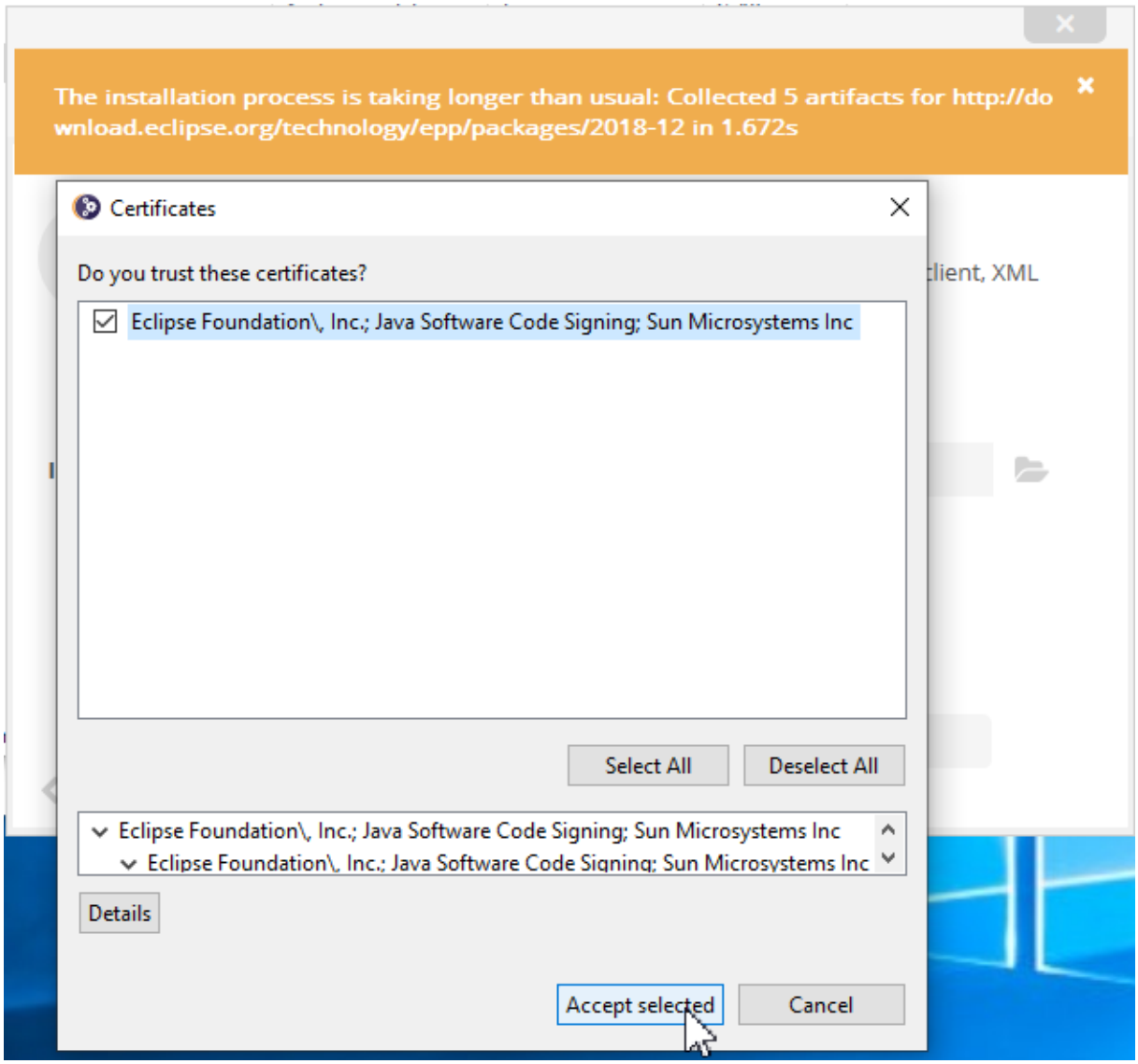
Da Teile aus dem Internet geladen werden, kann die Installation einige Zeit dauern.



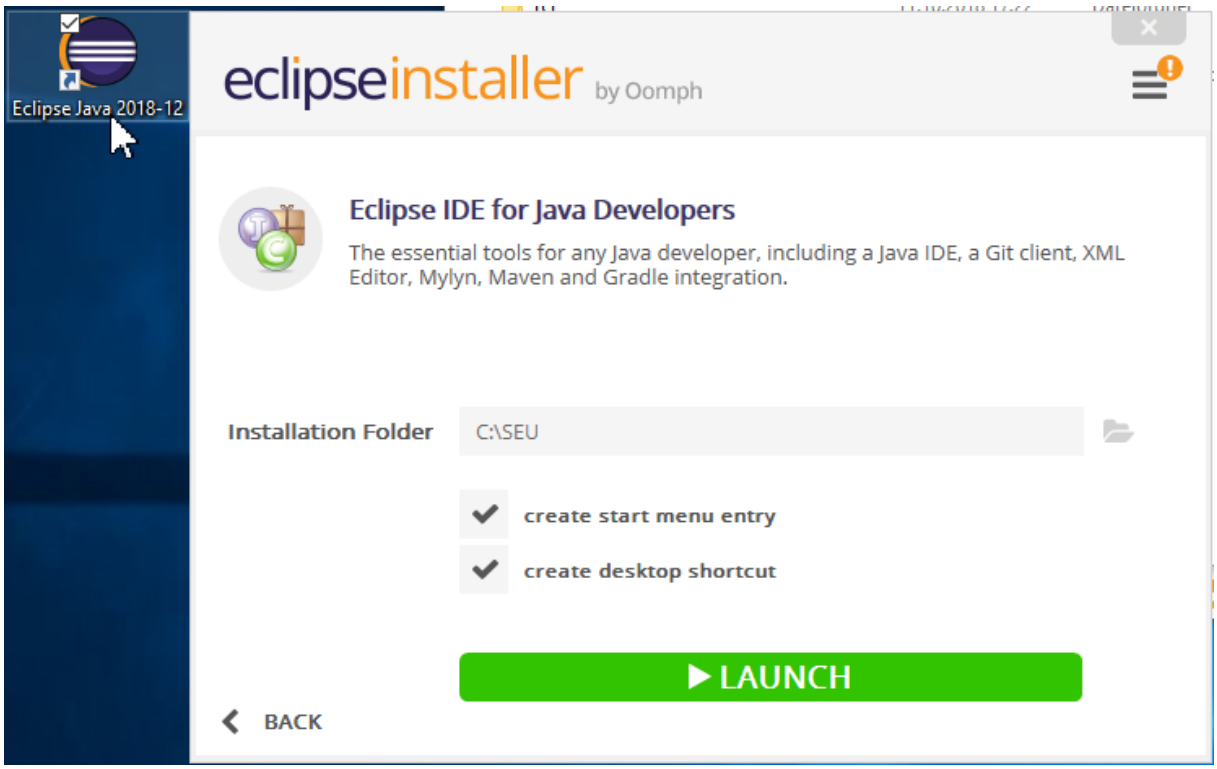
Es sind auch Probleme möglich.



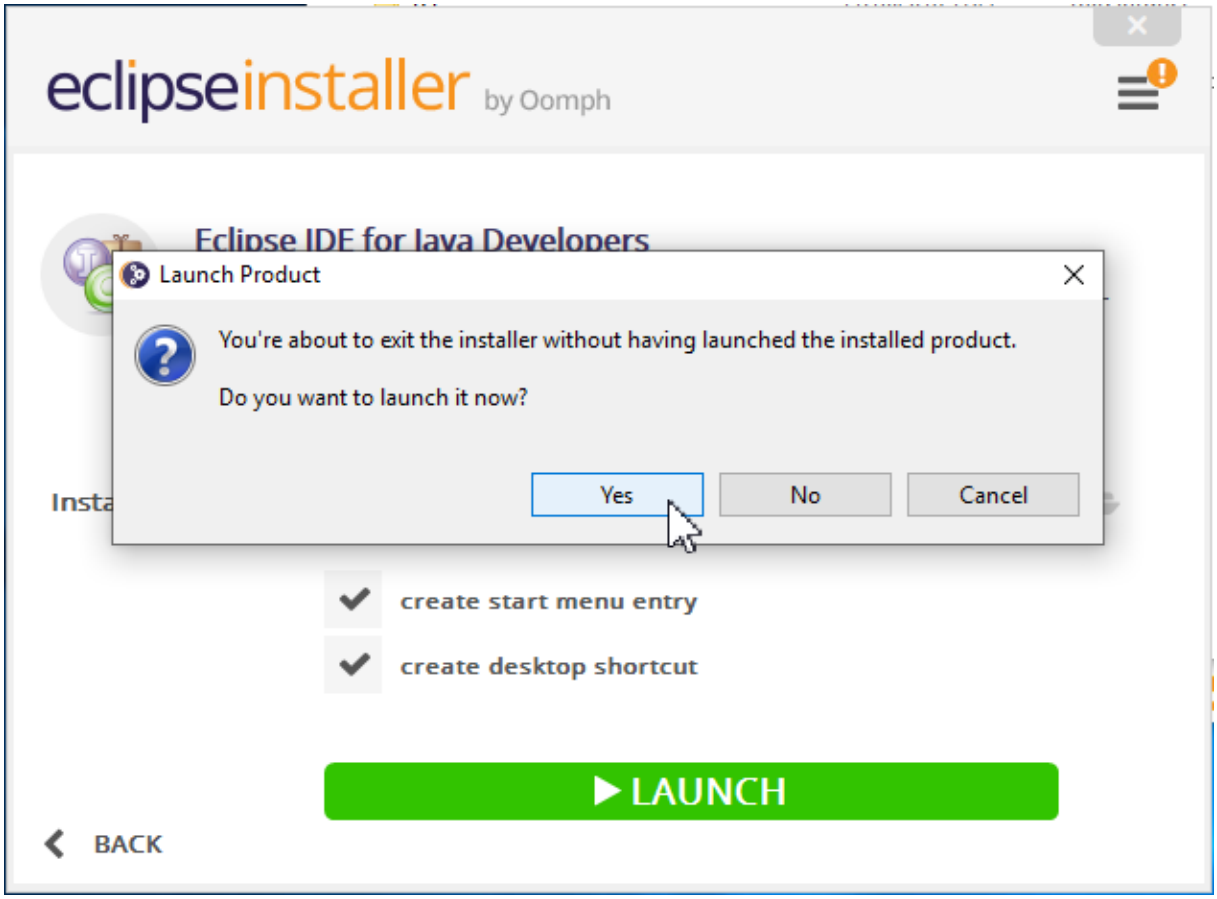
Es muss ein weiterer Haken links-oben gesetzt werden, um dann auf „Accept selected“ geklickt werden.



Nach der Installation kann Eclipse über den „Launch“-Knopf“ oder später z. B. über das Icon gestartet werden.



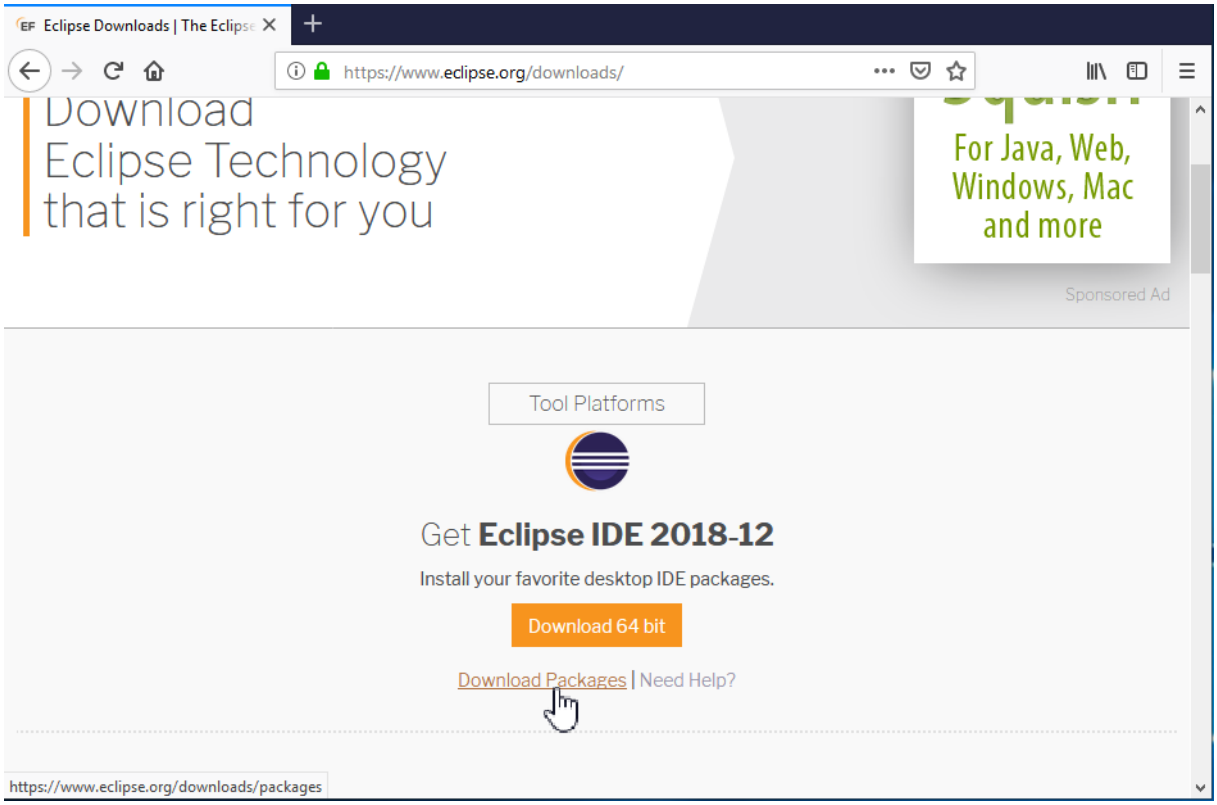
Da die Einrichtung noch nicht abgeschlossen ist, sollte der Launch-Knopf genutzt werden. Beim Abbruch wird ein Hinweis ausgegeben.



Details zur Einrichtung sind in „3.3 Erster Start“ beschrieben.

3.2 Nutzung der Eclipse-Zip-Version

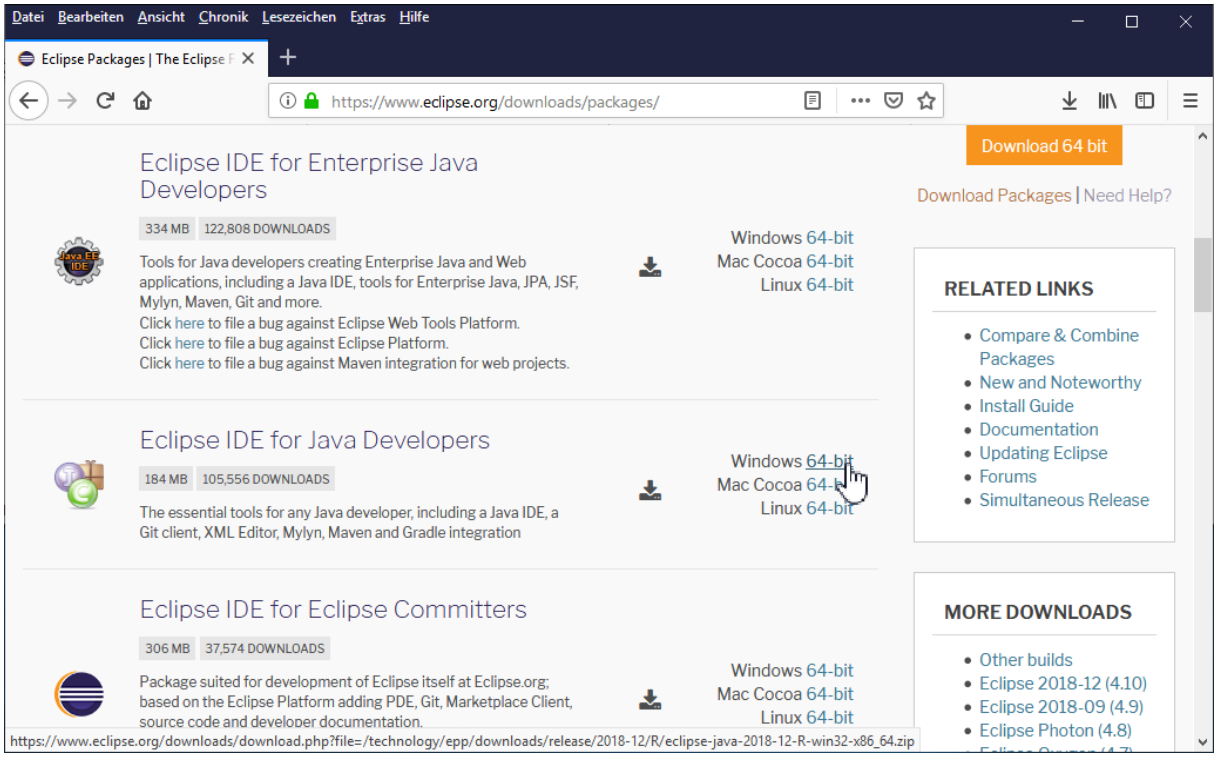
Auf der Download-Seite wird auf den Link „Download Packages“ geklickt.



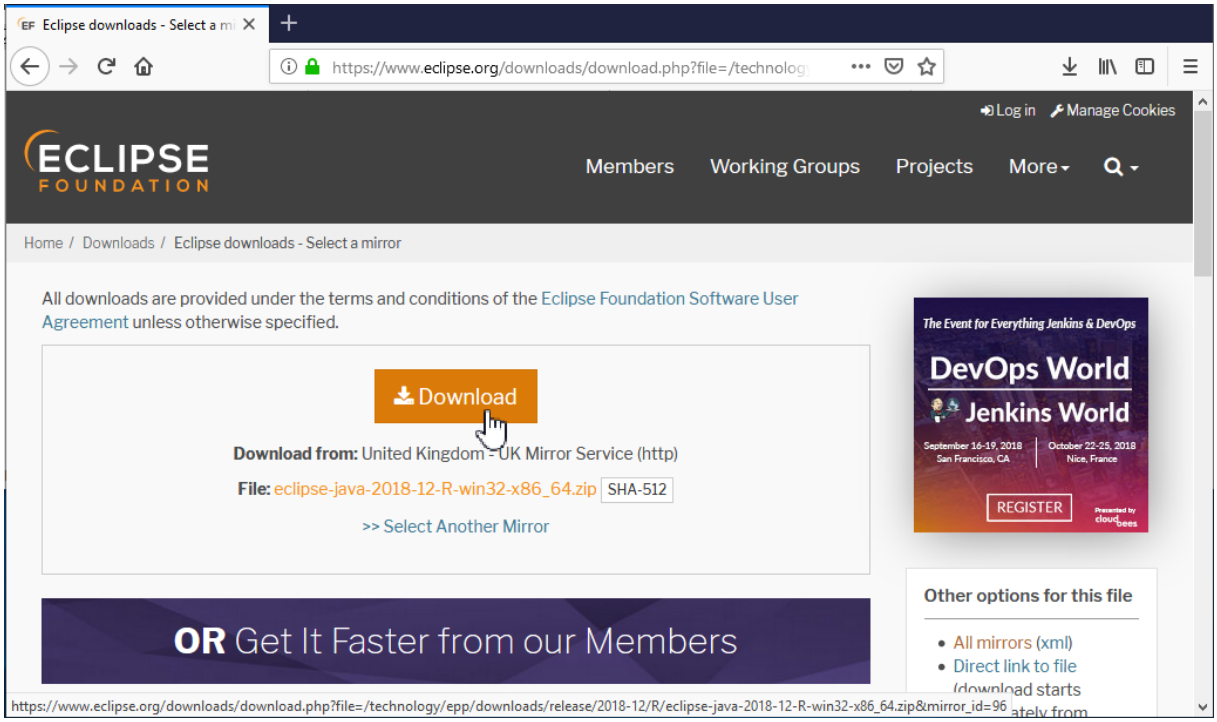
Für die Arbeit mit Java wird die „Eclipse IDE for Java Deleopers“ oder „Eclipse IDE for Java EE Developers“ zur Entwicklung von Web-Applikationen gewählt. Man beachte, dass man das Betriebssystem auswählen kann und dann für Windows weiterhin die passende Variante wählen muss.

Grundsätzlich gilt, dass die Eclipse-Version zur Java-Version passen muss, also 32 Bit zu 32 Bit (x86) oder 64 Bit zu 64 Bit (x64). Die 64-Bit-Variante läuft nur auf einem 64-Bit-Betriebssystem, auf dem die 32-Bit-Variante allerdings auch läuft. Generell werden 32-Bit-Versionen fast nicht mehr unterstützt.

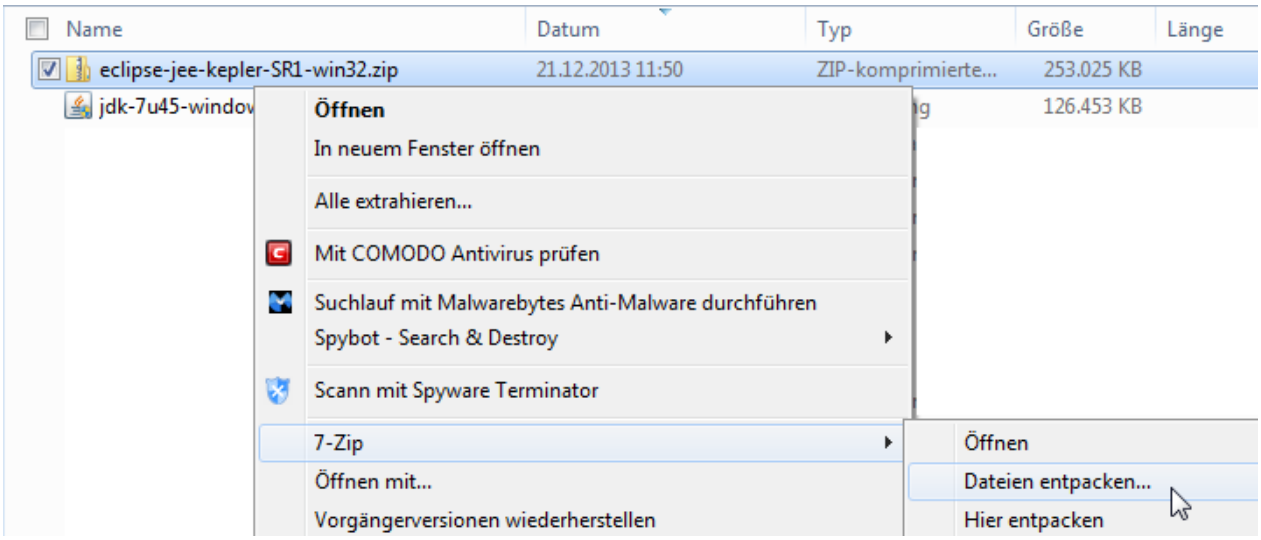
Am rechten Rand steht ein Kasten „More Downloads“ über dessen Links ältere Eclipse-Versionen erhältlich sind, so die letzte (?) 32-Bit-Version Eclipse 2018-12 (4.10).



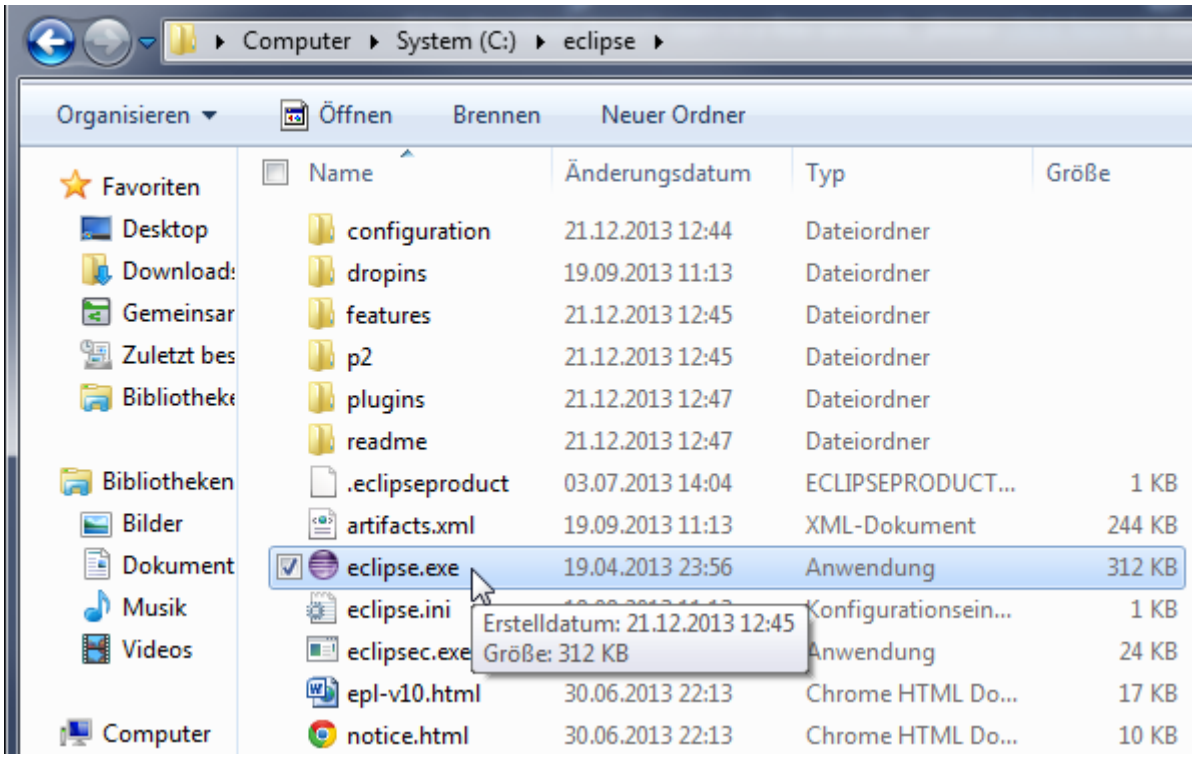
Der Download kann dann mit einem Klick auf den Knopf gestartet werden.



Die heruntergeladene zip-Datei kann in einem beliebigen Verzeichnis, im folgenden C:\ entpackt werden, was die eigentliche Installation von Eclipse abschließt. Man beachte, dass das zip-Verzeichnis bereits den Ordner eclipse enthält und man so direkt nach C:\ entpacken muss



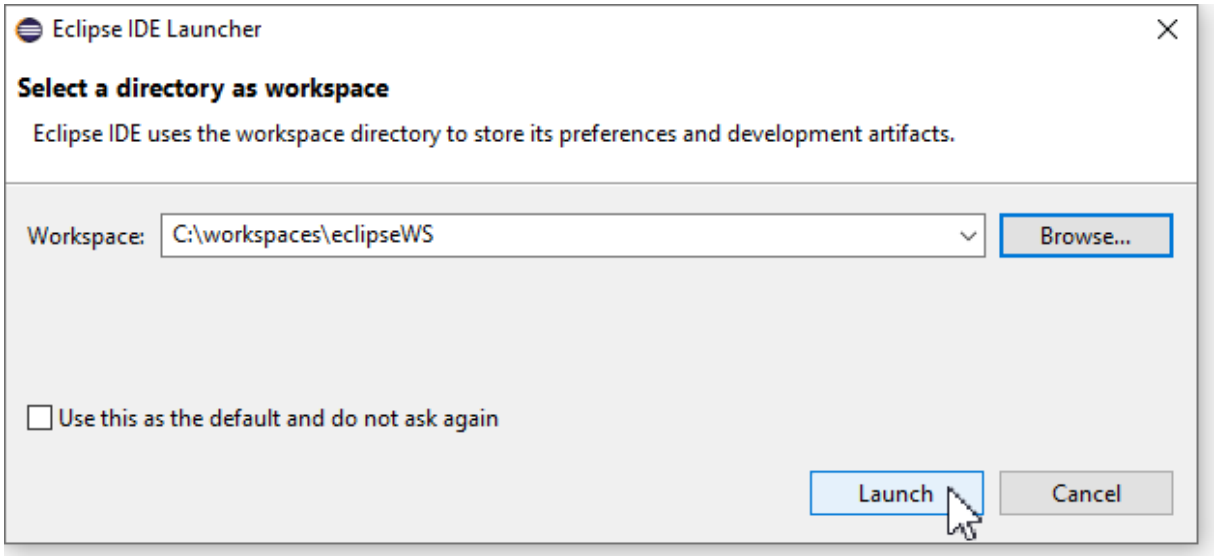
Der Programmstart erfolgt durch einen Doppelklick auf eclipse.exe im Eclipse-Ordner. Natürlich kann man sich dafür auch eine Verknüpfung auf der Oberfläche anlegen. Beim ersten Start können sich die Firewall oder weitere Systemprogramme melden. Man sollte auf eigene Verantwortung Eclipse die geforderten Rechte einräumen.



3.3 Erster Start

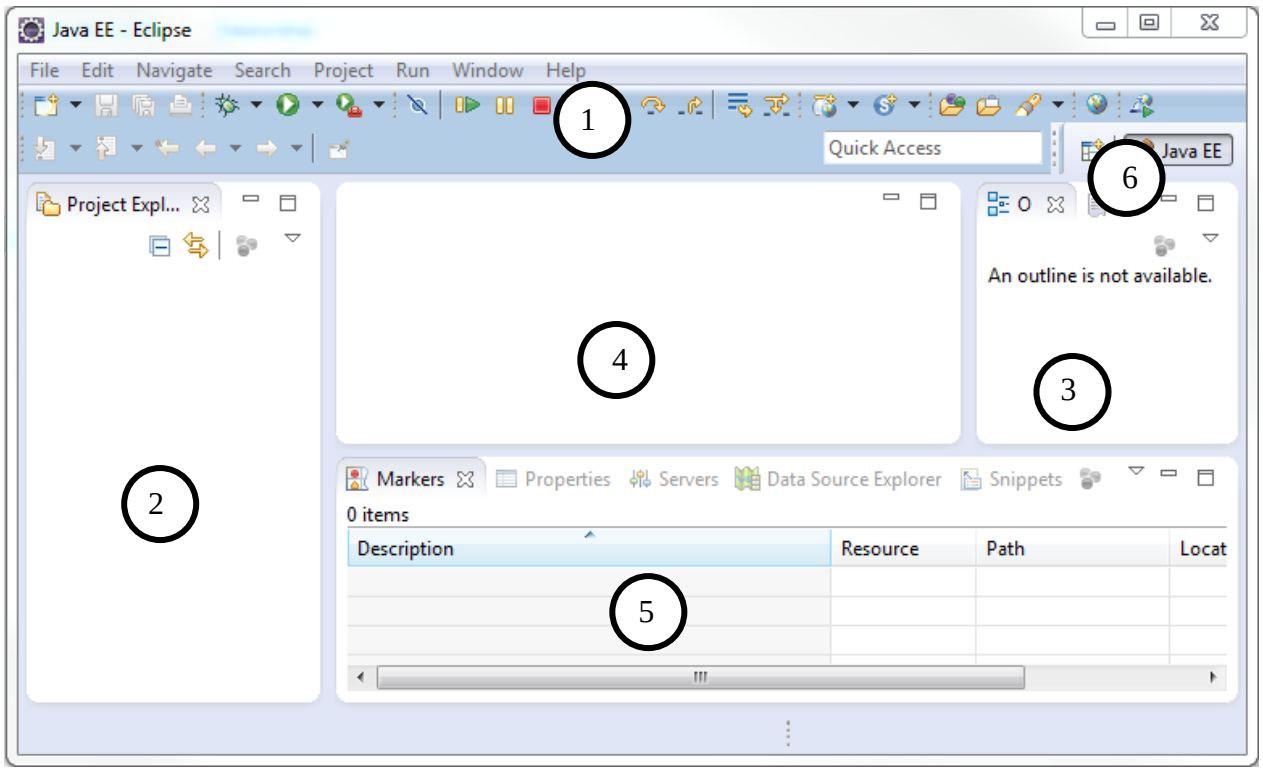
Wie für Java-Programme typisch, dauert der Programmstart etwas (zur Erinnerung: der Byte-Code muss zunächst in den Maschinencode der Maschine übersetzt werden, wird der Code dann erneut benötigt, liegt er bereits übersetzt vor, weshalb Java-Programme dann genau so schnell wie andere Programme sind).

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 nur in Ausnahmefällen 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 ansteuern 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 abhängig von der installierten Version 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.

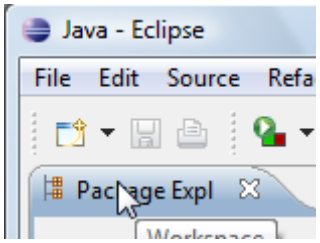
3: Hier findet eine Zusammenfassung der Informationen über den aktuell ausgewählten Projektbaustein statt. Dies kann z. B. eine Liste aller Variablen und Methoden einer Klasse sein.

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

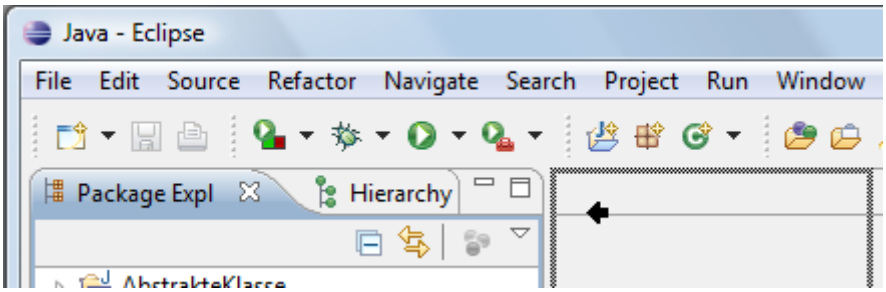
5: Verteilt auf die verschiedenen Reiter werden hier typischerweise Ausgaben zu aktuellen Vorgängen angezeigt. Dies können z. B. Meldungen des Kompilierungsprozesses, Listen mit Syntaxfehlern oder Eingabeaufforderungen von Nutzungsdialogen sein.

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

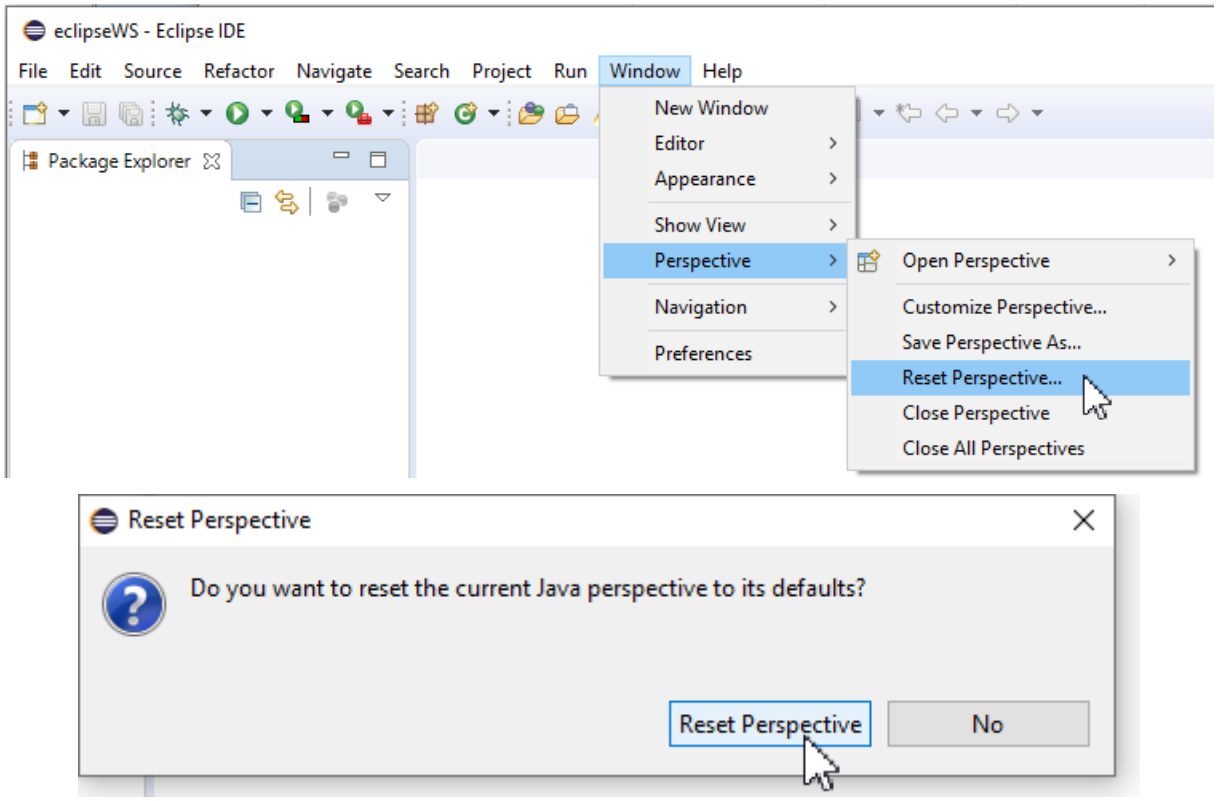
Man beachte, dass die Fenster frei verschiebbar und an verschiedenen Stellen andockbar sind. Zum Verschieben wird auf die Mitte eines Reiters mit der linken Maustaste gedrückt und dies dann festgehalten.



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 > Perspective > Reset Perspective ...“ den Ursprungszustand wiederherstellen.



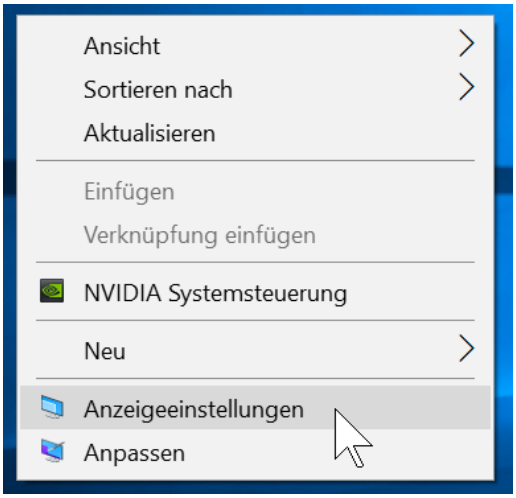
3.4 Nervige Probleme mit der Bildschirmauflösung

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).

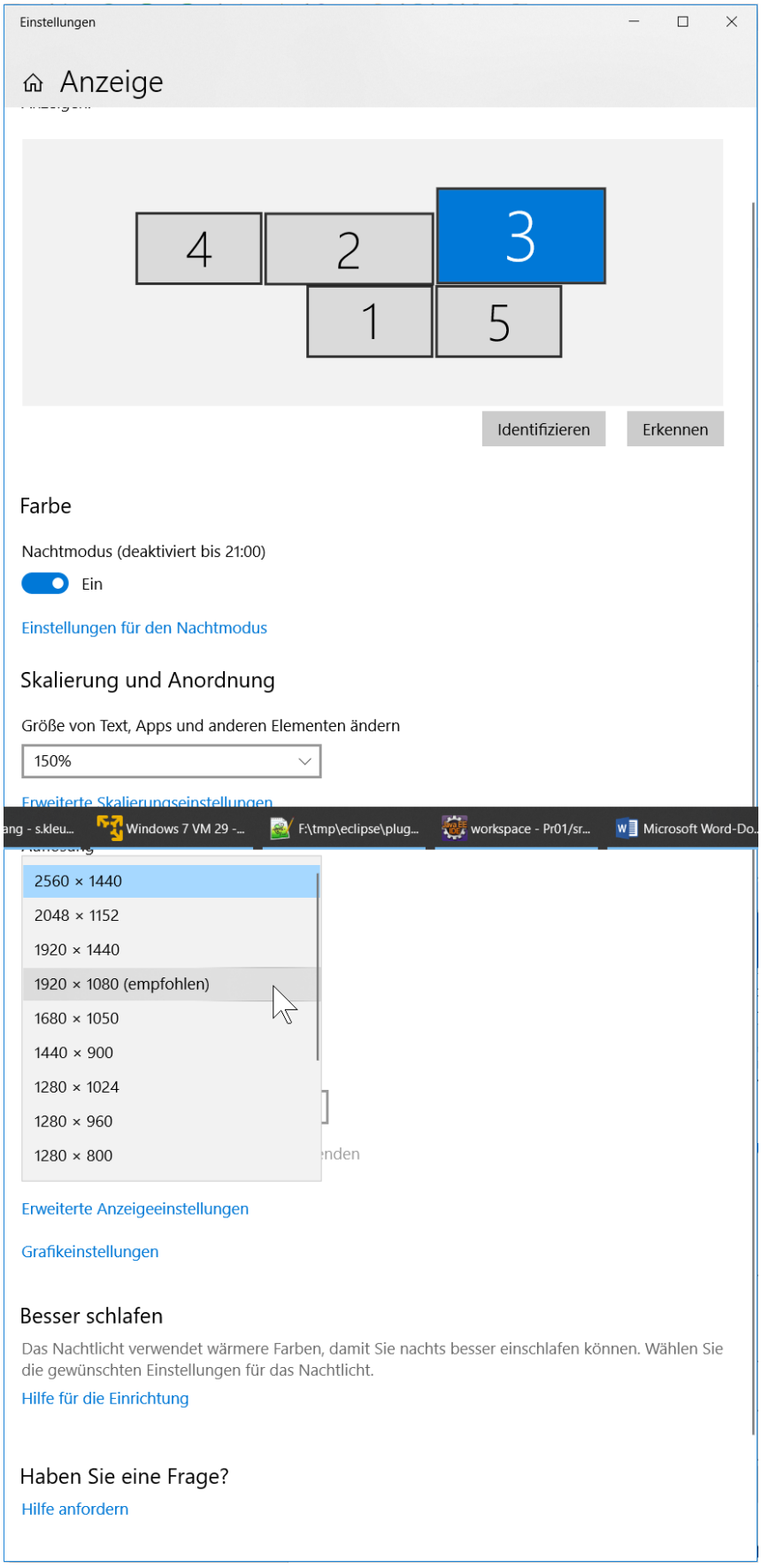
Es gibt verschiedene Lösungsansätze, damit Eclipse sich Bildschirme mit unterschiedlichen Auflösungen, gerade sehr hohen Auflösungen anpasst. Alle Lösungen haben Vor- und Nachteile.

3.4.1 Monitorauflösung anpassen

Pragmatisch kann man sich fragen, ob die hohe Auflösung wirklich benötigt wird. Wenn nicht, kann die Auflösung des verringert werden. Ein Weg ist ein Rechtsklick auf freier Monitorfläche und die Auswahl von „Anzeigeeinstellungen“.

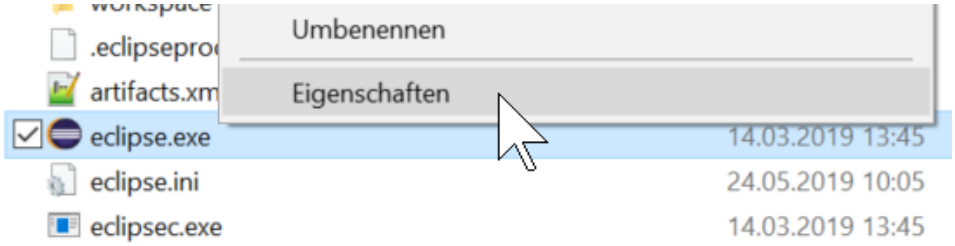


Hier kann für jeden Monitor die Auflösung eingestellt und wenn gewünscht die Größe des Texts skaliert werden.

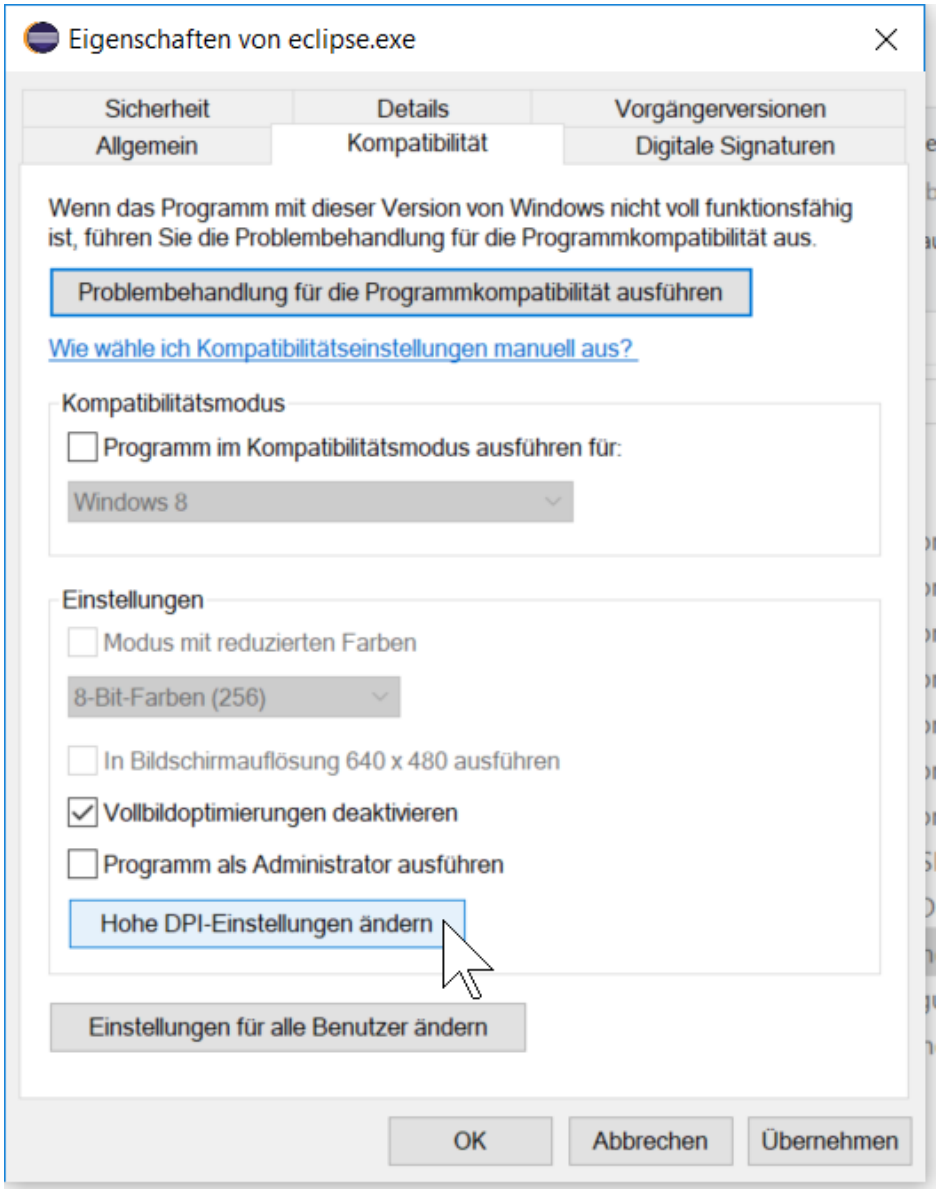

Einstellungen
Anzeige
4
2
3
1
5
Identifizieren
Erkennen
Farbe
Nachtmodus (deaktiviert bis 21:00)
Ein
Einstellungen für den Nachtmodus
Skalierung und Anordnung
Größe von Text, Apps und anderen Elementen ändern
150%
Erweiterte Skalierungseinstellungen
Windows 7 VM 29 -...
F:\tmp\eclipse\plug...
workspace - Pr01/sr...
Microsoft Word-Do...
2560 × 1440
2048 × 1152
1920 × 1440
1920 × 1080 (empfohlen)
1680 × 1050
1440 × 900
1280 × 1024
1280 × 960
1280 × 800
Erweiterte Anzeigeeinstellungen
Grafikeinstellungen
Besser schlafen
Das Nachtlicht verwendet wärmere Farben, damit Sie nachts besser einschlafen können. Wählen Sie die gewünschten Einstellungen für das Nachtlicht.
Hilfe für die Einrichtung
Haben Sie eine Frage?
Hilfe anfordern

3.4.2 DPI-Verhalten für Eclipse ändern

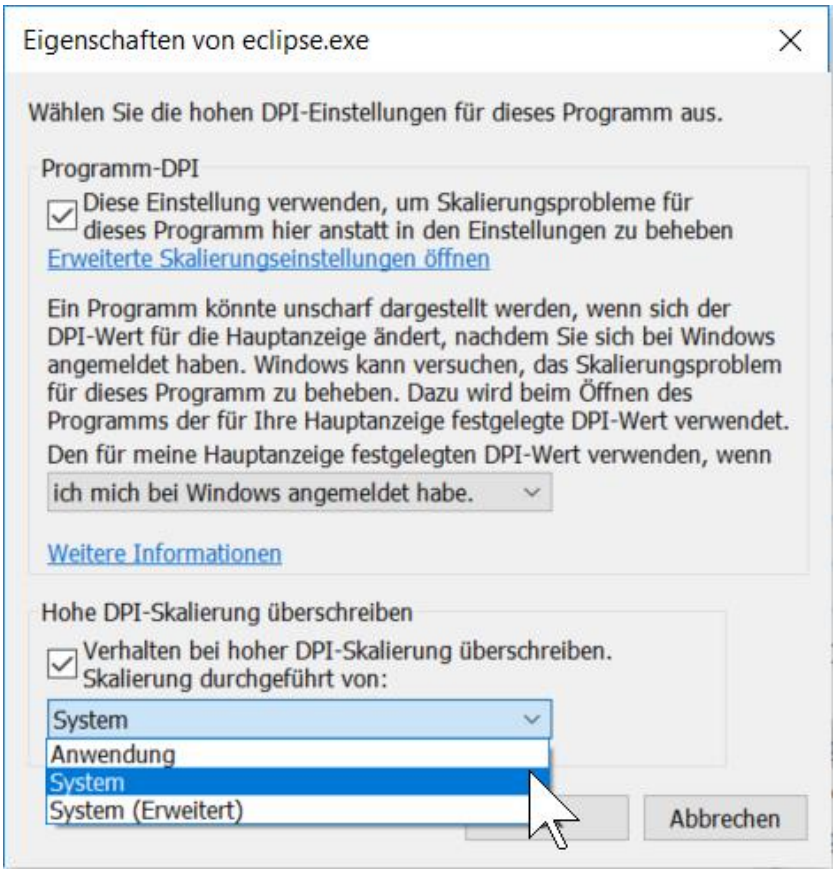
Eine weitere, öfter genutzte, Alternative stellt genau für eclipse.exe ein, dass das Betriebssystem und nicht die Software selbst zur Skalierung genutzt wird. Dazu wird ein Rechtsklick auf eclipse.exe gemacht und „Eigenschaften“ ausgewählt.



Es wird der Reiter „Kompatibilität gewählt und auf „Hohe DPI-Einstellungen ändern“ geklickt.

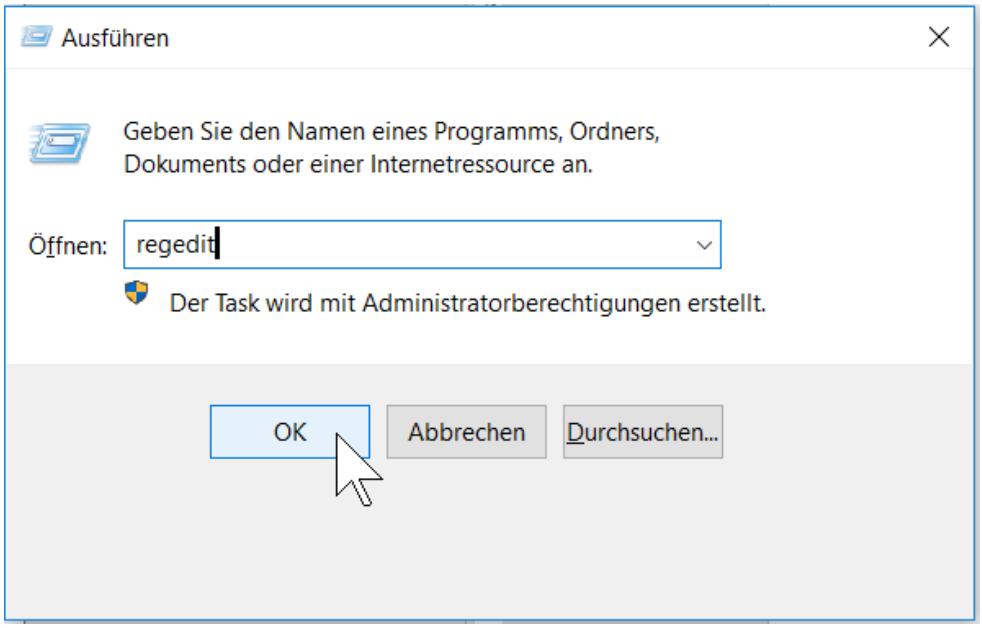


Es wird oben und unten jeweils ein Haken gesetzt und in der unteren Auswahlbox „System“ ausgewählt. Die Änderungen werden dann mit zweimal „OK“ übernommen. Nachteil ist, dass die Darstellung leicht verwaschen aussieht.

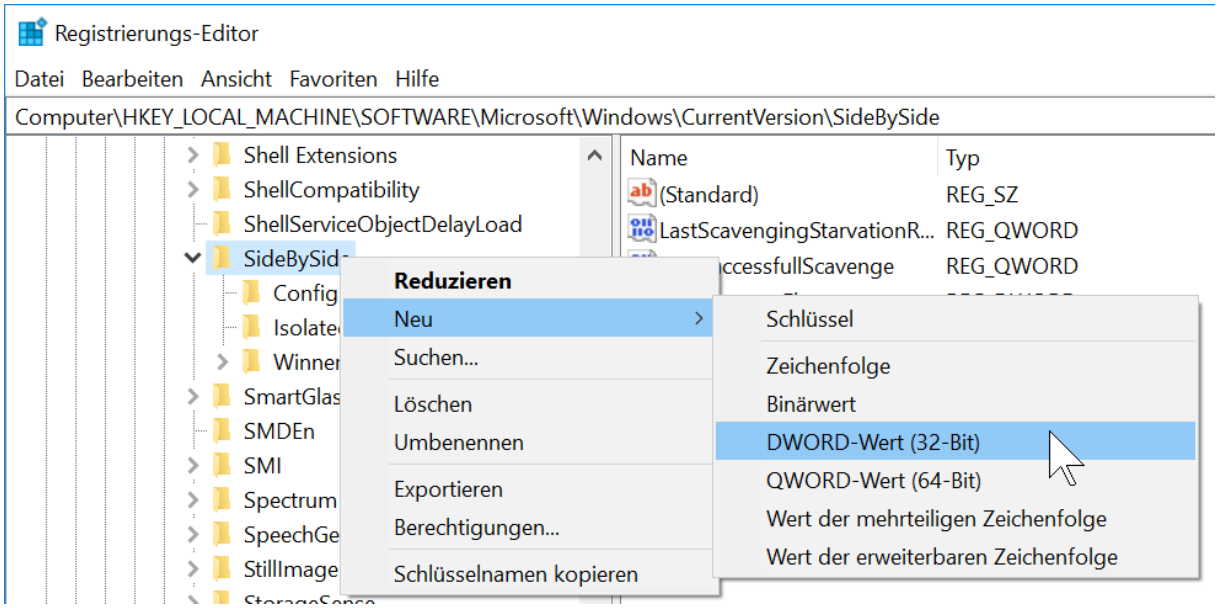


3.4.3 DPI-Verhalten für mehrere Programme ändern

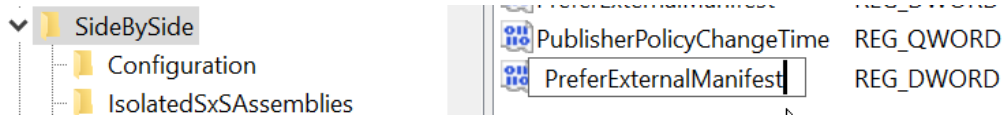
Soll für mehrere Programme eingestellt werden, dass das System und nicht die Software selbst die DPI-Einstellungen vornimmt, muss dies zunächst in der Windows-Registry festgelegt werden. Dazu wird die Windows-Taste zusammen mit „R“ gedrückt, danach „regedit“ eingegeben und „OK“ gedrückt.



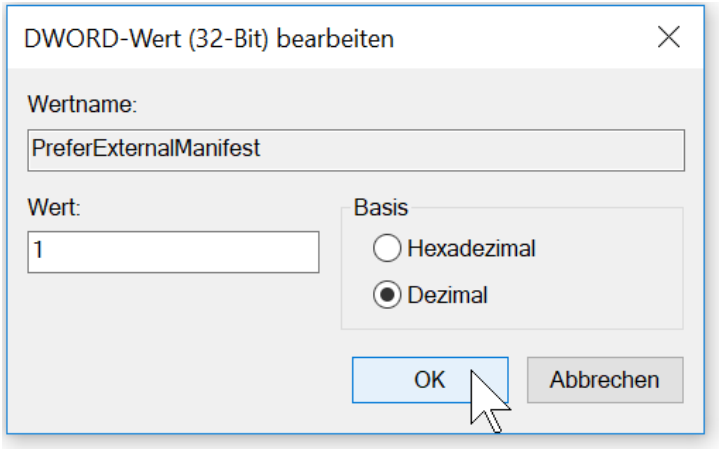
Links wird Schrittweise zu „HKEY_LOCAL_MACHINE > SOFTWARE > Microsoft > Windows > CurrentVersion > SideBySide“ manövriert. Ein Rechtsklick auf „SideBySide“ gemacht und „Neu > DWORD-Wert (32-Bit)“ ausgewählt.



Es wird als Name „PreferExternalManifest“ eingetragen und Return gedrückt.



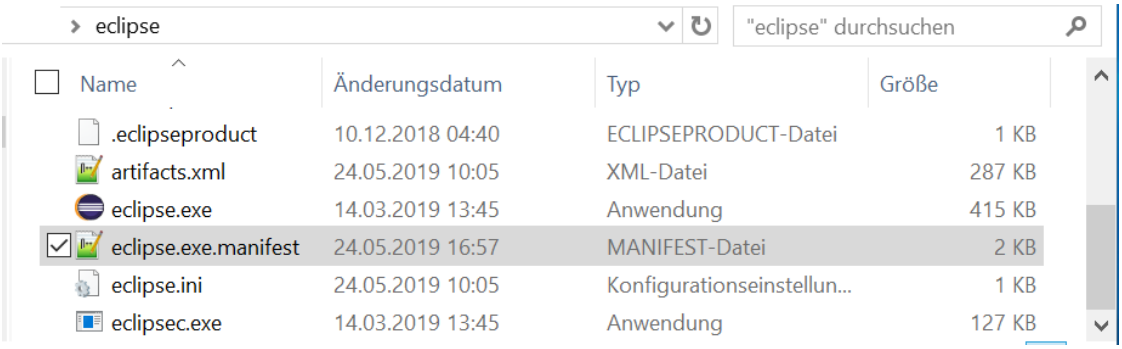
Danach wird ein Rechtklick auf dem neuen Eintrag „PreferExternalManifest“ gemacht und „Ändern…“ ausgewählt. Der Wert wird auf „1“ und die „Basis“ auf „Dezimal“ gesetzt. Alle Änderungen werden mit „OK“ übernommen und der Registrierungs-Editor geschlossen.



Für jedes Programm, das vom System aus an die Auflösung angepasst werden soll, muss für die jeweilige Datei prog.exe eine Datei prog.exe.manifest im gleichen Ordner angelegt werden. Im konkreten Fall wird die Datei eclipse.exe.manifest ergänzt, was auch im jeweiligen Ordner für java.exe und javaw.exe sinnvoll sein kann. Die Dateien haben den folgenden Inhalt.

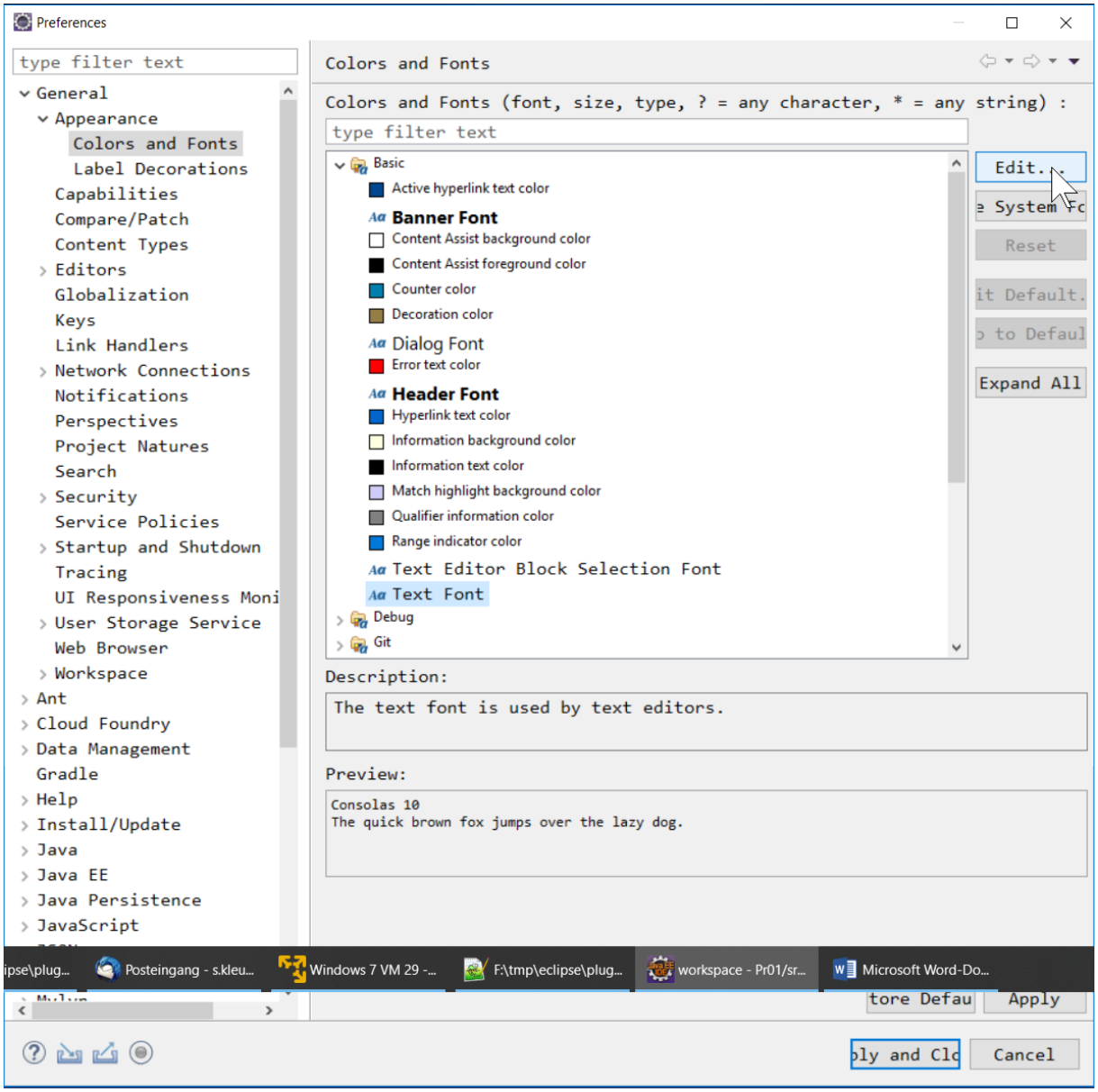
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0" xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">

<dependency>
 <dependentAssembly>
  <assemblyIdentity
   type="win32"
   name="Microsoft.Windows.Common-Controls"
   version="6.0.0.0" processorArchitecture="*"
   publicKeyToken="6595b64144ccf1df"
   language="*">
  </assemblyIdentity>
 </dependentAssembly>
</dependency>

<dependency>
 <dependentAssembly>
  <assemblyIdentity
   type="win32"
   name="Microsoft.VC90.CRT"
   version="9.0.21022.8"
   processorArchitecture="amd64"
   publicKeyToken="1fc8b3b9a1e18e3b">
  </assemblyIdentity>
 </dependentAssembly>
</dependency>
```

```
<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
 <security>
  <requestedPrivileges>
   <requestedExecutionLevel
    level="asInvoker"
    uiAccess="false"/>
  </requestedPrivileges>
 </security>
</trustInfo>

<asmv3:application>
 <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
  <ms_windowsSettings:dpiAware
xmlns:ms_windowsSettings="http://schemas.microsoft.com/SMI/2005/WindowsSettings">false</ms_windowsSettings:dpiAware>
 </asmv3:windowsSettings>
</asmv3:application>

</assembly>
```



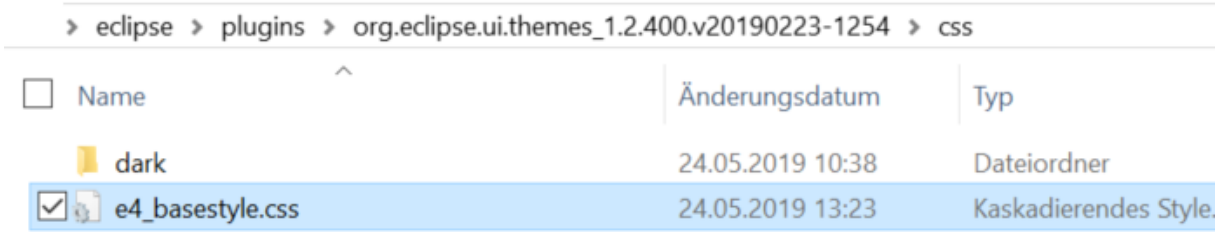
3.4.4 Fonts in Eclipse einstellen

In Eclipse selbst kann auf viele Fonts für unterschiedliche Teilaufgaben von Eclipse zugegriffen werden. Die Einstellungen sind unter „Window > Preferences“ und dann „General > Apperance > Colors an Fonts“ zu finden. Zunächst sollten die Einstellungen unter Basic bearbeitet werden.



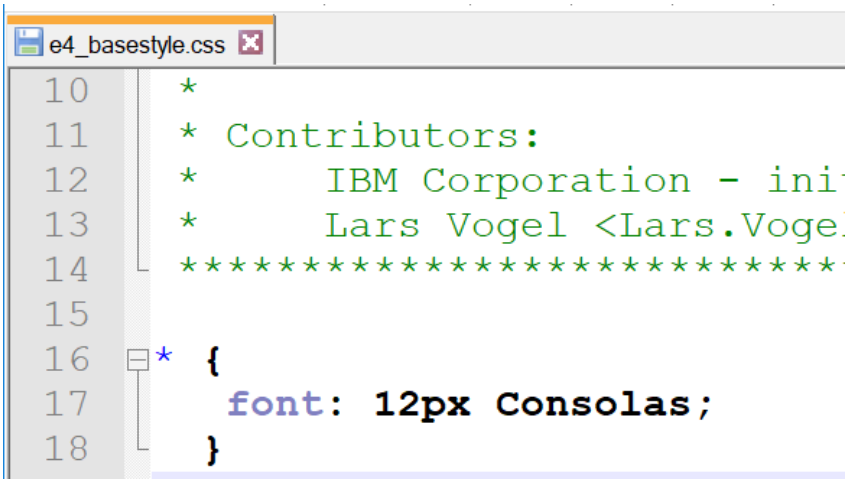
3.4.5 Font-Größe per CSS ändern

Eine weitere drastische Variante besteht darin, die GUI-Einstellungen von Eclipse zu ändern, die mit CSS eingestellt werden. Die zugehörige Datei befindet sich in einem Unterordner, genauer im Ordner eclipse\plugins\org.eclipse.ui.themes_1.2.400.v20190223-1254\css. Die zu ändernde Datei ist e4_basestyle.css.



Es wird am Anfang folgender Eintrag ergänzt, der alle Texte auf den angegebenen Font in der angegebenen Größe setzt. An einigen Stellen nutzt Eclipse den System-Font, die in oder um Eclipse herum nicht änderbar ist. Änderungen sind da nur über das Betriebssystem möglich und betreffen den gesamten Rechner.

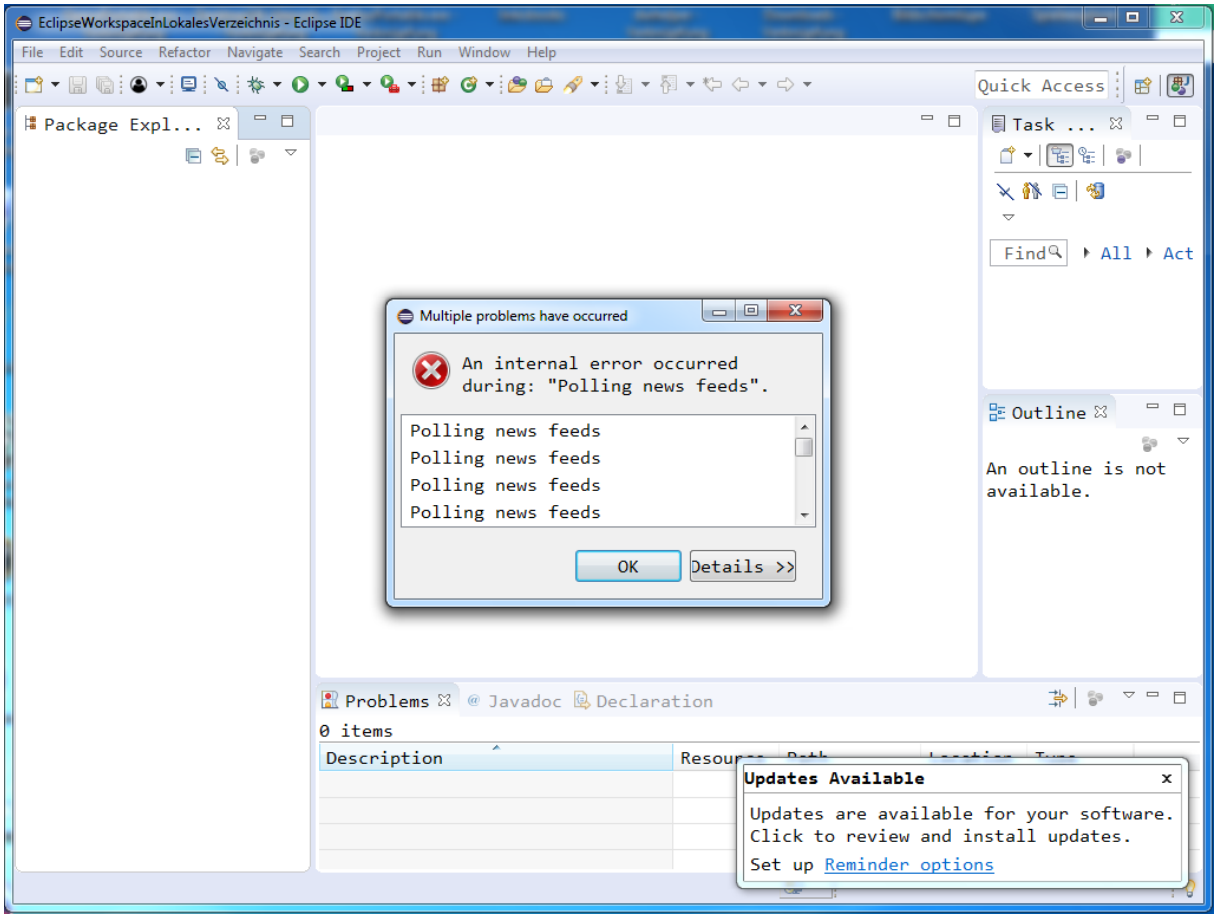
```
* {
   font: 12px Consolas;
  }
```



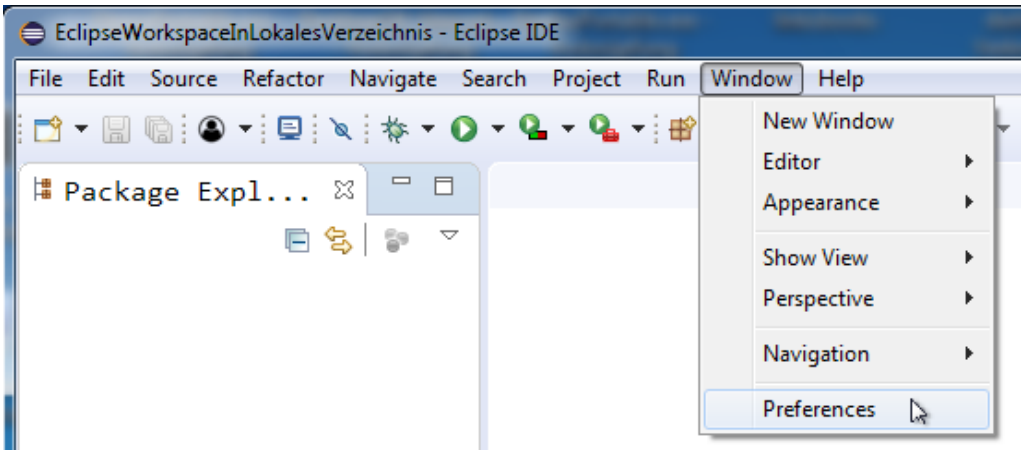
Die vorher beschriebenen Änderungen innerhalb von Eclipse werden davon überschrieben.

3.5 Fehlermeldung: Polling news feeds

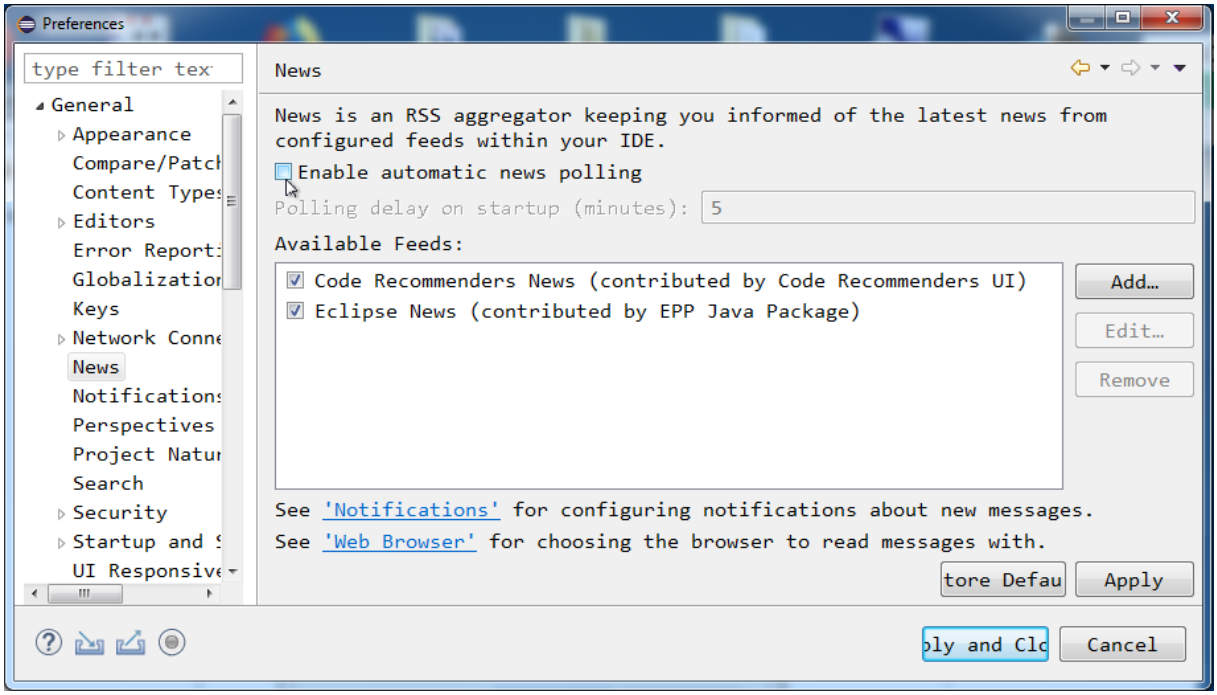
Eclipse hat ein interessantes Benachrichtigungssystem, dass allerdings nicht mit allen Komponenten stabil zusammenläuft, so dass es in einigen Eclipse-Varianten zu folgender Fehlermeldung in der Mitte kommen kann.



Der einzige Workaround dies zu verhindern ist es es, das Benachrichtigungssystem abzuschalten. Dazu wird zunächst auf „Windows > Preferences“ geklickt.

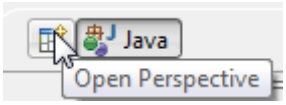


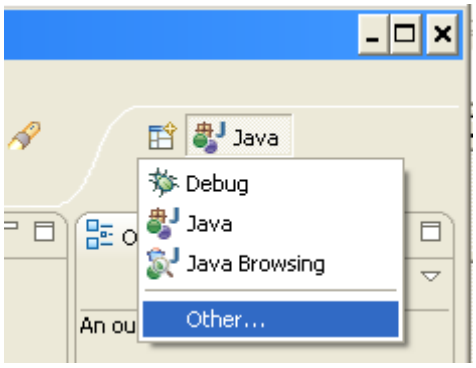
Zunächst wird „General > News“ ausgewählt und dann der Haken bei „Enable automatic news polling“ weggenommen. Die Änderung wird mit „Apply and Close bestätigt.


Preferences
type filter tex
General
Appearance
Compare/Patcl
Content Type
Editors
Error Report
Globalizatio
Keys
Network Conn
News
Notification
Perspectives
Project Natu
Search
Security
Startup and
UI Responsiv
News
News is an RSS aggregator keeping you informed of the latest news from configured feeds within your IDE.
Enable automatic news polling
Polling delay on startup (minutes): 5
Available Feeds:
Code Recommenders News (contributed by Code Recommenders UI)
Eclipse News (contributed by EPP Java Package)
Add...
Edit...
Remove
See 'Notifications' for configuring notifications about new messages.
See 'Web Browser' for choosing the browser to read messages with.
tore Defau
Apply
ply and Clo
Cancel

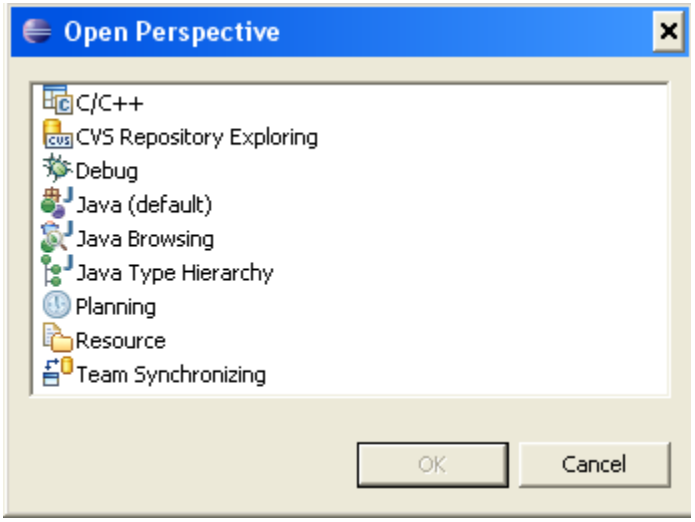
4 Einstellungen und erste Einrichtungen in Eclipse (4.11.0)

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

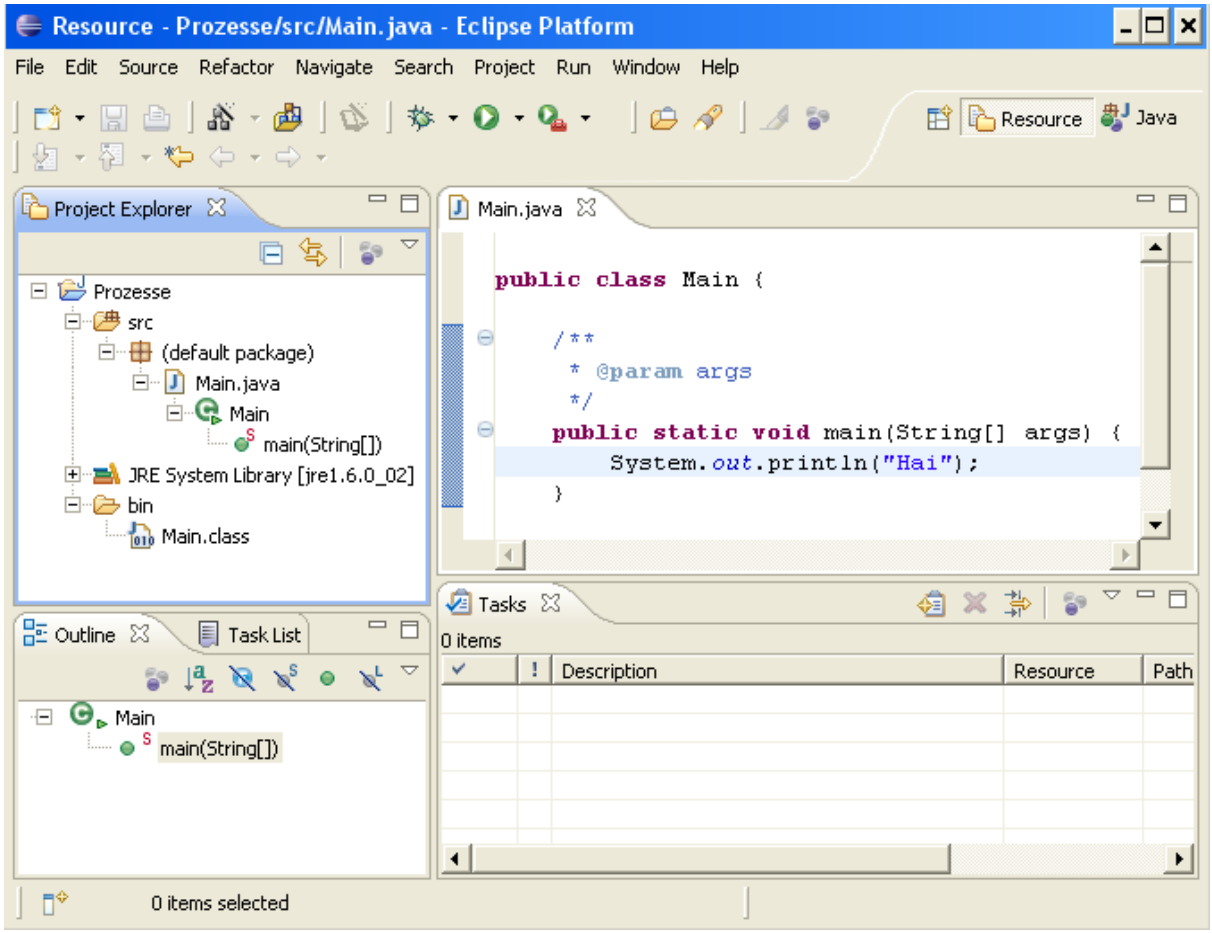




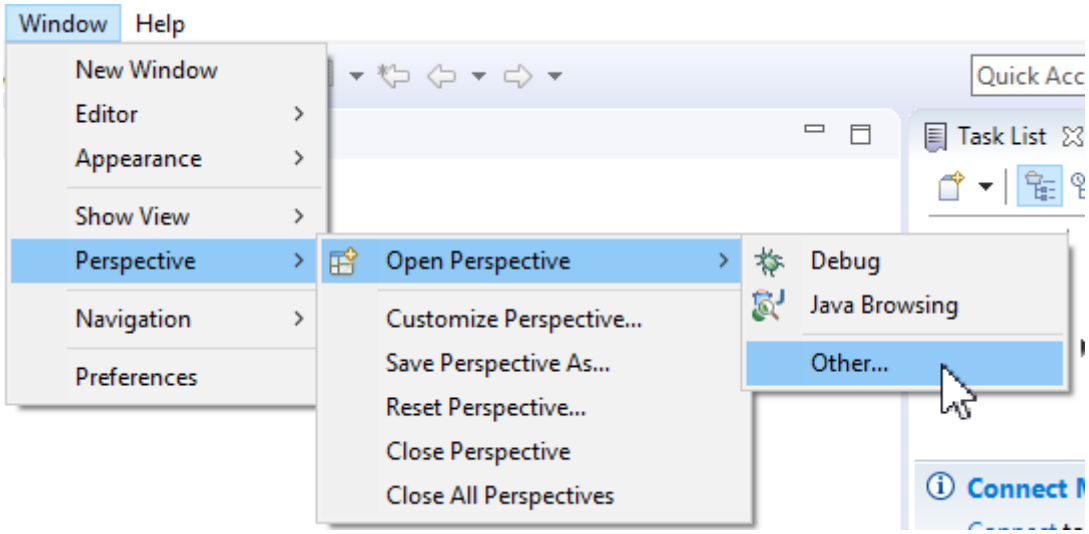
Unter „Other“ erhält man eine vollständige Übersicht über die vorhandenen Sichten. Diese Übersicht hängt von den vorhandenen Plugins ab und kann wie folgt aussehen.



Interessant ist u. a. die Resource-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.

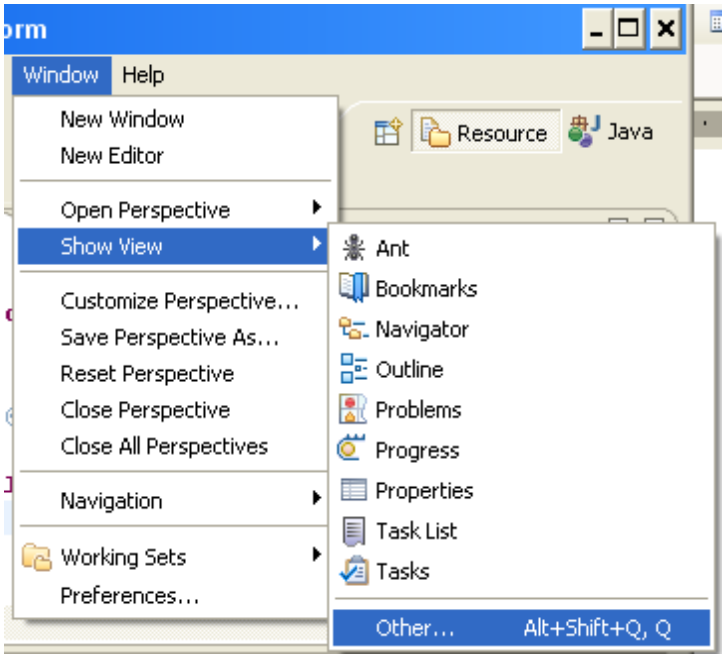


Die Sicht kann auch unter „Window>Open Perspective“ eingestellt werden.

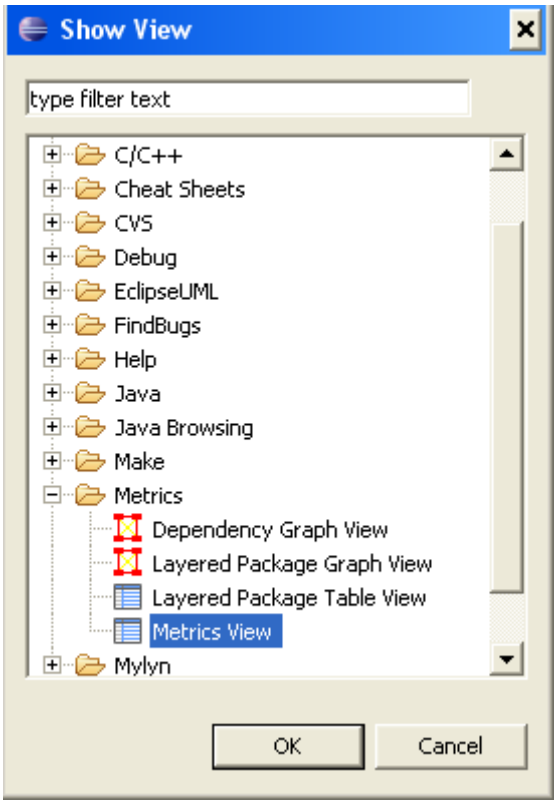


Das obige Menü bietet ebenfalls den Punkt „Reset Perspective“, mit man alle in einer Sicht vorgenommenen Änderungen, z. B. aus Versehen geschlossene Fenster, wieder rückgängig machen kann. Weiterhin besteht die Möglichkeit, individuell eingestellte Sichten zu speichern, so dass auch diese immer wieder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

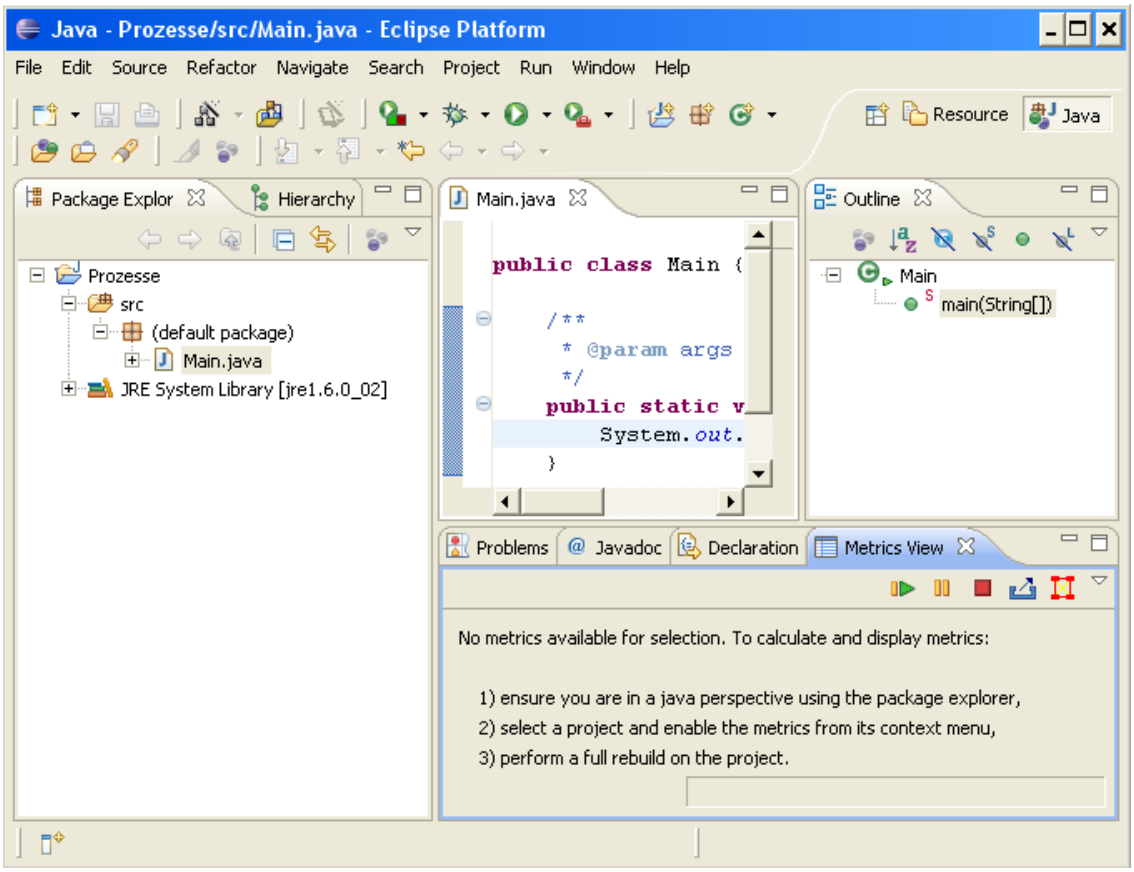
Häufig bieten Plugins zusätzliche Fenster. Diese können unter „Show View“ in der vorherigen Sicht ausgewählt werden.



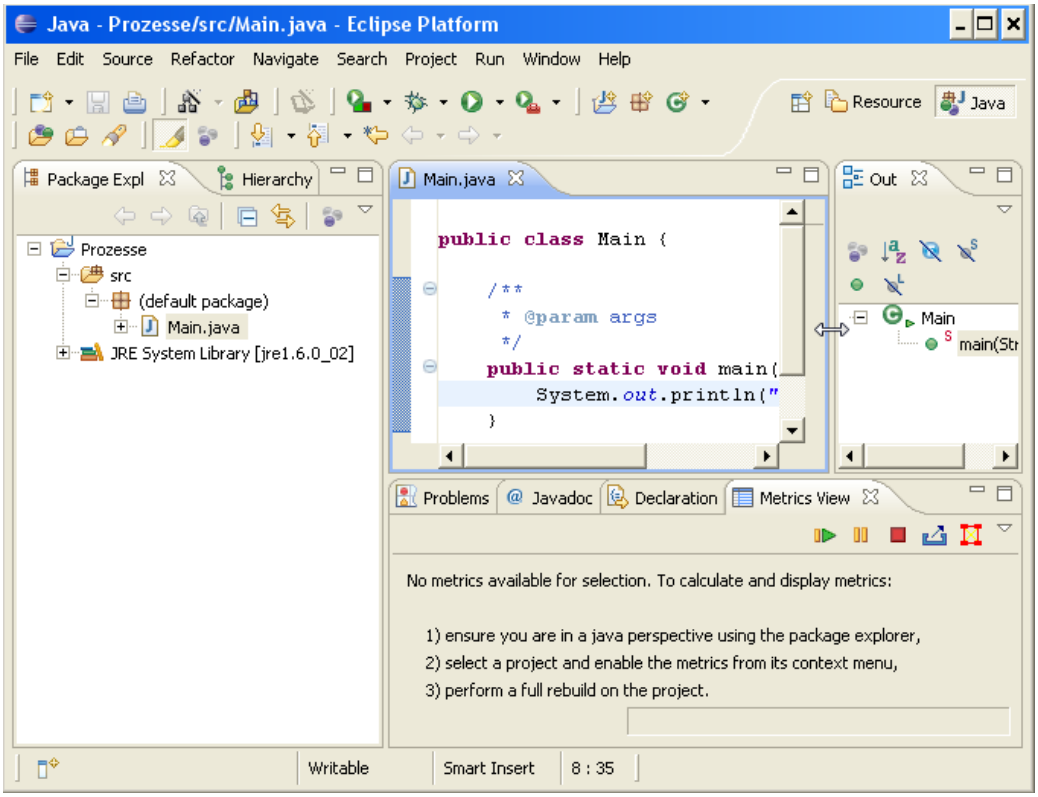
Wählt man z. B. wie im folgenden Bild gezeigt unter „Other“ den „Metrics View“



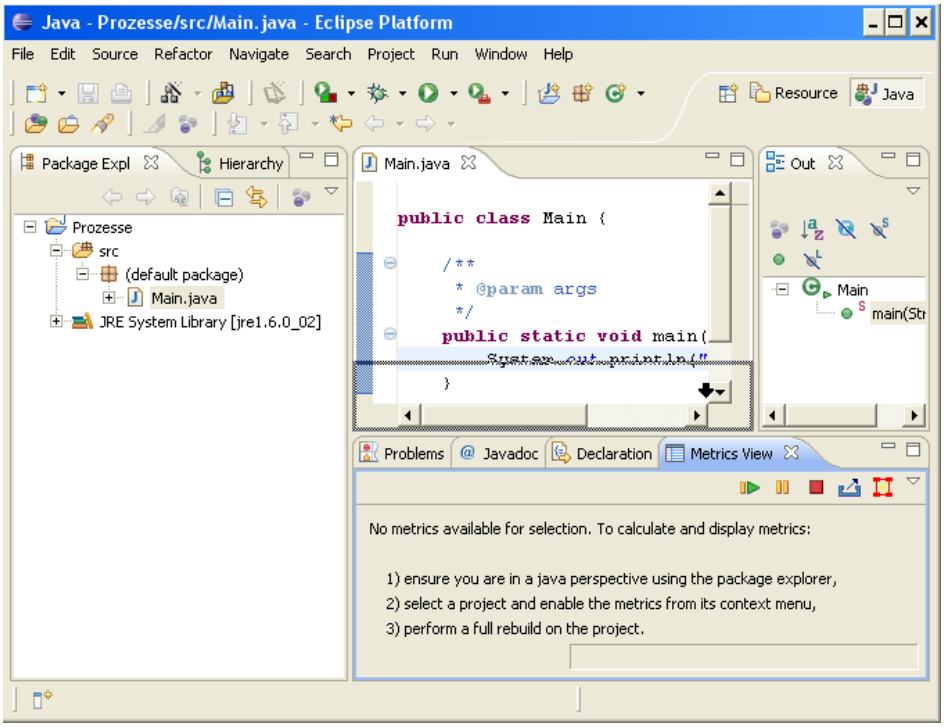
dann wird ein zusätzlicher Reiter mit diesem View angezeigt, wie es unten zu erkennen ist.



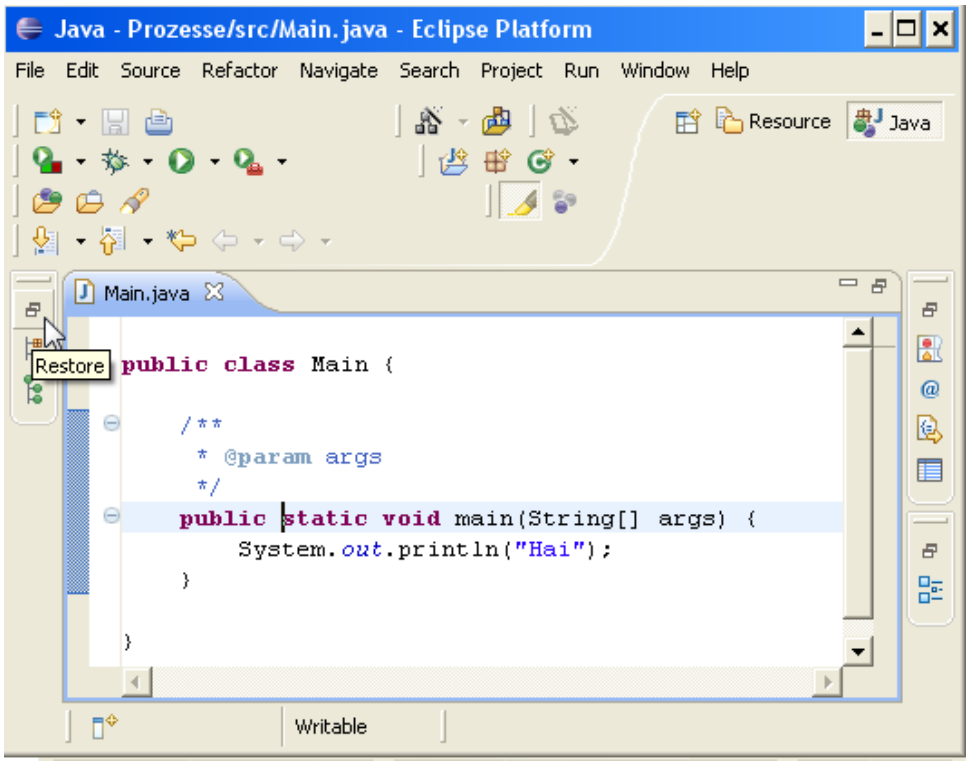
Die Fenstergrößen kann man grundsätzlich durch das Verschieben der Verbindungslinien ändern, wie man im folgenden Bild rechts in der Mitte mit dem Doppelpfeil sieht.



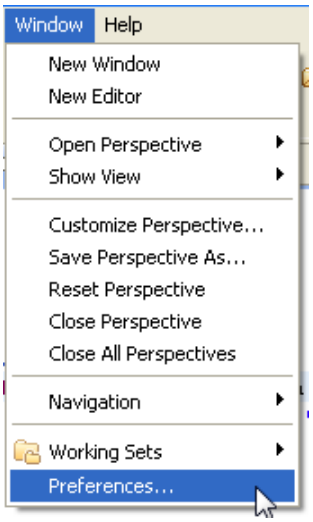
Klickt man mit der linken Maustaste in der Mitte auf den Namen eines Reiters und hält die Maustaste fest, kann dieser im Fenster verschoben werden und einen neuen Platz einnehmen. Damit ist der Reiter verschiebbar oder es wird ein zusätzliches Detailfenster, wie in der Mitte des folgenden Bildes mit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



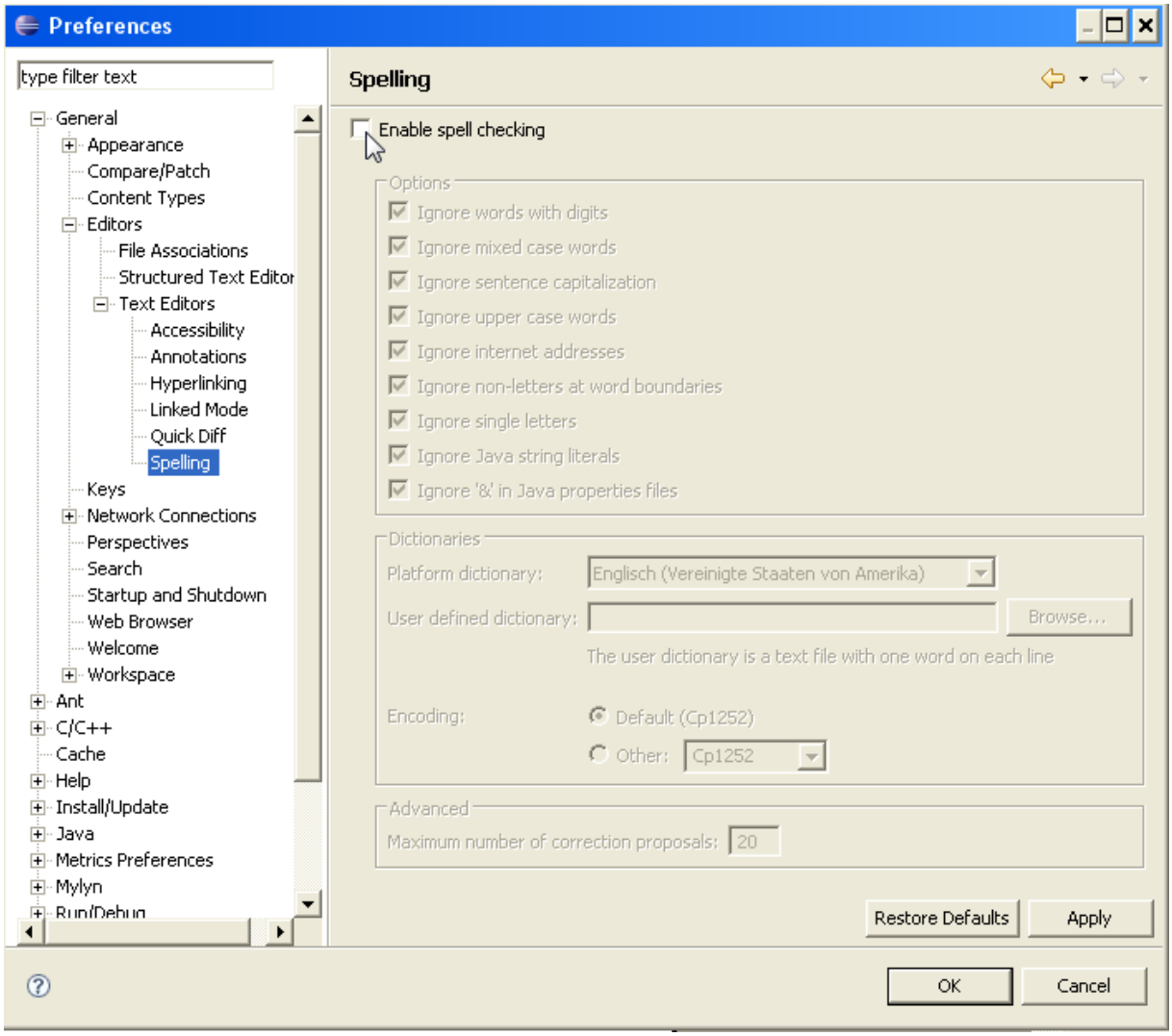
Durch einen Doppelklick auf einen Reiter nimmt dieser, wie folgt gezeigt, die gesamte Eclipse-Oberfläche ein. Durch einen erneuten Doppelklick oder einen Klick auf Restore links oben am Rand, kehrt man zur normalen Fensteranordnung zurück.



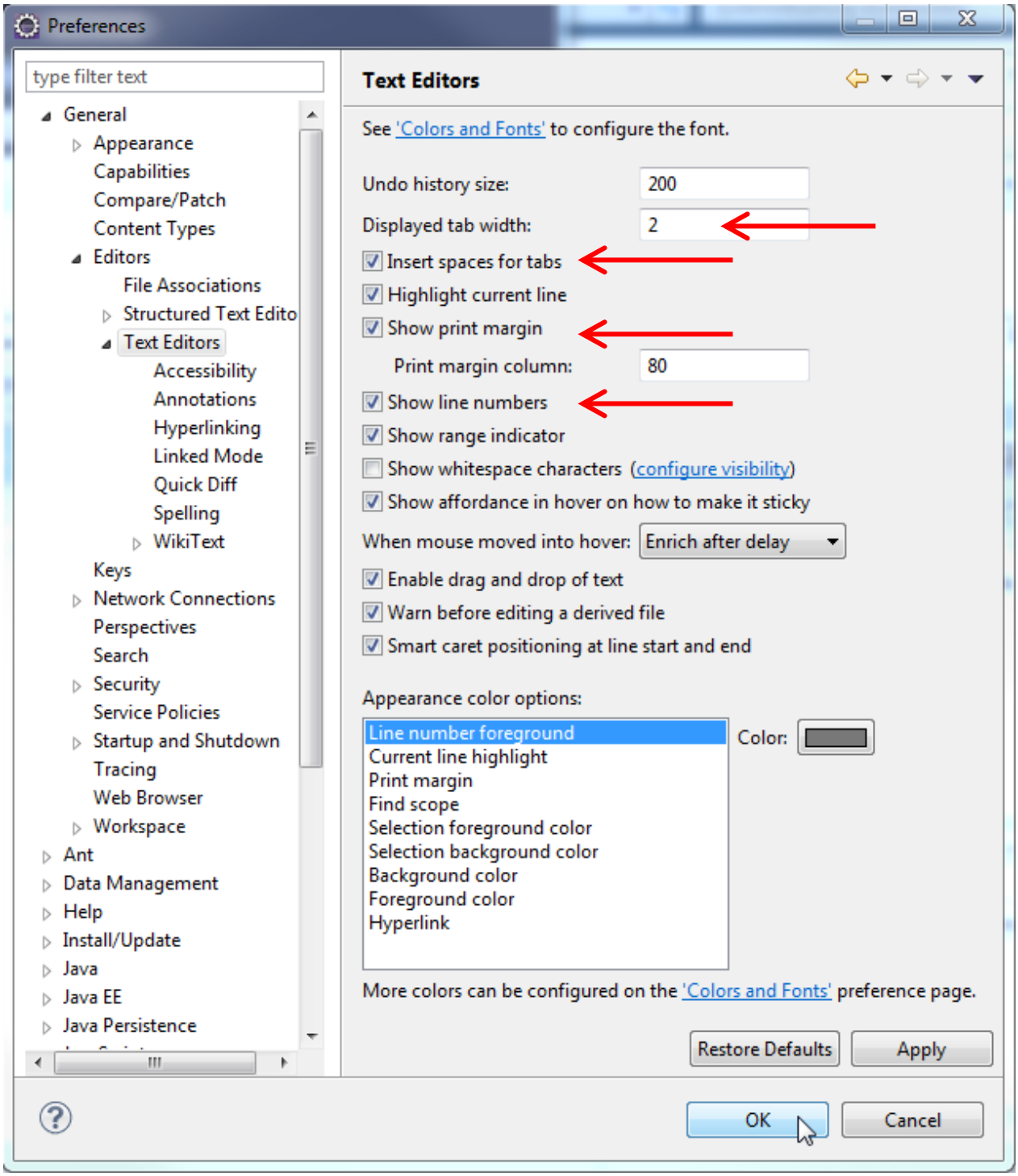
Detaileinstellungen von Eclipse werden unter „Window > Preferences“ vorgenommen.



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.

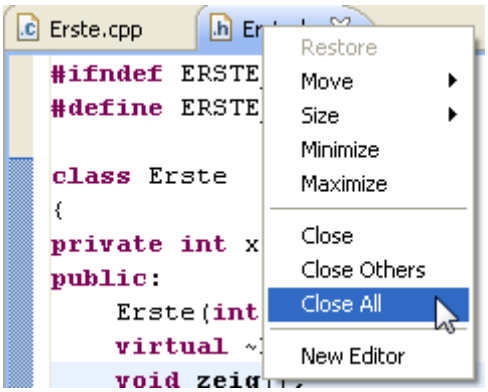


Das folgende Bild zeigt die Möglichkeit, u. a. Zeilennummern und einen Rand für den Druckbereich einzustellen. Weiterhin wird oben vereinbart statt Tabulatoren zum Einrücken Leerzeichen zu nutzen, was allerdings leider nicht sofort für den Java-Editor gilt.

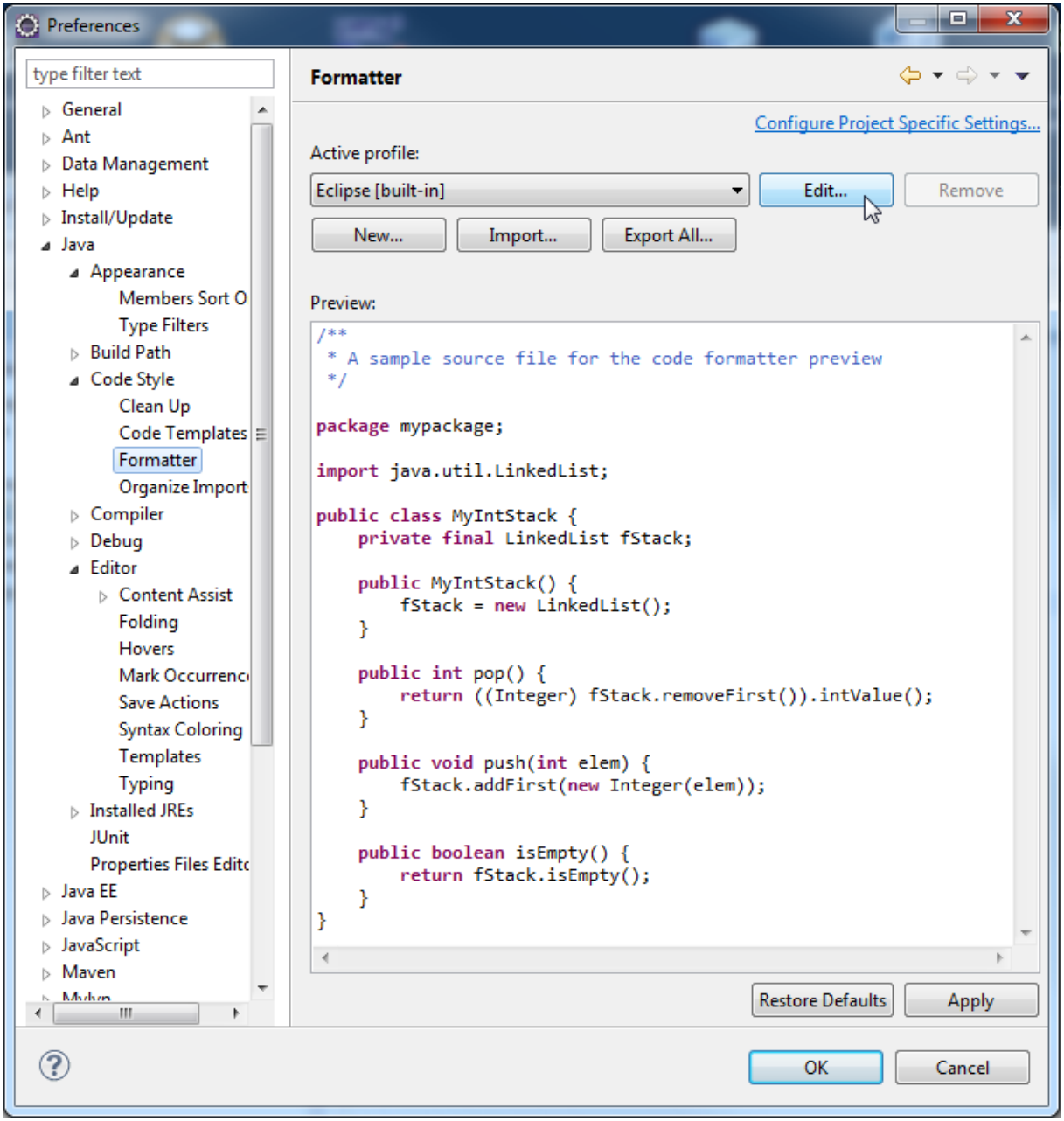


Wenn Dateien geändert werden, erkennt man an einem kleinen Stern rechts neben dem Namen, dass diese Datei verändert und noch nicht gespeichert wurde.

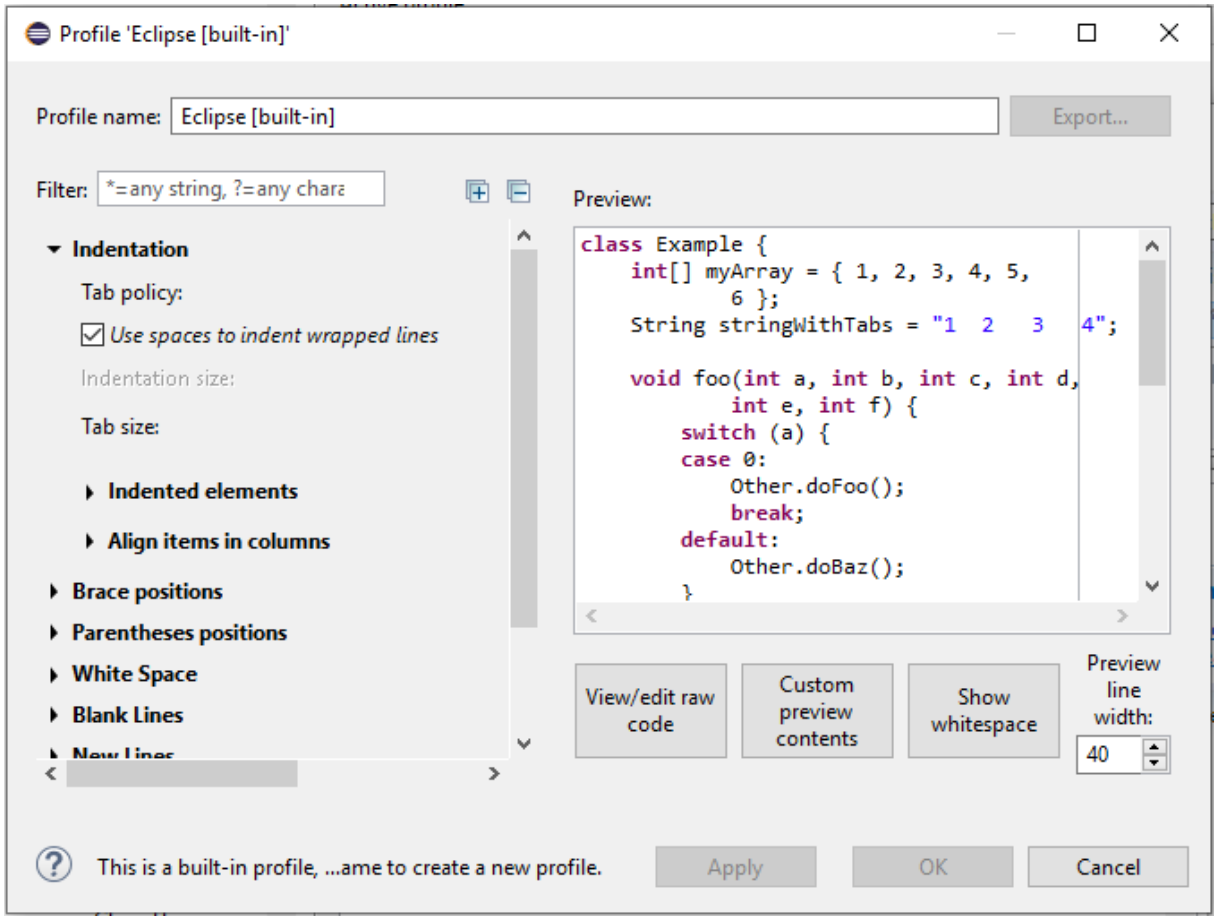
Will man mehrere Dateien auf einmal schließen, so kann man einen Rechtsklick auf einen geöffneten Reiter einer beliebigen Datei machen und erhält den folgenden Auswahldialog.



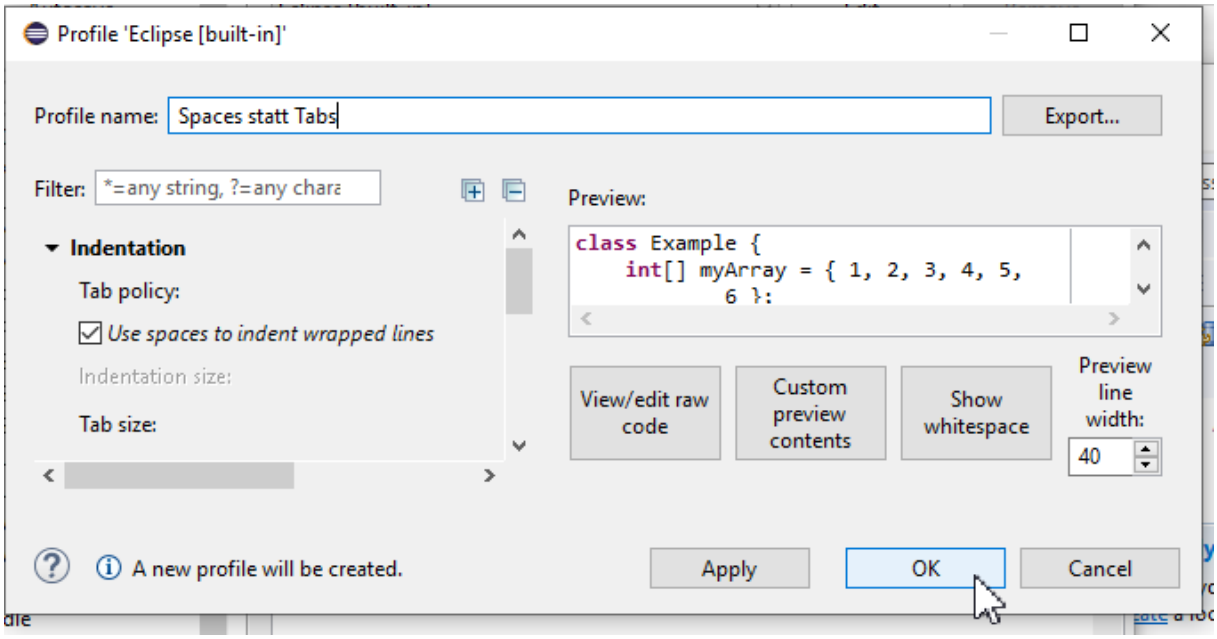
Möchte man die Code-Formatierung in Java ändern, muss man bei den Preferences „Java > Code Style > Formatter“ auswählen und oben auf „Edit“ klicken.



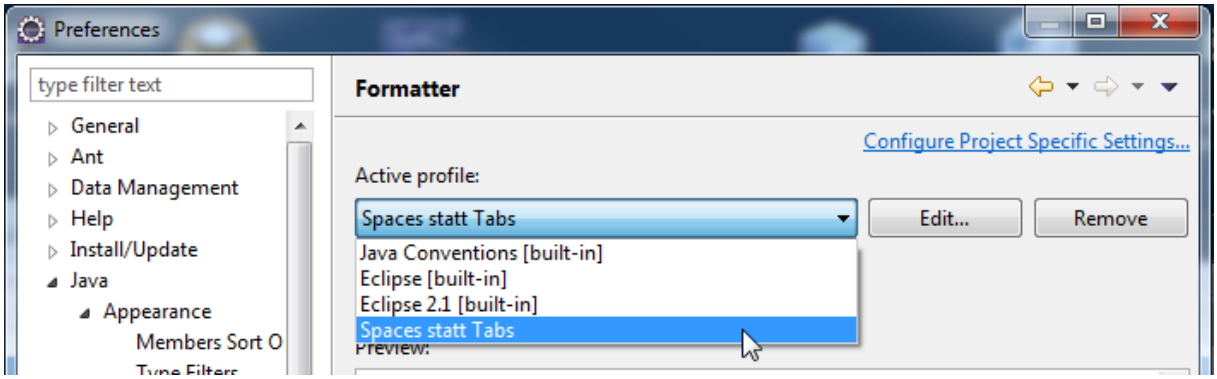
Links-oben kann dann die „Tab Policy:“ auf „Use spaces to indent wrapped lines“ geändert werden. Hier sind viele weitere Einstellungen möglich. In Großprojekten in Unternehmen werden diese Einstellungen vorgegeben, damit jeder einfach den Code jedes anderen lesen kann. Gerade Anfänger in der Software-Entwicklung sollten im Wesentlichen bei den vorgegebenen Einstellungen bleiben und sich keinen esoterischen Einrückungsstil angewöhnen, obwohl man „sich selbst sicher ist“, dass es so „viel besser“ funktioniert.



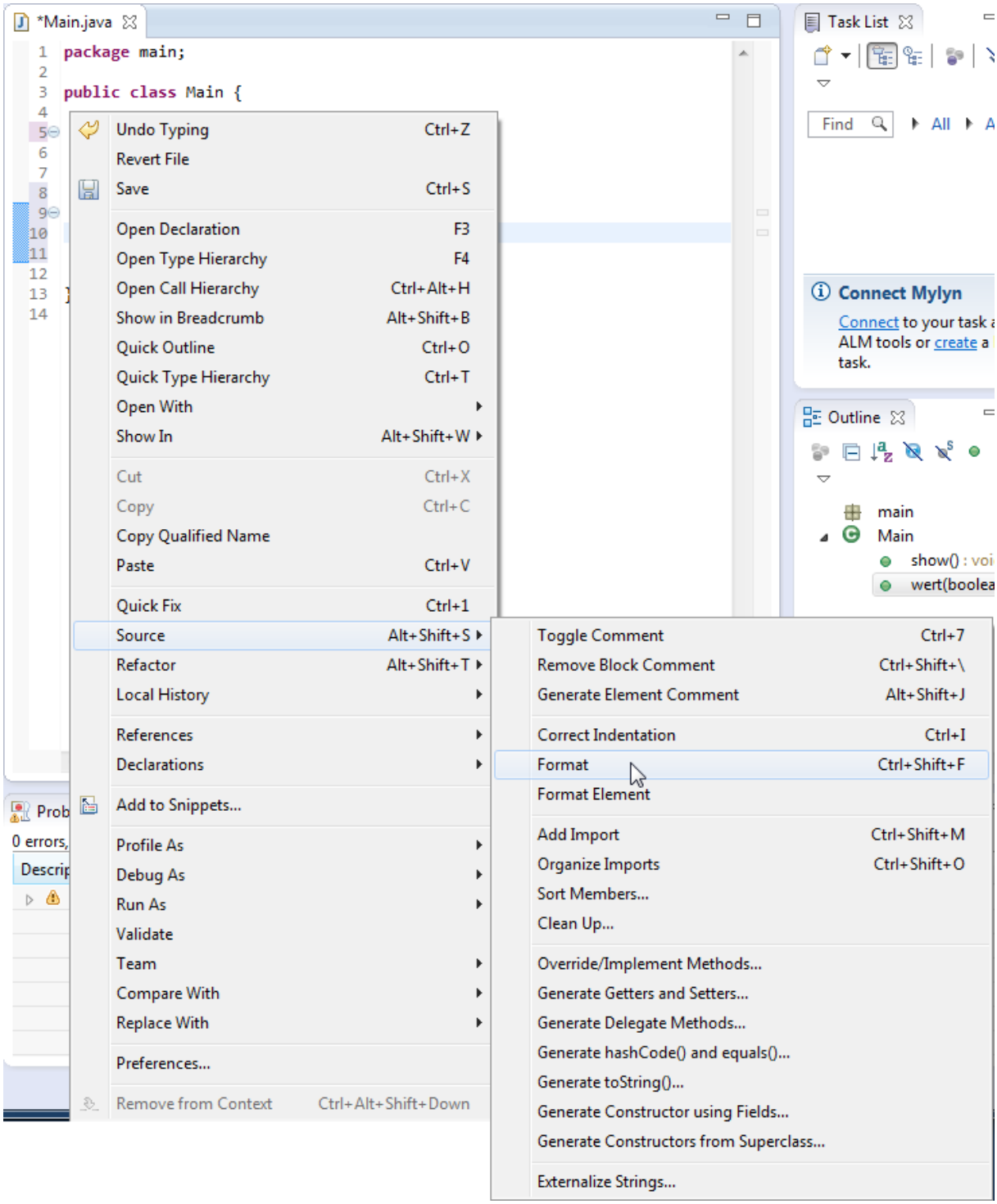
Vor dem Abspeichern muss ganz oben ein eigener „Profil name“ vergeben werden.



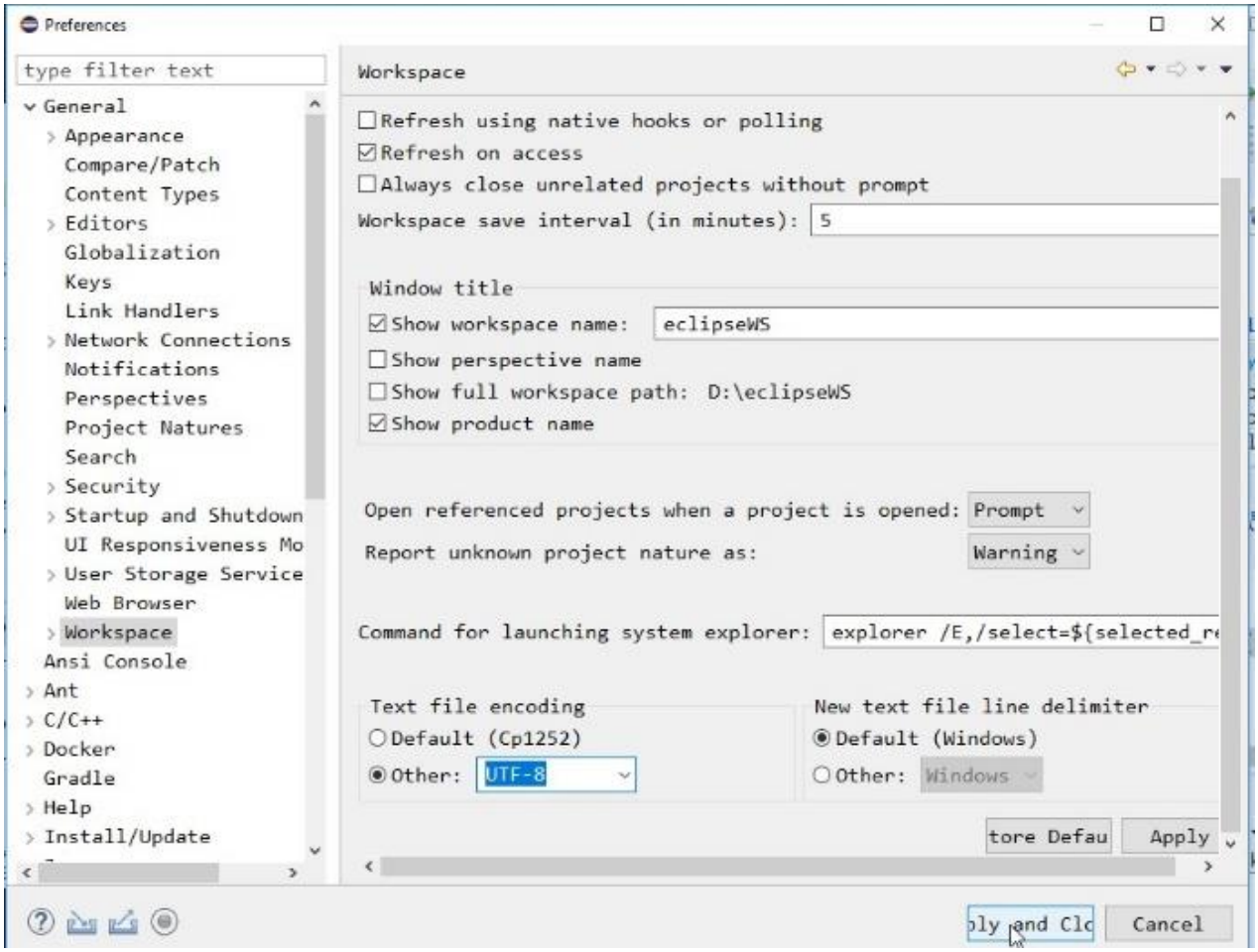
Das Profil muss dann auch ausgewählt werden.



Generell kann dann der Code automatisch mit der angegebenen Formatierung formatiert werden. Dazu wird ein Rechtsklick an einer freien Stelle im Editor-Fenster gemacht, „Source“ und dann „Format“ ausgewählt.



Eine sehr wichtige Einstellung ist die verwendete Zeichenkodierung mit der Textdateien, also auch Quellcode, abgespeichert werden. Um möglichst flexibel zu sein, sollte unter „General > Workspaces“ links-unten unter „Text file encoding“ der Eintrag auf „Other“ und in der Auswahl auf „UTF-8“ zu setzen.

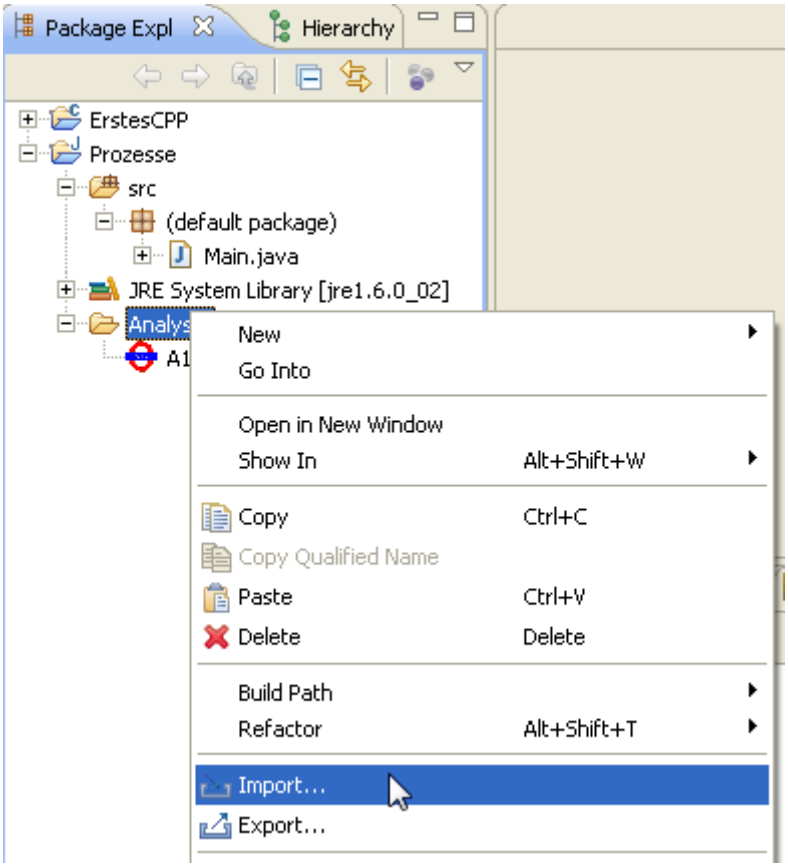

Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Globalization
Keys
Link Handlers
Network Connections
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutdown
UI Responsiveness Mo
User Storage Service
Web Browser
Workspace
Ansi Console
Ant
C/C++
Docker
Gradle
Help
Install/Update
Workspace
Refresh using native hooks or polling
Refresh on access
Always close unrelated projects without prompt
Workspace save interval (in minutes): 5
Window title
Show workspace name: eclipseWS
Show perspective name
Show full workspace path: D:\eclipseWS
Show product name
Open referenced projects when a project is opened: Prompt
Report unknown project nature as: Warning
Command for launching system explorer: explorer /E,/select=${selected_re
Text file encoding
Default (Cp1252)
Other: UTF-8
New text file line delimiter
Default (Windows)
Other: Windows
tore Defau
Apply
Cancel

5 Dateien importieren (4.11.0)

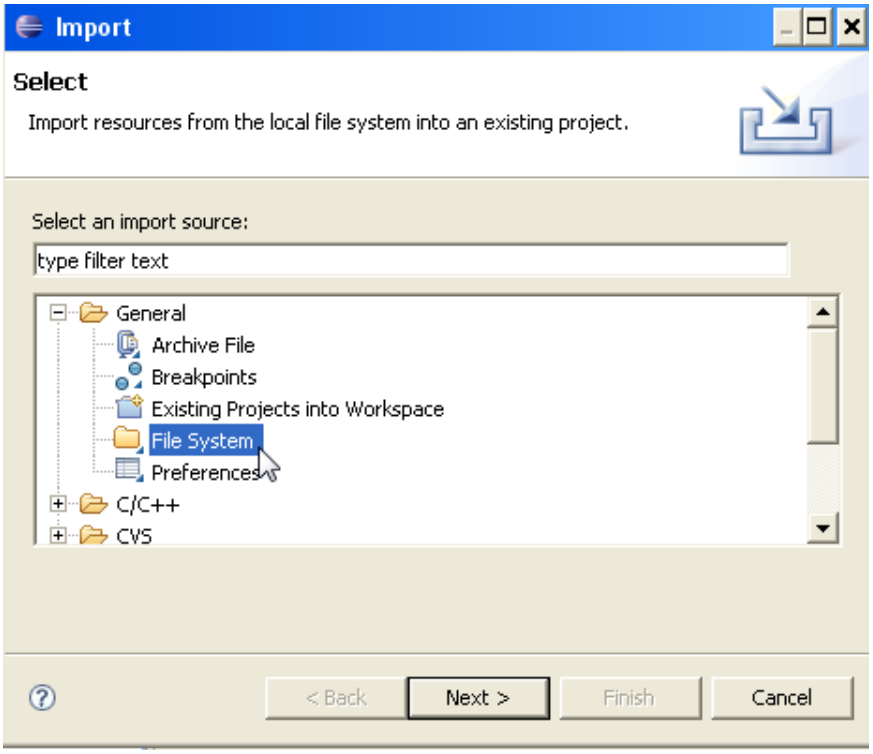
Es gibt verschiedene Varianten, was für Dateien man in ein Eclipse-Projekt laden kann. Die einfachste Möglichkeit besteht darin, dass eine Datei direkt genutzt werden soll. Die zweite Variante ist, dass zu verändernde Dateien in einer zip- oder jar-Datei vorliegen. Die Einbindung einer jar-Datei zur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

5.1 Einzelne Dateien importieren

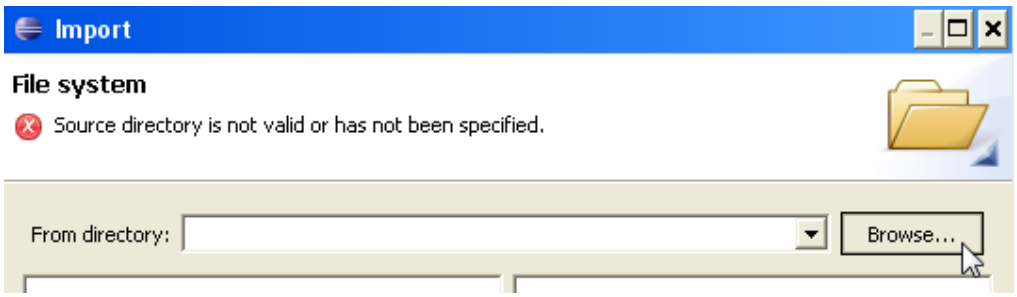
Zunächst soll eine einfache Datei in ein Eclipse-Projekt kopiert werden. Gestartet wird z. B. mit einem Rechtsklick auf den Ordner in den die Datei geladen werden soll und anschließendem Klick auf „Import..“.

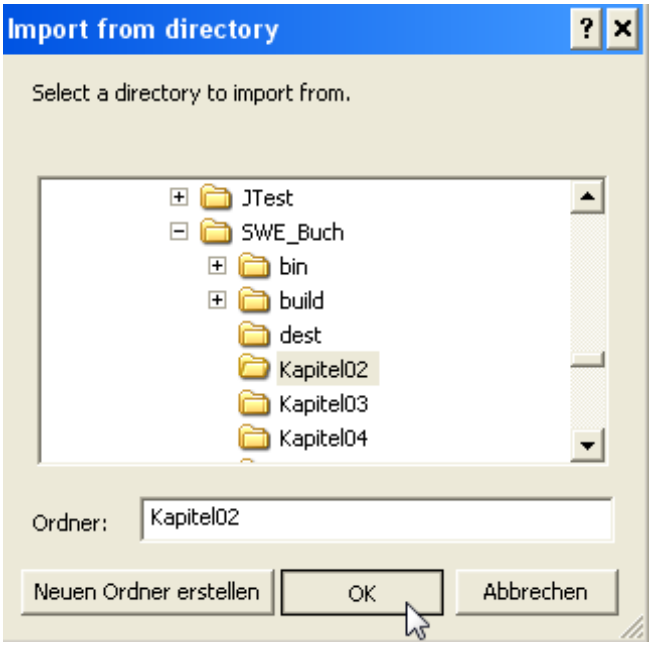


Aus den angezeigten Möglichkeiten wird dann „General > File System“ ausgewählt.

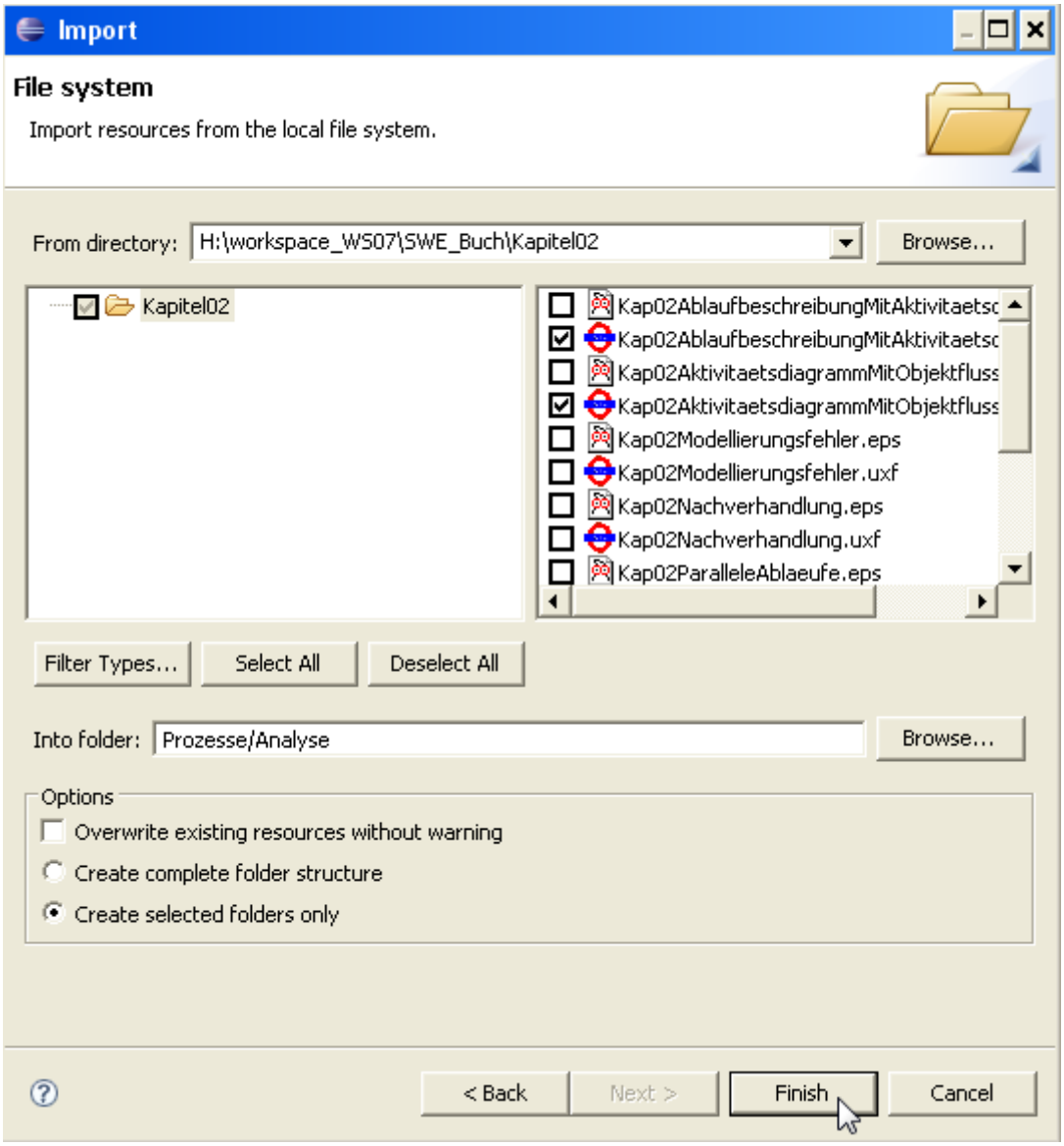


Danach kann man unter „Browse“ zu dem Verzeichnis manövrieren, aus dem man eine Datei oder mehrere Dateien einladen möchte.





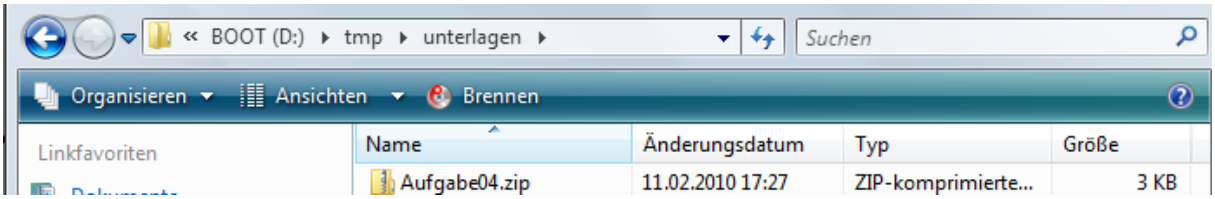
In dem sich dann öffnenden Dialog kann man auf der linken Seite in verschiedene Unterverzeichnisse manövrieren und dann auf der rechten Seite die gewünschten Dateien auswählen. Man beachte noch die meist nicht zu ändernden Einstellungen unten im Dialog.



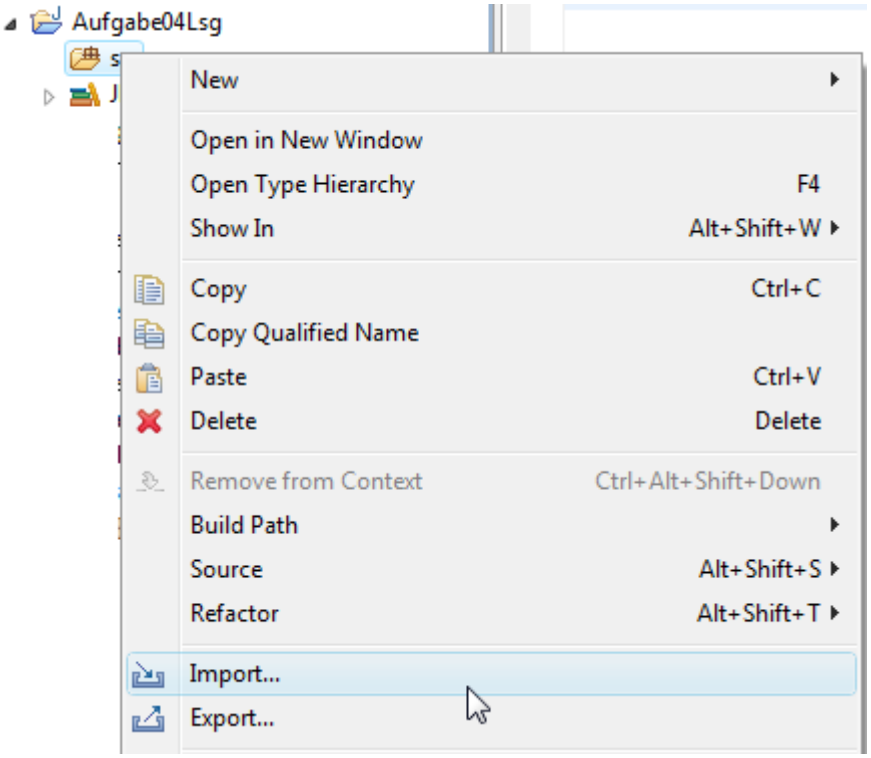
Nach einem Klick auf „Finish“ werden die Dateien dann in das Eclipse-Projekt kopiert.

5.2 Dateien aus einer Zip-Datei importieren

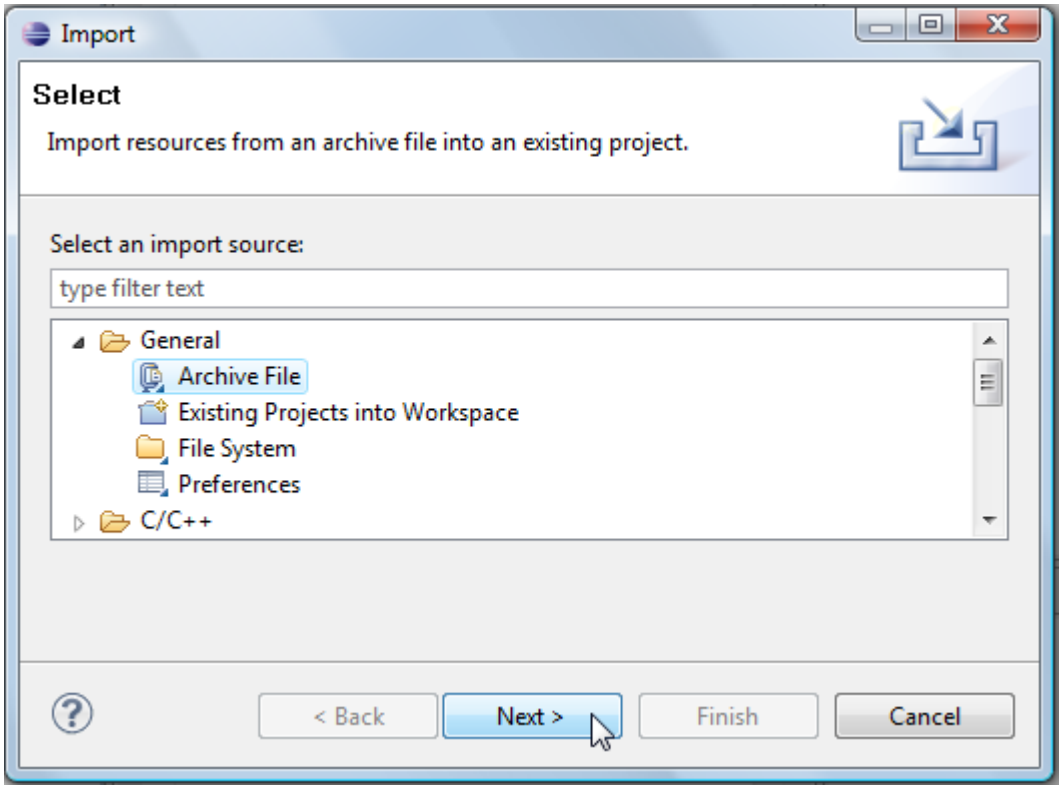
Möchte man Dateien aus einer zip- oder jar-Datei in das Projekt zur Bearbeitung kopieren, ist der Ablauf sehr ähnlich. Es wird angenommen, dass sich die Dateien in einer Zip-Datei Aufgabe04.zip befinden.



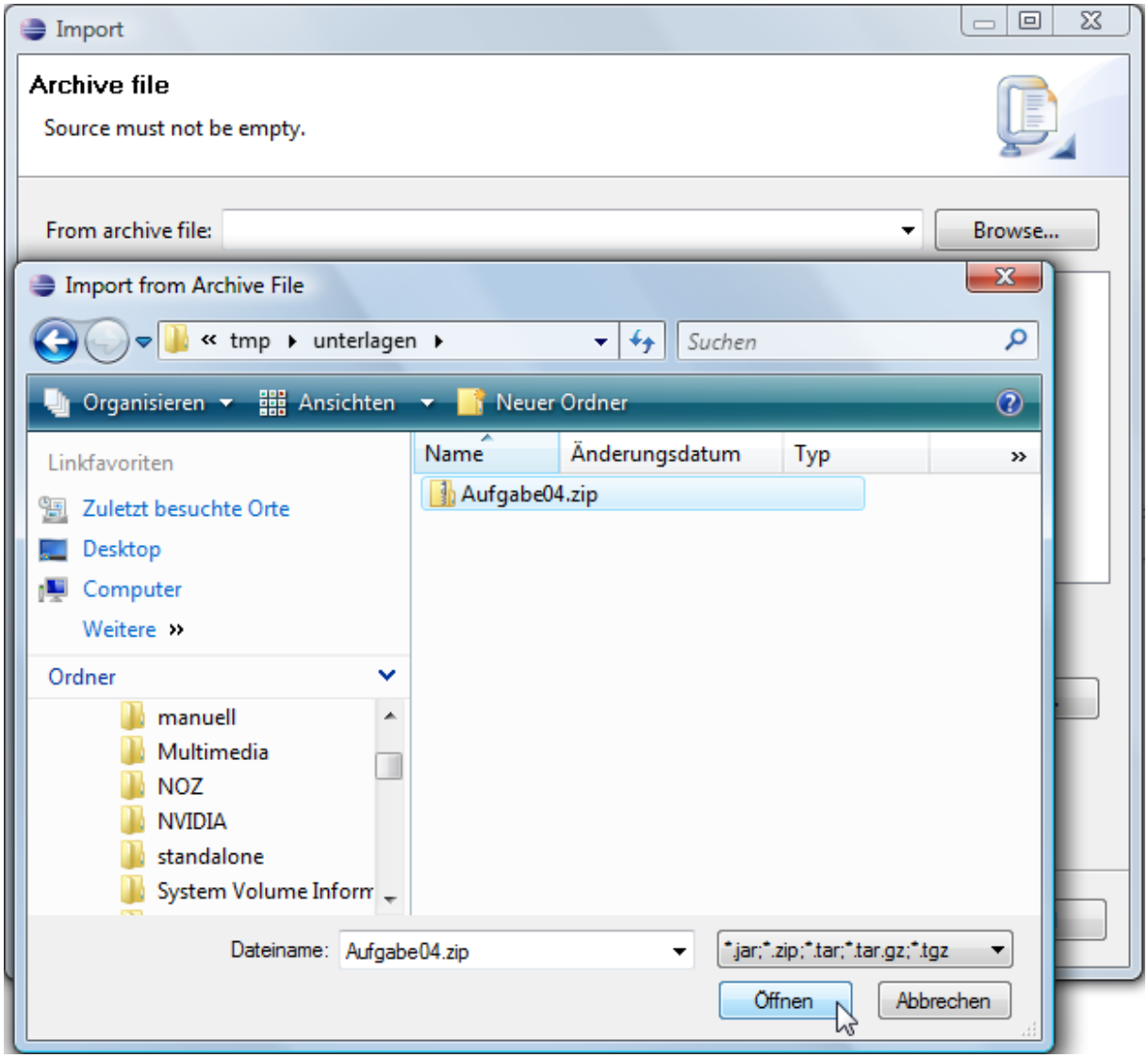
Gegeben Sei ein Projekt Aufgabe04Lsg, dann wird ein Rechtsklick auf dem Ordner src ausgeführt und „Import“ gewählt.



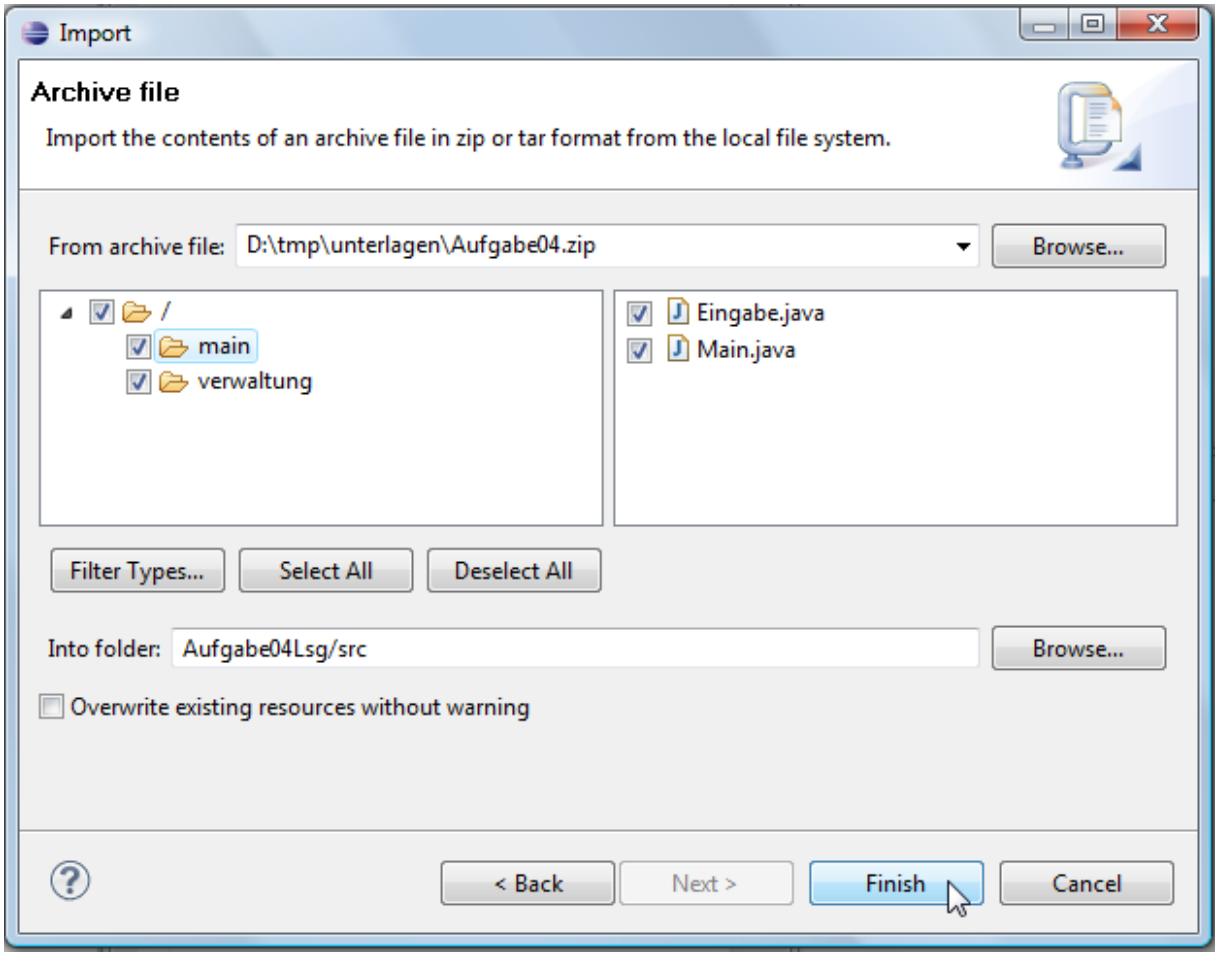
Jetzt wählt man unter „General“ dann „Archive File“ und dann „Next>“



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



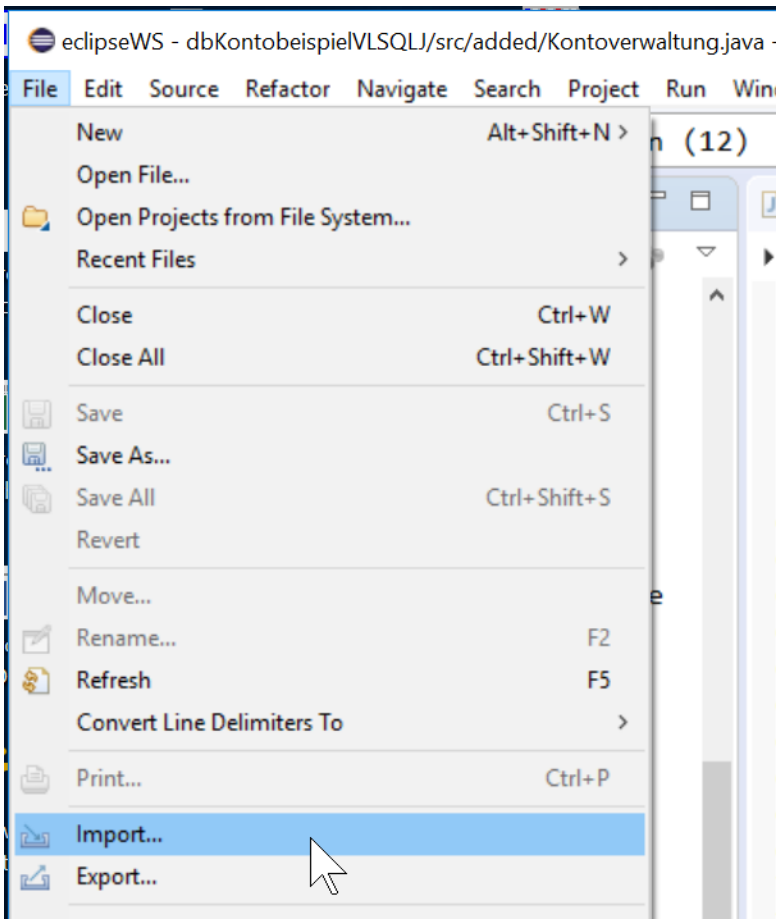
Nun werden die gewünschten Ordner und Dateien markiert und mit „Finish“ in das Projekt kopiert, so dass die Dateien bearbeitet werden können. Man beachte den Ordner unter „Into Folder:“, der zum Archiv passen muss. Als Problem könnte z. B. im Archiv sich noch der Ordner src auf oberster Ebene befinden, 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.



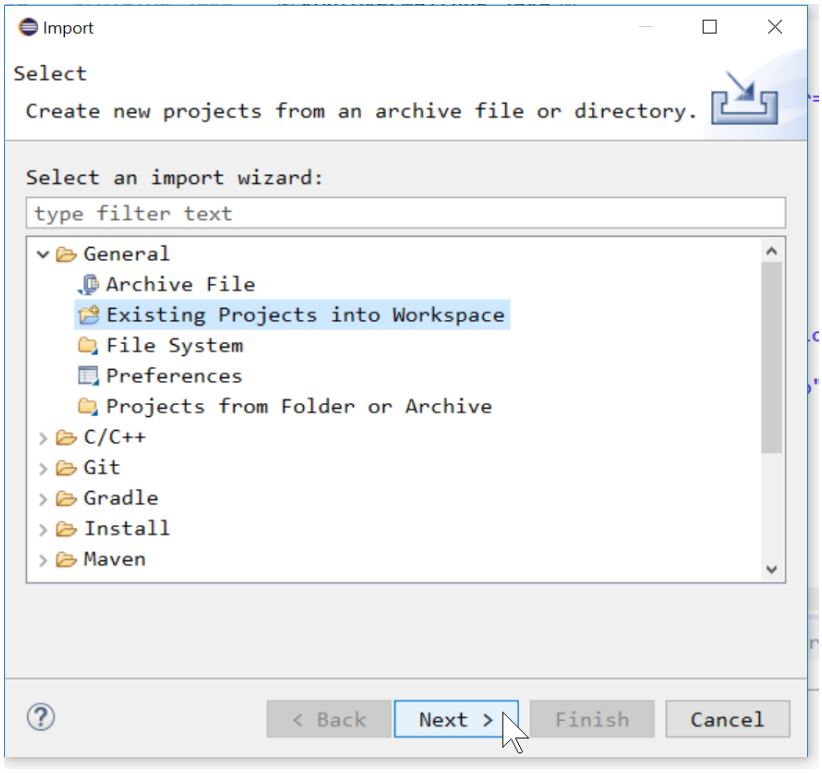
5.3 Importieren eines gegebenen Eclipse-Projekts

Im nächsten Fall soll ein gegebenes Eclipse-Projekt in den aktuellen Workspace importiert werden. Dies funktioniert auch, wenn ein Projekt als Zip-Datei geladen und im aktuellen Workspace ausgepackt wird und jetzt dort bearbeitet werden soll. Dieser Fall wird von Eclipse erkannt und die Dateien nicht nochmals kopiert.

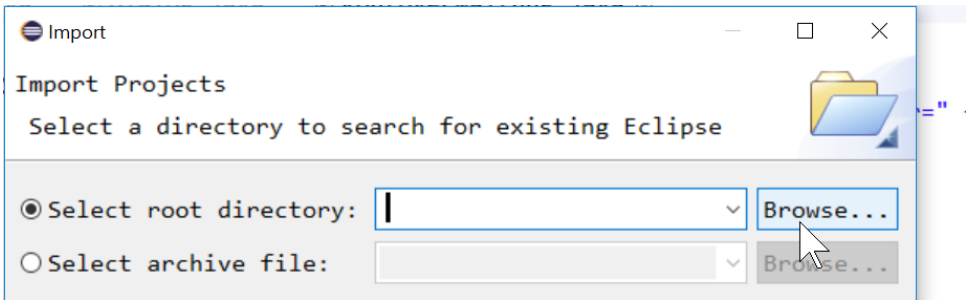
Wieder wird „File > Import…“ gewählt.



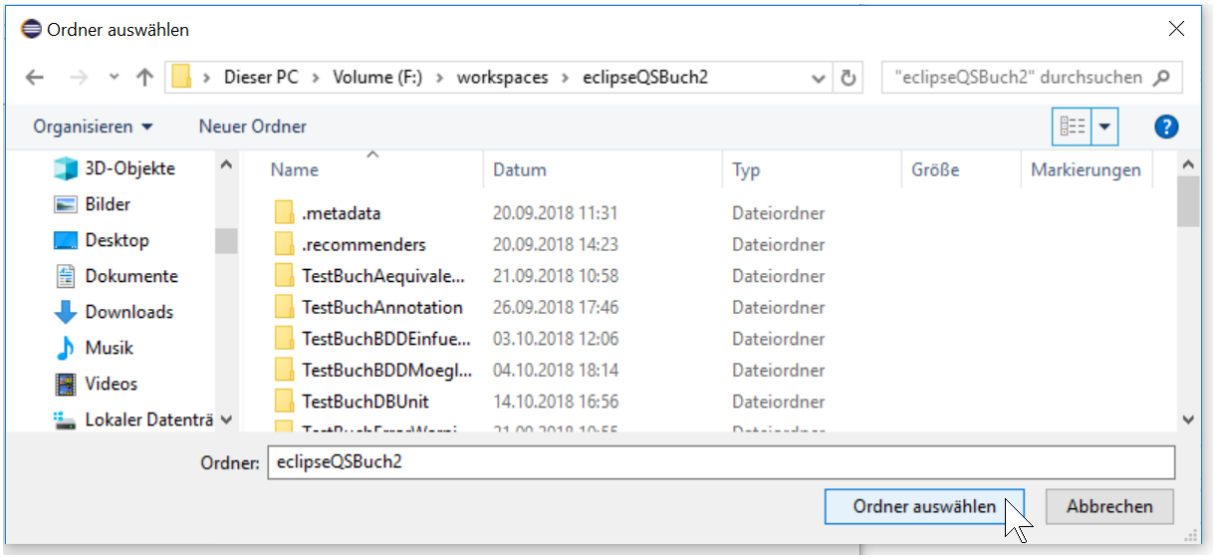
Es wird „General > Existing Projects into Workspace“ ausgewählt und „Next >“ geklickt.



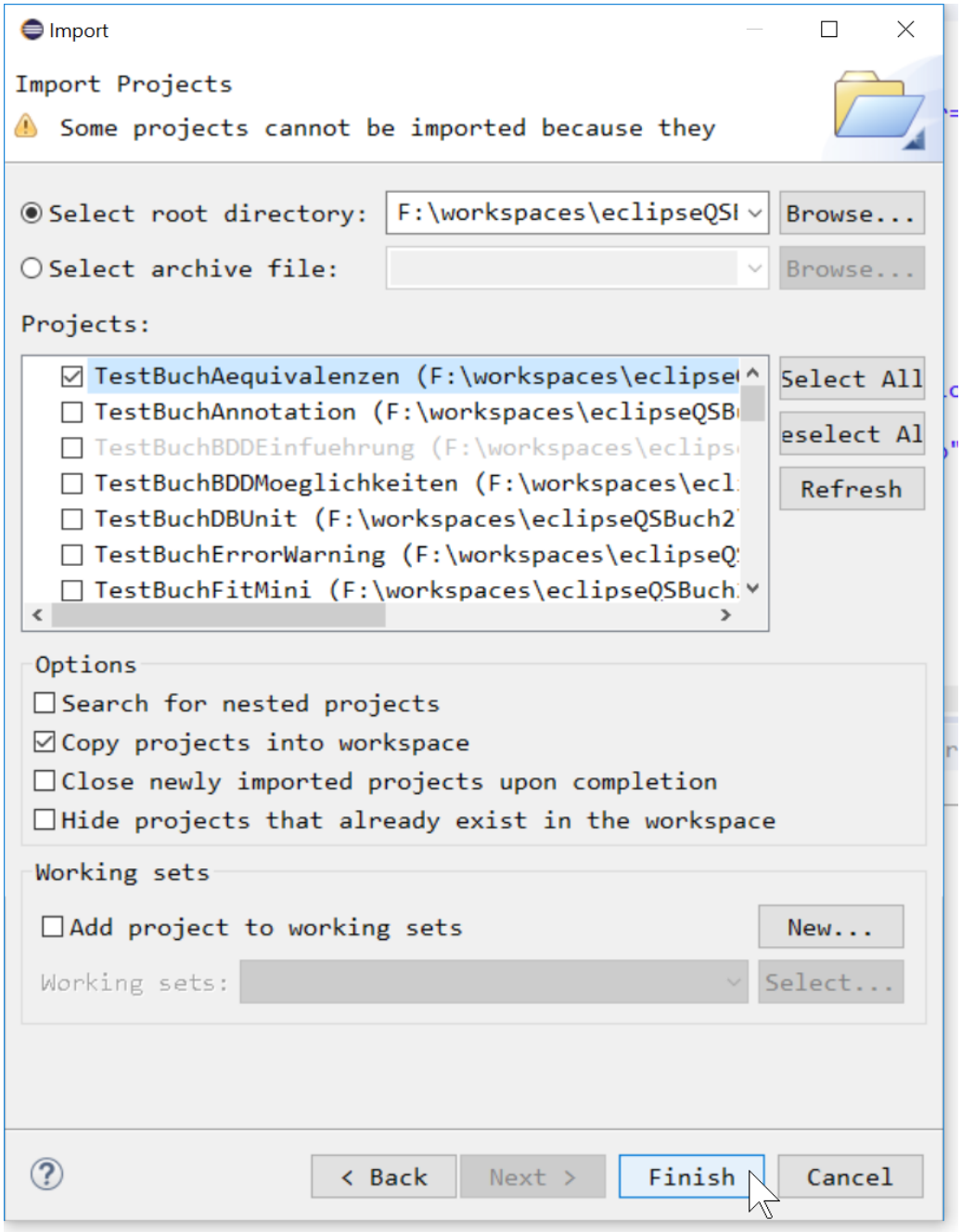
Nun wird rechts-oben mit „Browse…“ geklickt.



Es wird zu dem Verzeichnis gesteuert, das das gewünschte Eclipse-Projekt enthält und „Ordner auswählen“ geklickt.

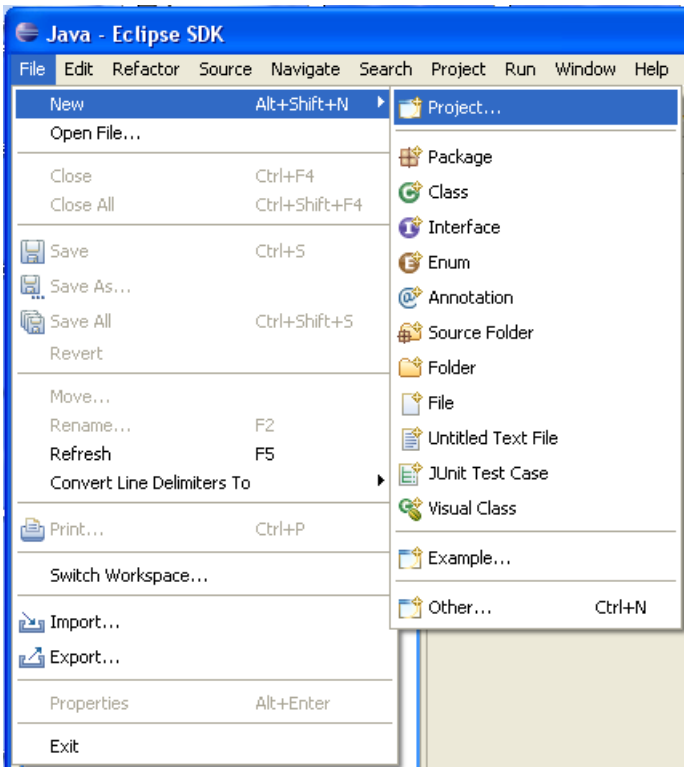


Eclipse findet automatisch mögliche Projekte. Die genaue Auswahl kann über Haken links erfolgen. Die Knöpfe rechts erlauben es alle Projekte zu selektieren („Select All“) oder keines zu selektieren („Deselect All“). Der automatisch gesetzte Haken bei „Copy projects into workspace“ garantiert, dass eine Kopie erstellt wird und die Quelle unberührt bleibt. Durch einen Klick auf „Finish“ wird der Import abgeschlossen und das Projekt kann „normal“ genutzt werden.

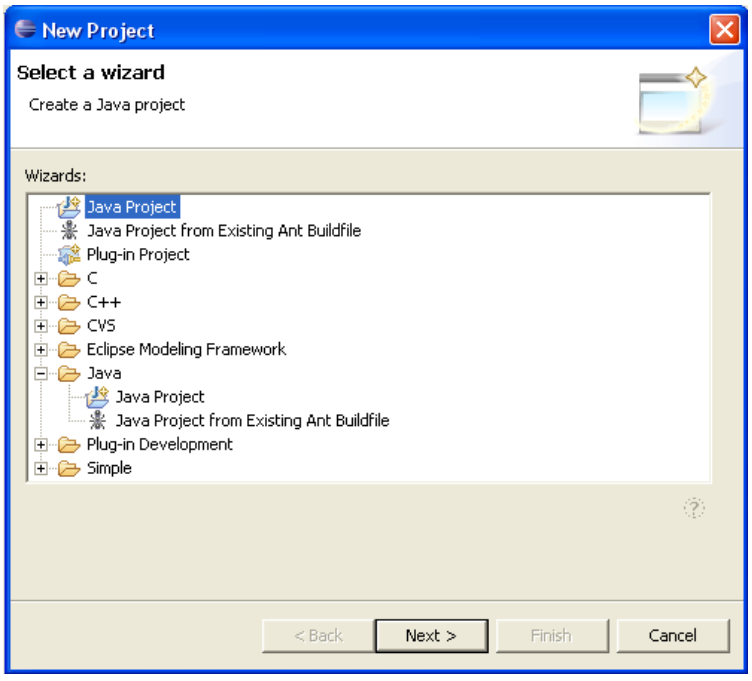

Import
Import Projects
Some projects cannot be imported because they
Select root directory:
F:\workspaces\eclipseQSI
Browse...
Select archive file:
Browse...
Projects:
TestBuchAequivalenzen (F:\workspaces\eclipse
TestBuchAnnotation (F:\workspaces\eclipseQSB
TestBuchBDDEinfuehrung (F:\workspaces\eclips
TestBuchBDDMoeglichkeiten (F:\workspaces\ecl
TestBuchDBUnit (F:\workspaces\eclipseQSBuch2
TestBuchErrorWarning (F:\workspaces\eclipseQ
TestBuchFitMini (F:\workspaces\eclipseQSBuch
Select All
eselect Al
Refresh
Options
Search for nested projects
Copy projects into workspace
Close newly imported projects upon completion
Hide projects that already exist in the workspace
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel

6 Anlegen eines Java-Projekts (4.20.0)

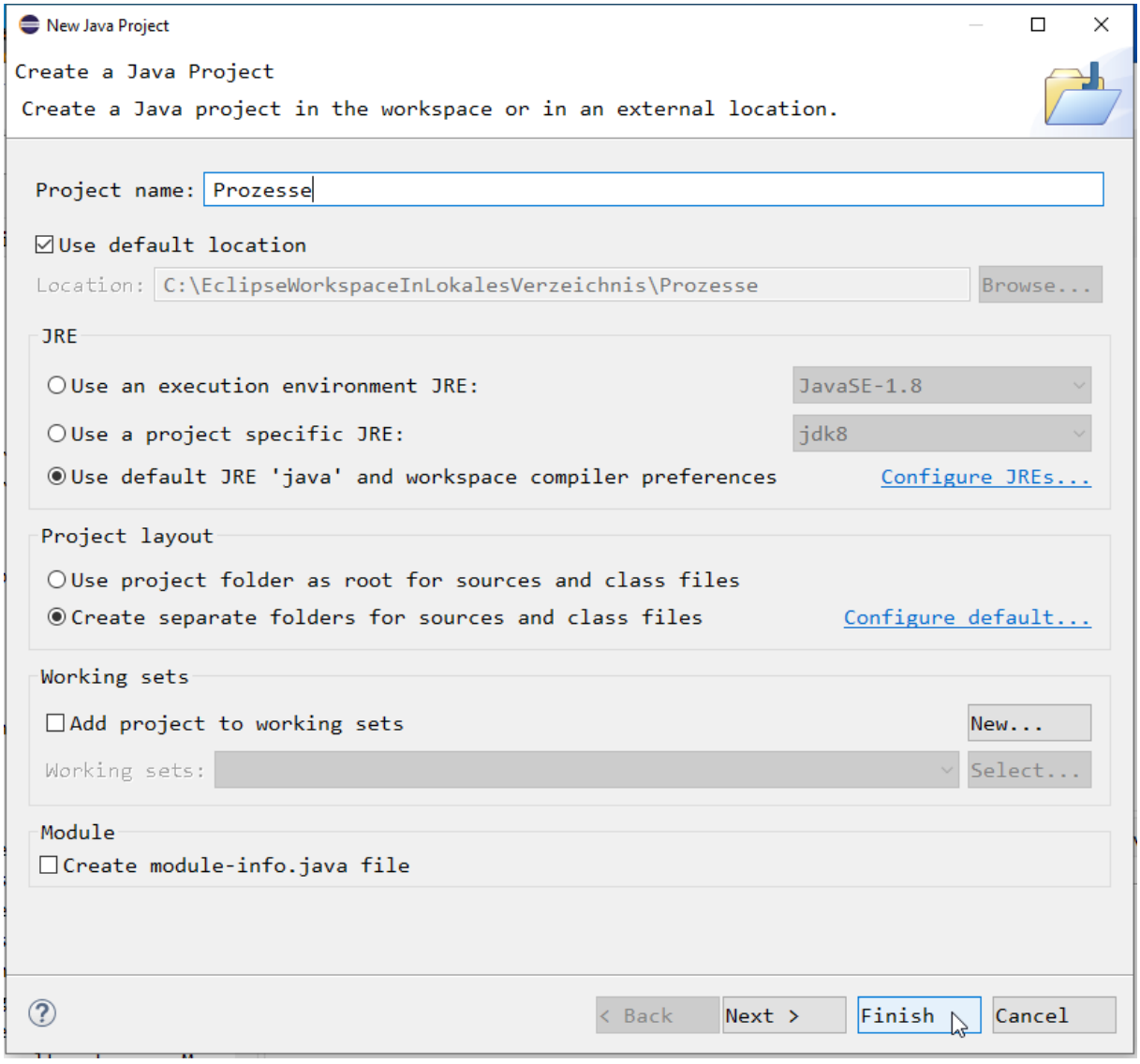
Zunächst wird ein neues Projekt unter „File > New > Project“ angewählt, wobei der obere Teil dieses Menüs sich den zuletzt getroffenen Auswahlen anpasst.



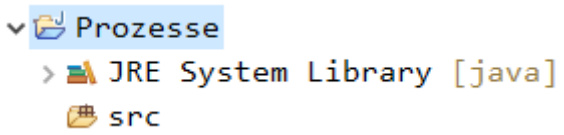
Dann wird „Java > Java Project“ angewählt und dann „Next >“ geklickt. Dieser Punkt kann mit gleicher Bedeutung mehrfach im Auswahlbaum erscheinen.



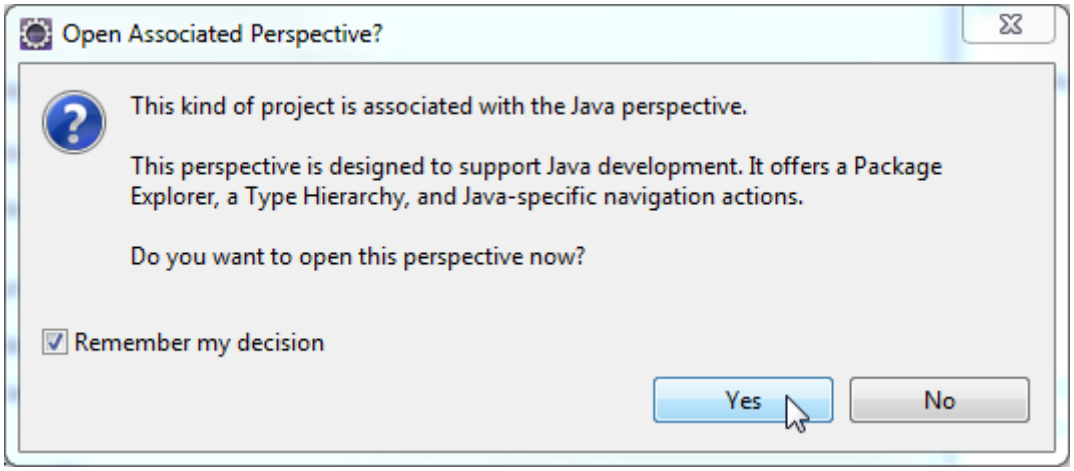
Im folgenden Fenster wird oben der Projektname eingetragen. Weiter unten unter „Module“ muss festgelegt werden, ob eone Datei module-info.java angelegt werden soll, mit der festtgehalten wird, dass das Modul-System von Java genutzt wird. Ist man Anfänger oder/und wird Java nur fürt Beispielcode genutzt, sollte Java bis einschließlich Java 8 genutzt werden oder soll der Code im Wesentlichen damit kompatibel bleiben, ist der Haken wegzunehmen. In modernen Projekten bleibt dieser Haken stehen. Für das hier gezeigte Einführungsbeispiel wird der Haken weggenommen. Im nachfolgenden Text wird kurz beschrieben, was zusätzlich zu machen ist, wenn der Haken stehen bleibt. Weitere Details zu dem Thema stehen in Kapitel 7.11 , da wird auch gezeigt, dass ein Projekt nachträglich auf Module umgestellt werden kann. Die weiteren Einstellungen sollten so übernommen werden. Danach kann die Projekt-Einrichtung für das neue Java-Projekt unter „Finish“ abgeschlossen werden.



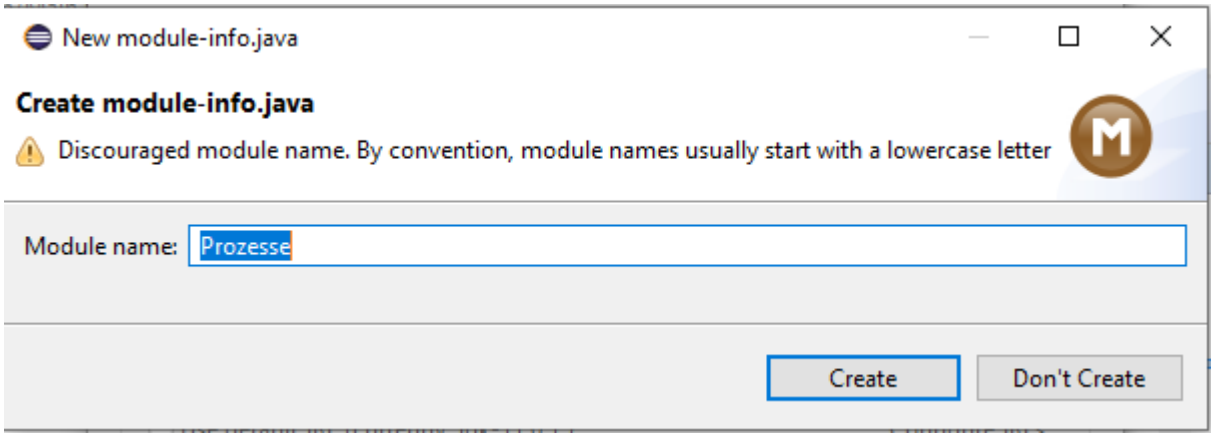
Das erstellte Projekt sieht abhängig von der genutzten Java-Version generell wie folgt aus.



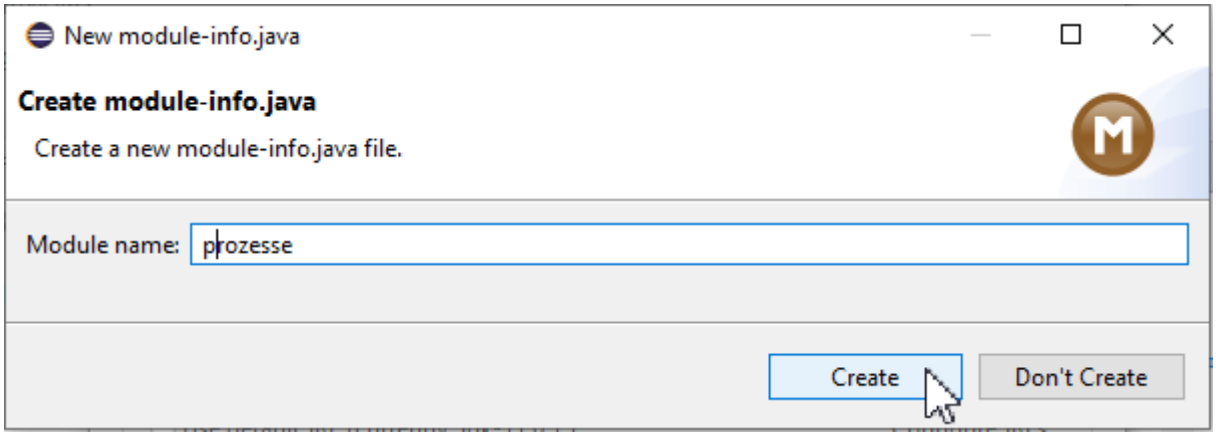
Sollte man mit einer anderen Sicht gearbeitet haben, wird man darauf hingewiesen, dass ein Wechsel sinnvoll ist. Dies kann man auch automatisieren, wenn man den Haken auf der linken Seite setzt.



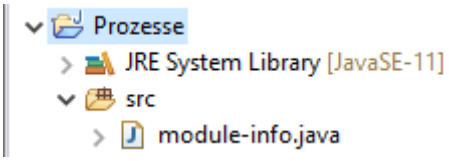
Solle der Haken zur Erstellung von module-info.java gesetzt sein, öffnet sich ein Fenster das einen „Module name“ abfragt, der mit einem kleinen Buchstaben beginnen soll.



Man könnte auf die Erstellung auch zunächst verzichten. Im konkreten Fall wird der kleingeschriebene Projektname genutzt.



Das entstandene Projekt sieht dann wie folgt aus. Detailangaben hängen von der genutzten Java-Version ab.

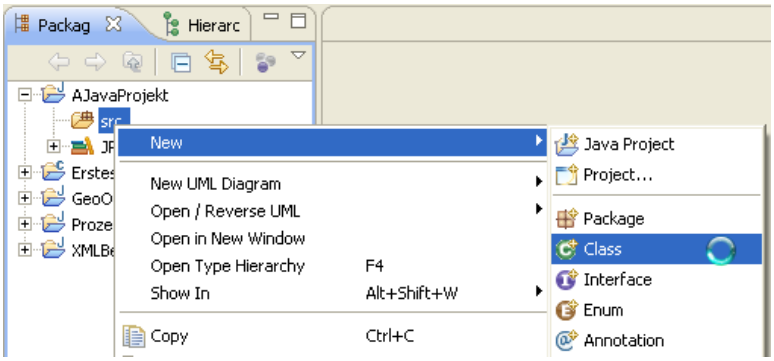


7 Java-Entwicklung mit Eclipse (4.11.0)

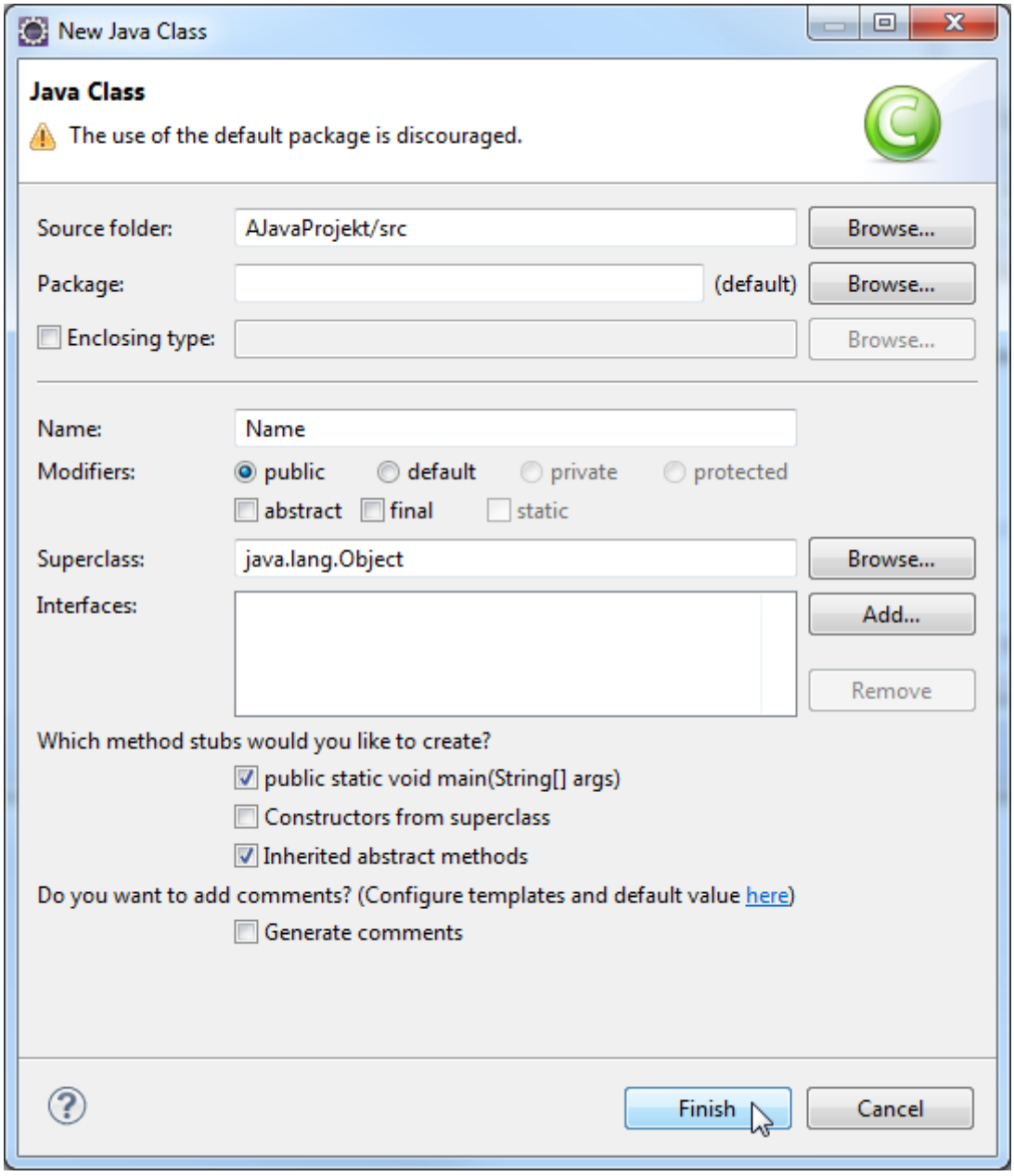
Es wird davon ausgegangen, dass ein Java-Projekt angelegt wurde und man sich in der Java-Sicht befindet (in den vorherigen Kapiteln erklärt).

7.1 Anlegen einer Klasse

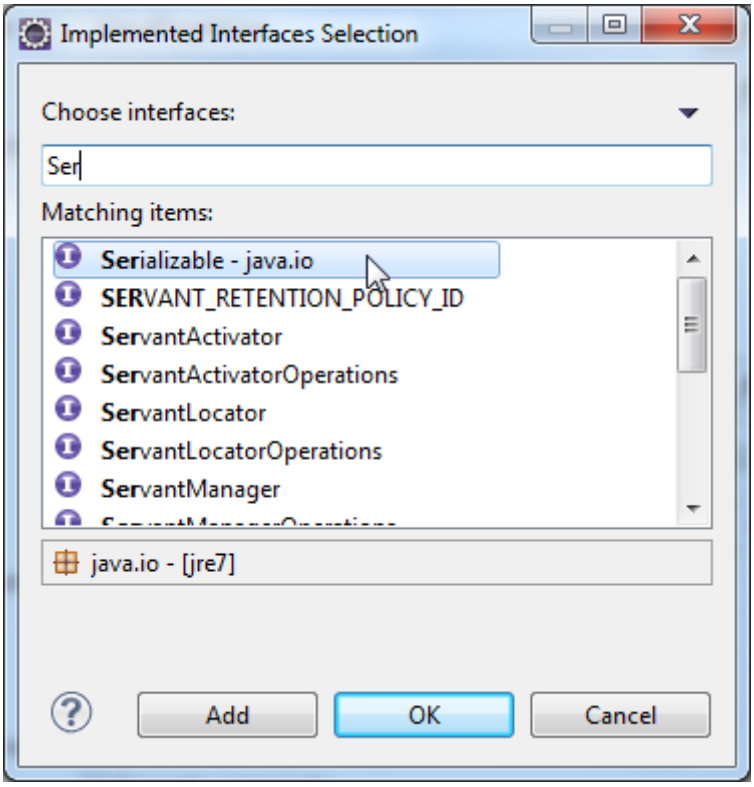
Zum Anlegen einer neuen Klasse wird z. B. ein Rechtsklick auf einen Source-Ordner, d. h. src oder ein Paket, gemacht und „New > Class>“ ausgewählt.



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.



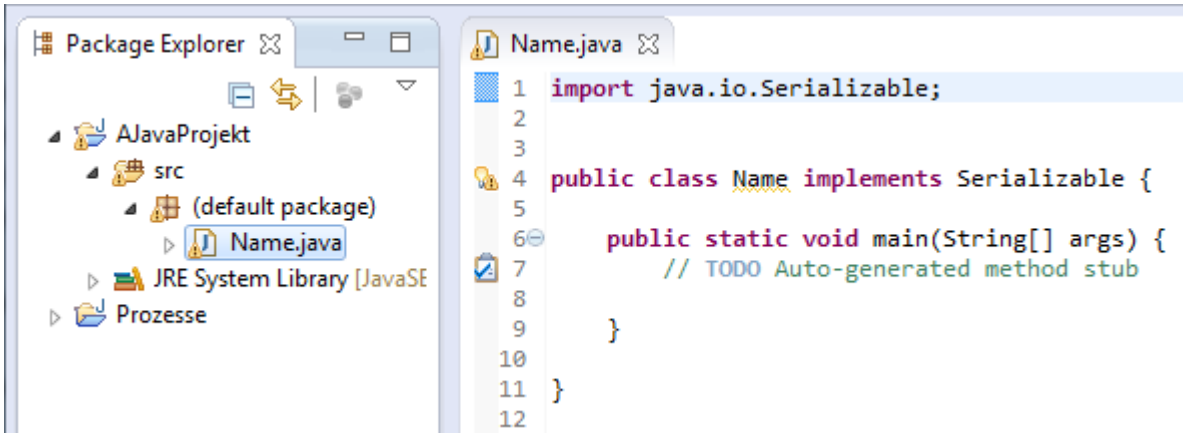
Weiterhin kann unter Interfaces angegeben werden, welche Interfaces (Schnittstellen) von der neuen Klasse implementiert werden. Die konkrete Auswahl findet über den Button am rechten Rand „Add...“ im vorherigen Fenster statt. Im sich öffnenden Fenster kann man dann oben den Namen des Interfaces eingeben, 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.



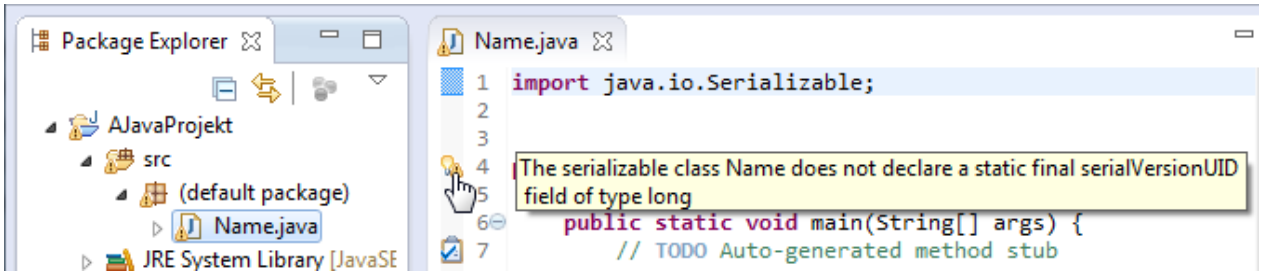
Kehrt man dann mit „OK“ zur Klassenerstellung zurück, kann weiterhin noch ausgewählt werden, ob die main()-Methode zum Programmstart in dieser Klasse generiert wird. Die Klassenerstellung wird mit „Finish“ abgeschlossen.

7.2 Automatische Fehlerkorrektur

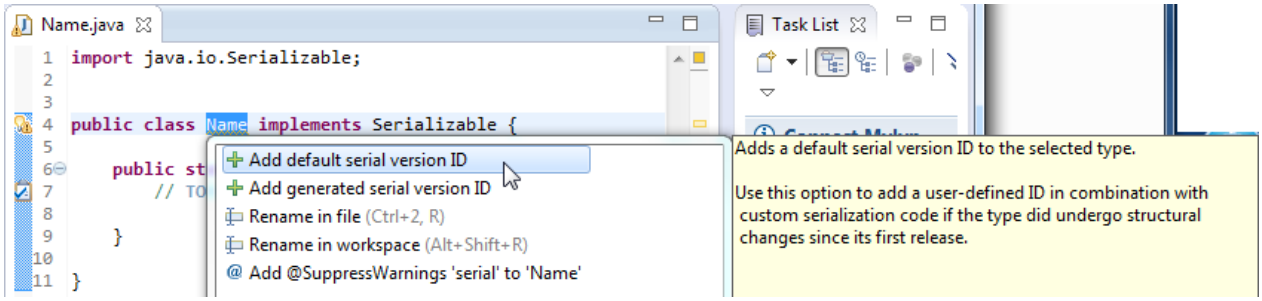
Im folgenden Beispiel sieht man die generierte Klasse, die auch zeigt, wo der Entwickler noch Kommentare ergänzen, bzw. was er noch machen muss (TODO). Am linken Rand des Editorfensters erkennt man eine Glühbirne mit einem Ausrufungszeichen, wodurch auf ein Problem, aber keinen echten Fehler hingewiesen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



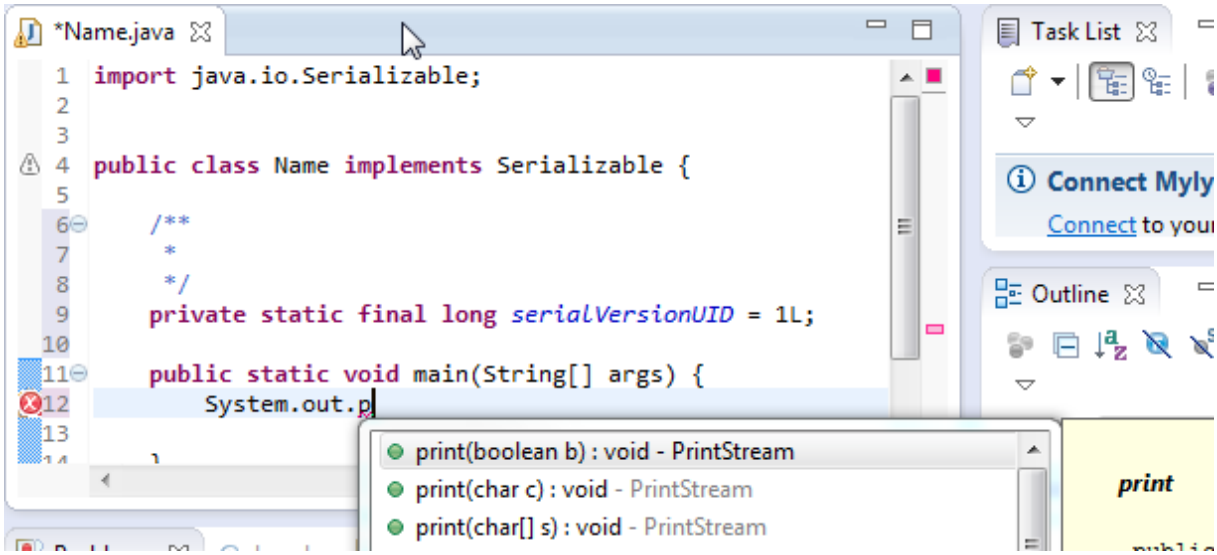
Schiebt man die Maus auf die Fehler- oder Problemmeldung, erhält man detailliertere Angaben, die auch im unteren Fenster „Problems“ sichtbar sind.



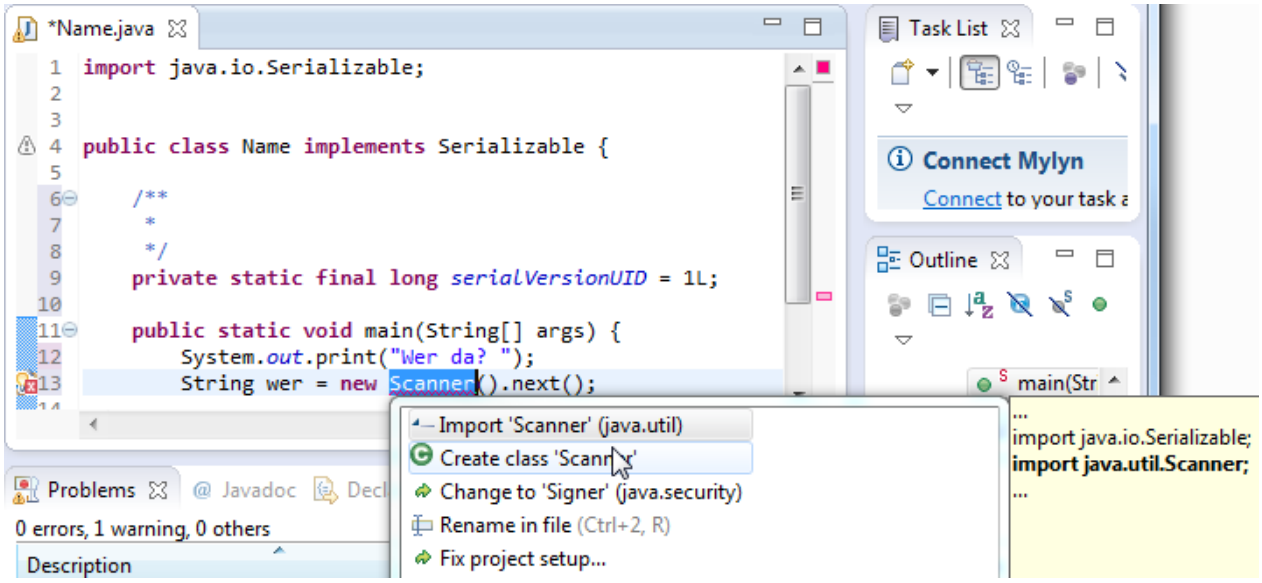
Macht man einen Linksklick auf eine Fehler- oder Problemmeldung, erhält man meist Korrekturvorschläge, von denen einer mit einem Linksklick angenommen werden kann.



Ohne auf die Details des Problems einzugehen, wird hier der erste Vorschlag mit einem Doppelklick 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.



Das eingegebene Programm sieht wie folgt aus.

```
import java.io.Serializable;
import java.util.Scanner;

public class Name implements Serializable {

	private static final long serialVersionUID = 1L;

	public static void main(String[] args) {
		System.out.print("Wer da? ");
		Scanner scanner = new Scanner(System.in);
		String wer = scanner.next();
		System.out.println("Moin, " + wer);
		scanner.close();
	}
}
```

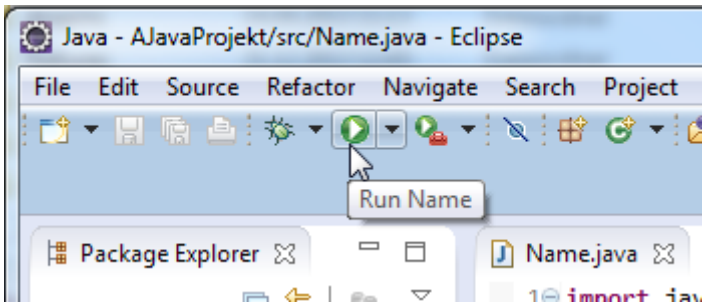
7.3 Starten von Java-Programmen

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.

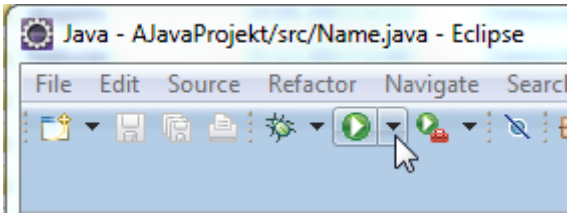
*Name.java

```
1  import java.io.Serializable;
2  import java.util.Scanner;
3
4  public class Name implements Serializable {
5
6      private static final long serialVersionUID = 1L;
7
8      public static void main(String[] args) {
9          System.out.print("Wer da? ");
10         Scanner scanner = new Scanner(System.in);
11         String wer = scanner.next();
12         System.out.println("Moin, "+wer);
13         scanner.close();
14     }
15 }
```

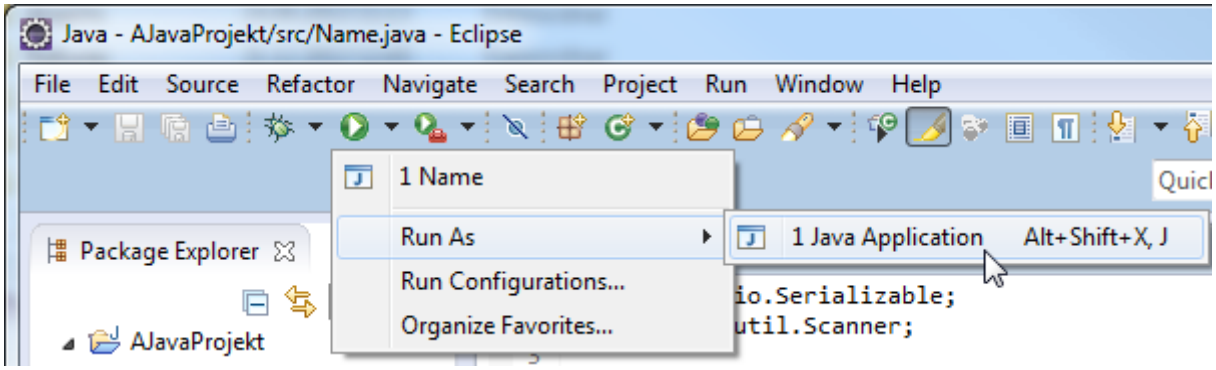
Zum Start wird einfach der Run-Button geklickt, mit dem immer versucht wird, dass Programm mit der im Editor aktiven Klasse zu starten.



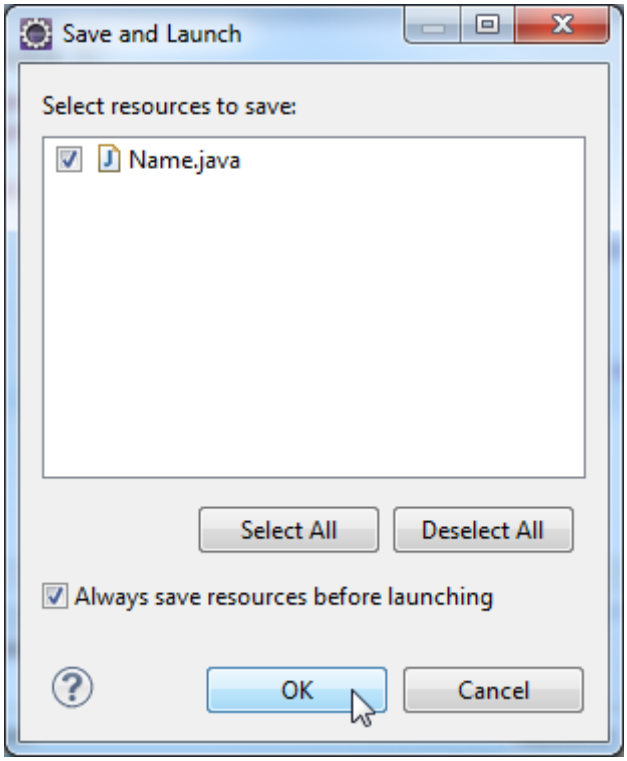
Alternativ und bei mehreren Ausführungsvarianten, muss beim Start der kleine Pfeil nach unten neben dem eigentlichen Ausführungspfeil gedrückt werden.



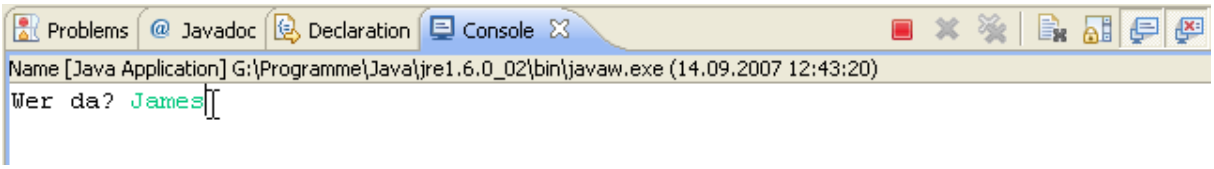
Im einfachsten Fall, der hier zunächst betrachtet wird, wird dann der Start als „Java Application“ ausgewählt. Es ist zu beachten, dass diese Auswahl nur möglich ist, wenn eine Klasse ausgewählt ist, die gerade aktiv im Editor bearbeitet werden kann und eine main()-Methode enthält.



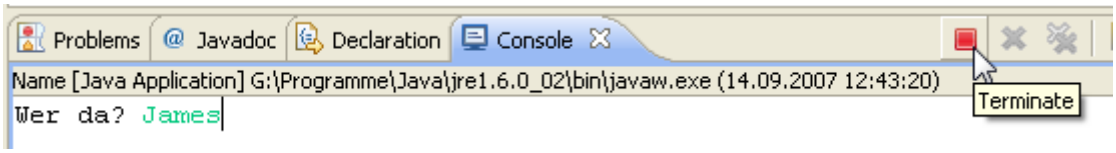
Sollte die Klasse noch nicht gespeichert sein, wird man darauf hingewiesen. Dies lässt sich mit einem Haken links unten für die nächsten Male automatisieren.



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.



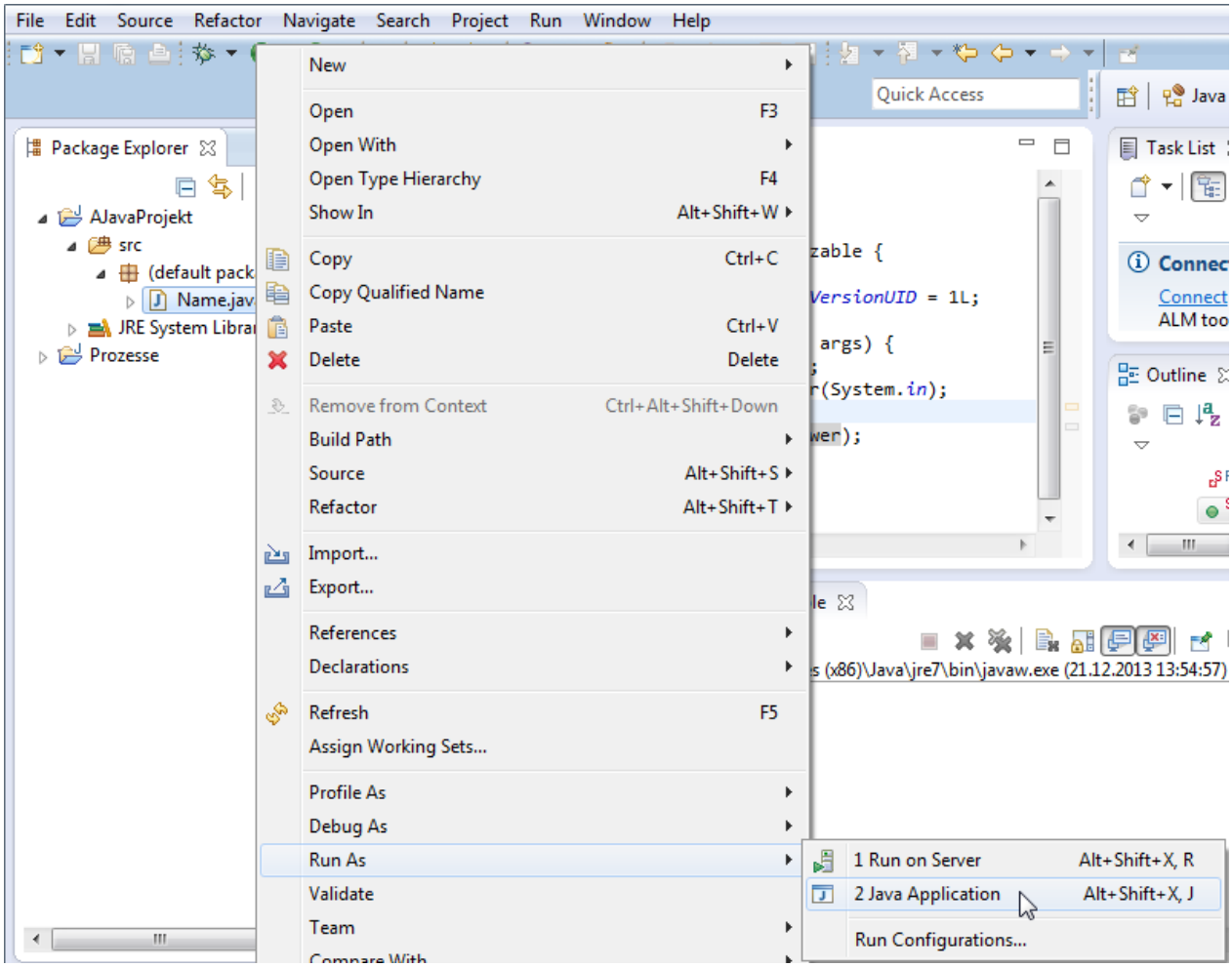
Das rote Quadrat rechts-oben zeigt an, dass dieses Programm noch läuft. Durch einen Klick auf dieses Quadrat könnte das Programm terminiert werden.



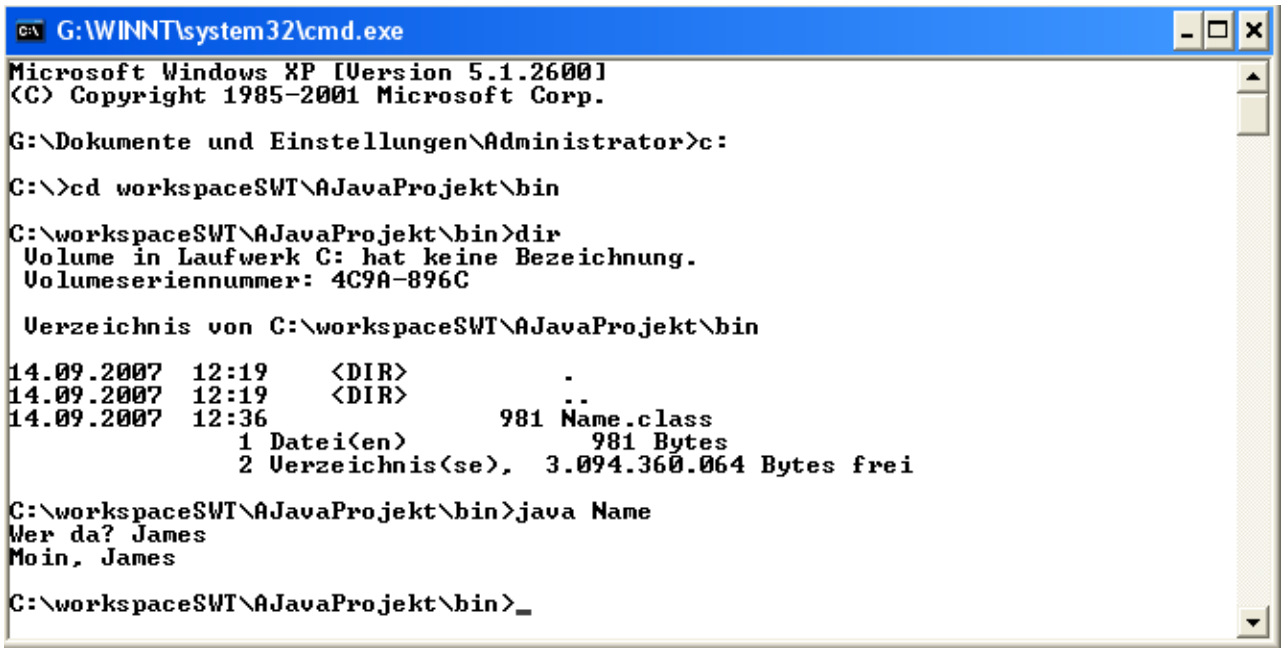
Ausgaben finden ebenfalls im Konsolen-Fenster statt, weiterhin erkennt man an der Information <terminated>, dass das Programm beendet wurde.

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“.

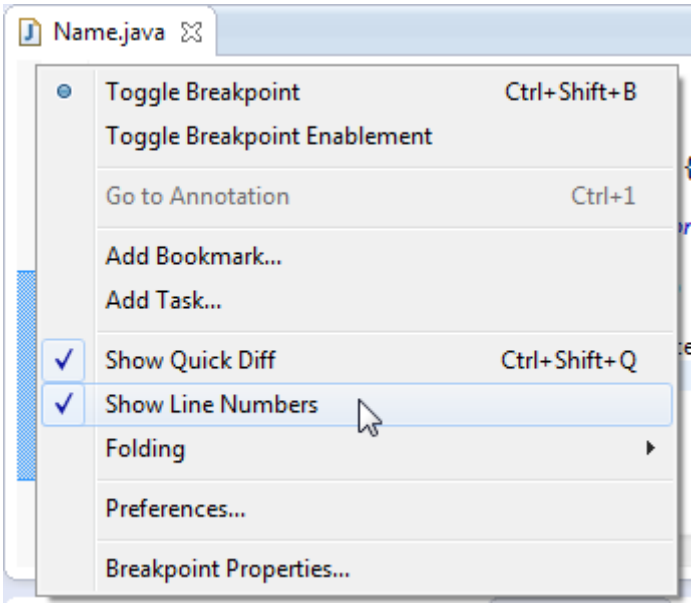


Java-Programme, die innerhalb von Eclipse laufen, können auch außerhalb gestartet werden. Dazu muss man wissen, wo das Projekt gespeichert ist und in das dortige bin-Verzeichnis wechseln.



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.

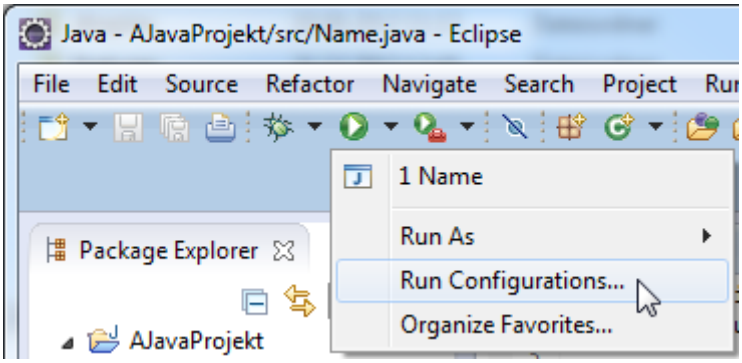


7.5 Parameterübergaben und Assertions einschalten

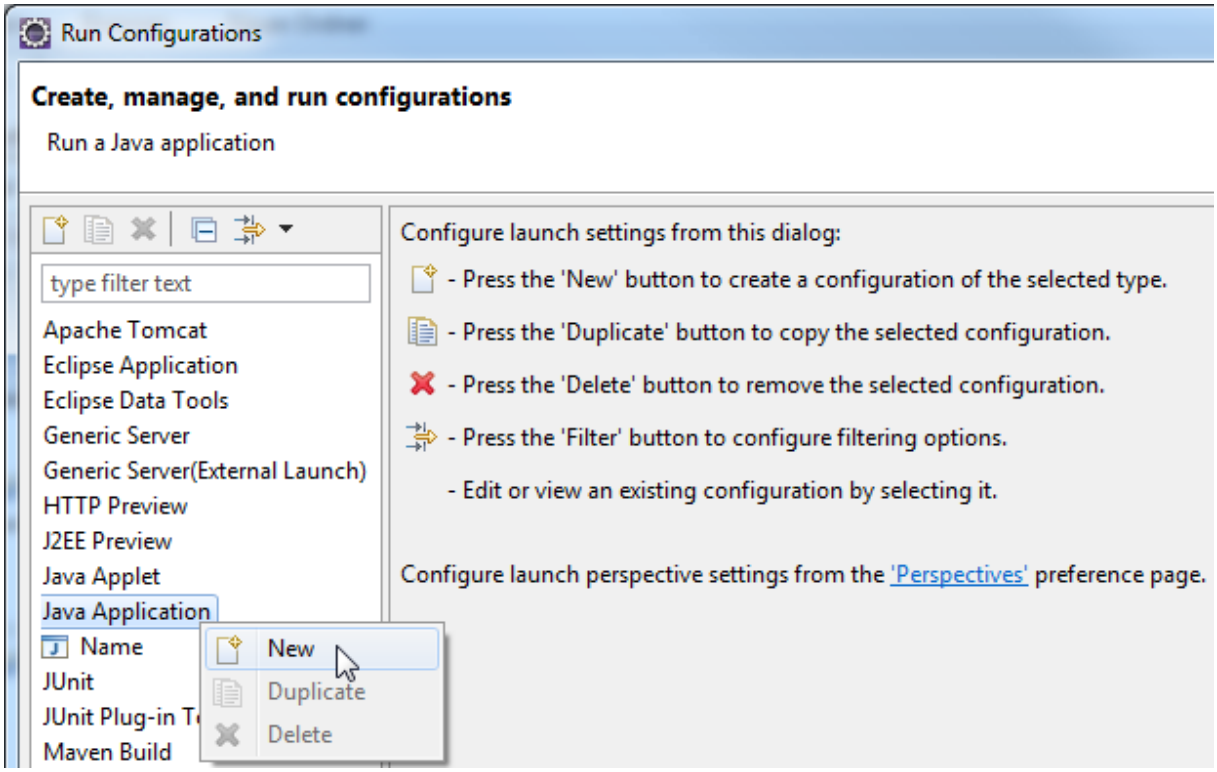
Die Main-Methode wird jetzt wie folgt abgeändert Man beachte, dass es sich hierbei um den Java-Befehl assert handelt und um keine Methode eines Test-Frameworks.

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

}
```

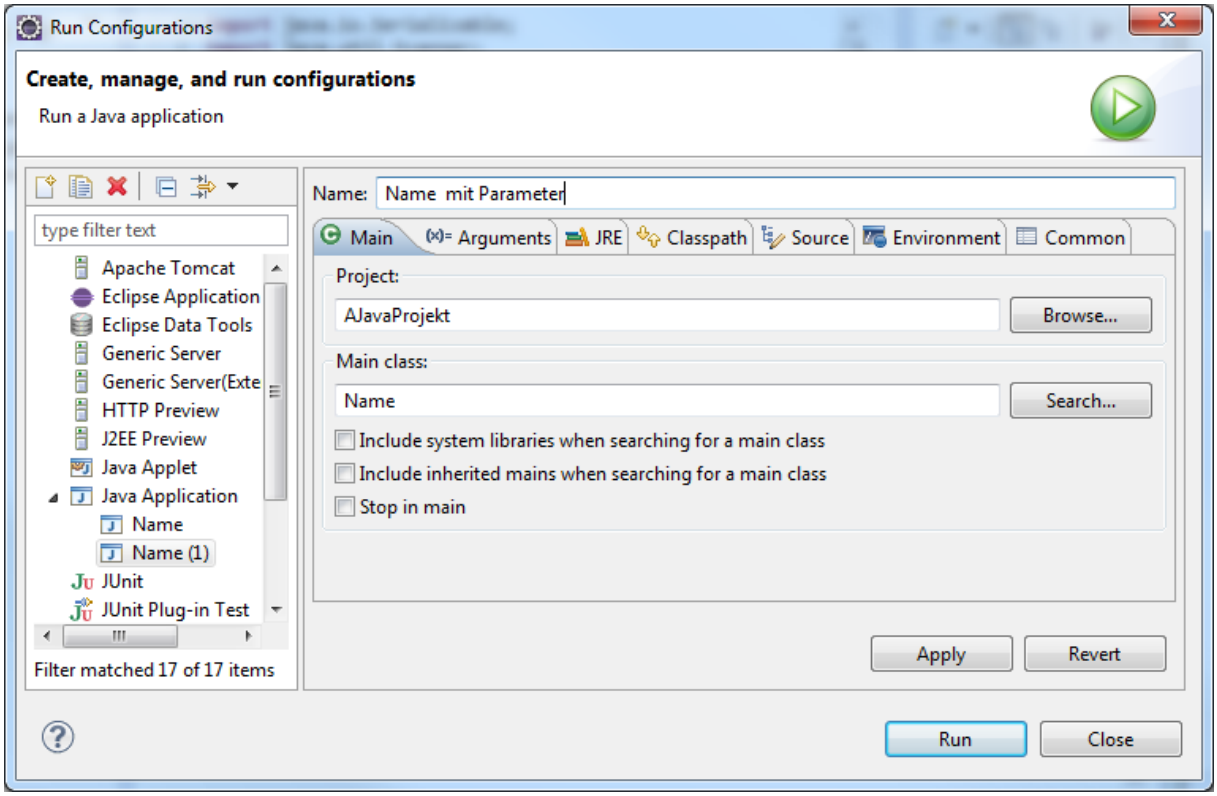
Möchte man in Java Parameter verarbeiten, die eigentlich über die Kommandozeile übergeben wurden, muss man zum Starten zunächst wieder den kleinen Pfeil nach unten neben dem Ausführungspfeil nutzen und „Run Configurations...“ wählen.



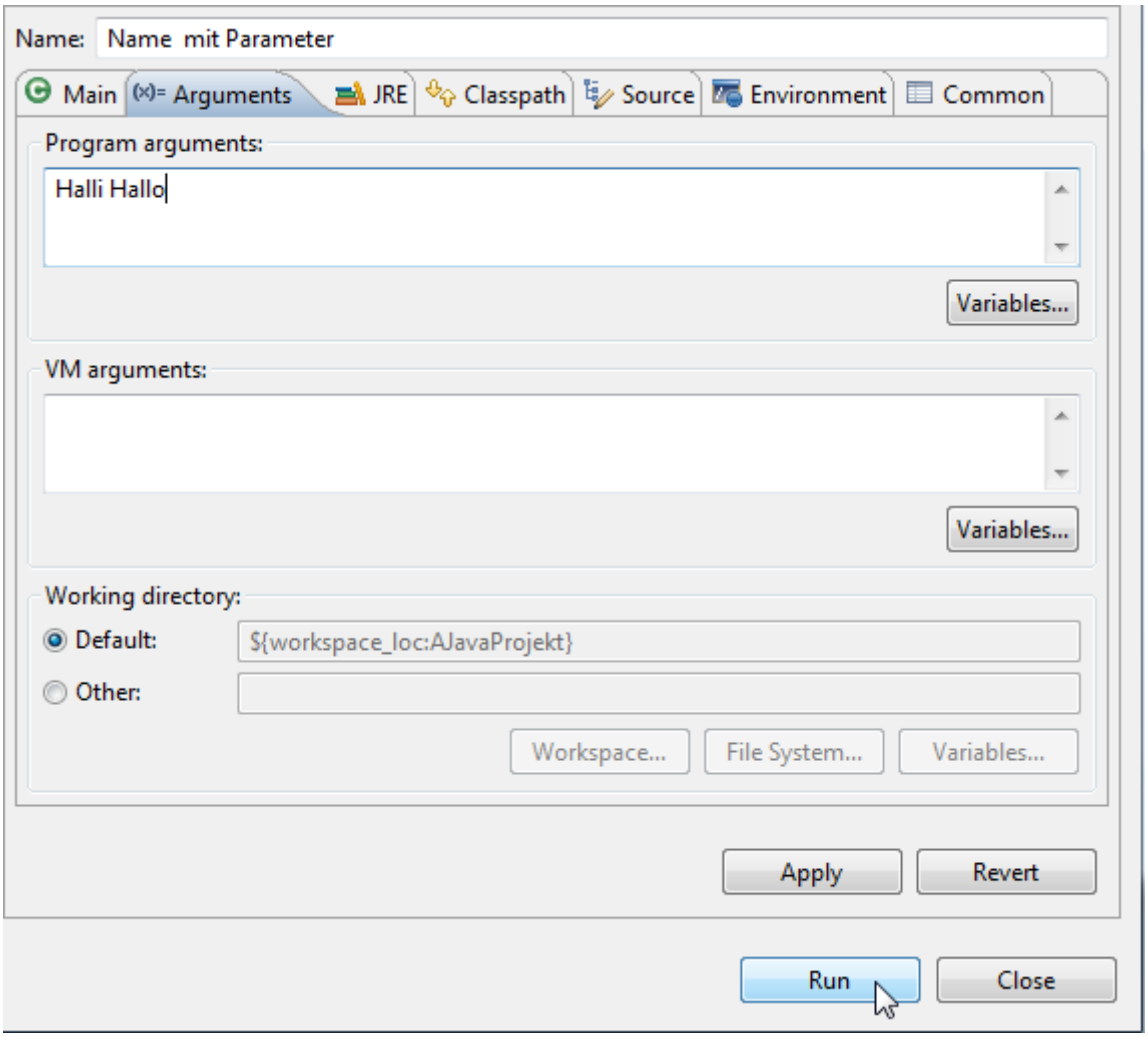
Man erreicht dann die Einstellungsmöglichkeiten, die für einen Programmstart gewählt werden sollen. Diese Einstellungen werden auch *Konfiguration* genannt. Man erkennt unter Java Application, dass es bereits eine Konfiguration für das Programm gibt. Man kann diese entweder bearbeiten, oder zunächst eine Konfiguration, wie hier gezeigt, anlegen. Zunächst wird durch einen Rechtsklick auf „Java Application“ „New“ ausgewählt.



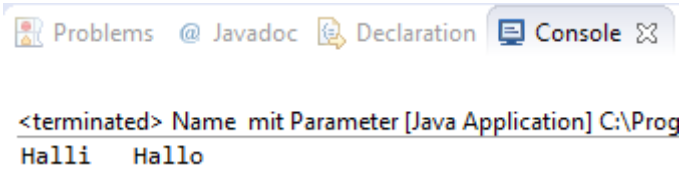
Jeder Konfiguration kann oben ein Name gegeben werden, so dass sie immer wieder genutzt werden kann. Auswählbare Konfigurationen wurden im vorherigen Fenster, dort z. B. nur „Name“, angezeigt.



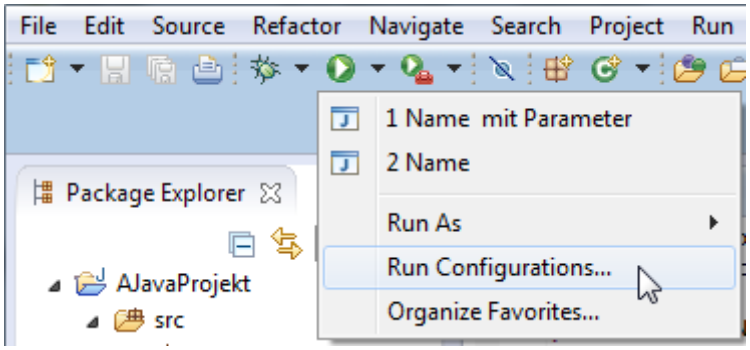
Um die Eingaben von der Kommandozeile anzugeben, gibt es den Reiter „(x)= Arguments“.



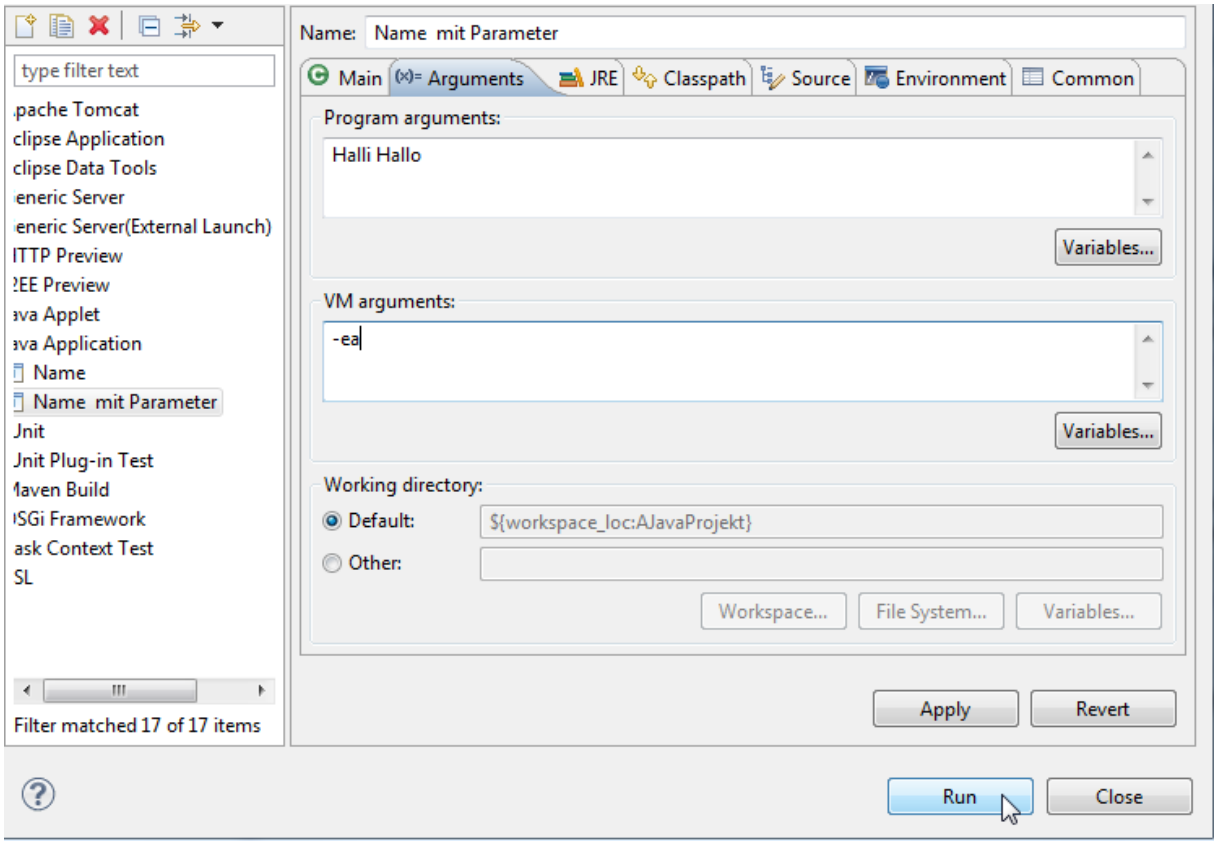
Wird das Programm jetzt mit „Run“ gestartet, erhält man folgendes Ergebnis.



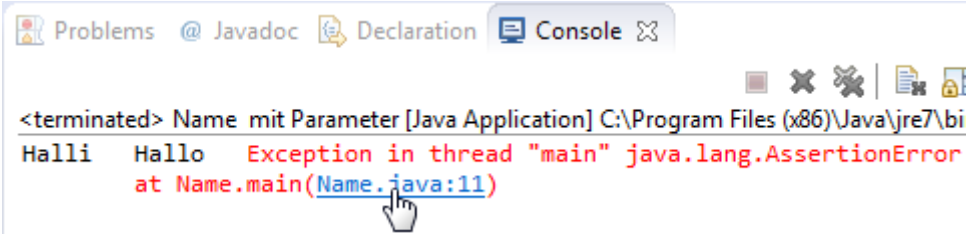
Es fällt auf, dass die Assertion nicht ausgeführt wird. Assertions müssen erst eingeschaltet werden. Dazu wechselt man wieder mit „Open Run Dialog“ zu der jetzt auswählbaren Konfiguration. In der Konfiguration wird dann der virtuellen Maschine mitgeteilt, dass die Assertions eingeschaltet werden sollen (-ea). Dazu wird wieder der kleine Pfeil neben den Run-Button geklickt und „Run Configurations“ ausgewählt. Man sieht oben, dass auch die vorher aufgerufenen Konfiguarationen direkt wählbar sind.



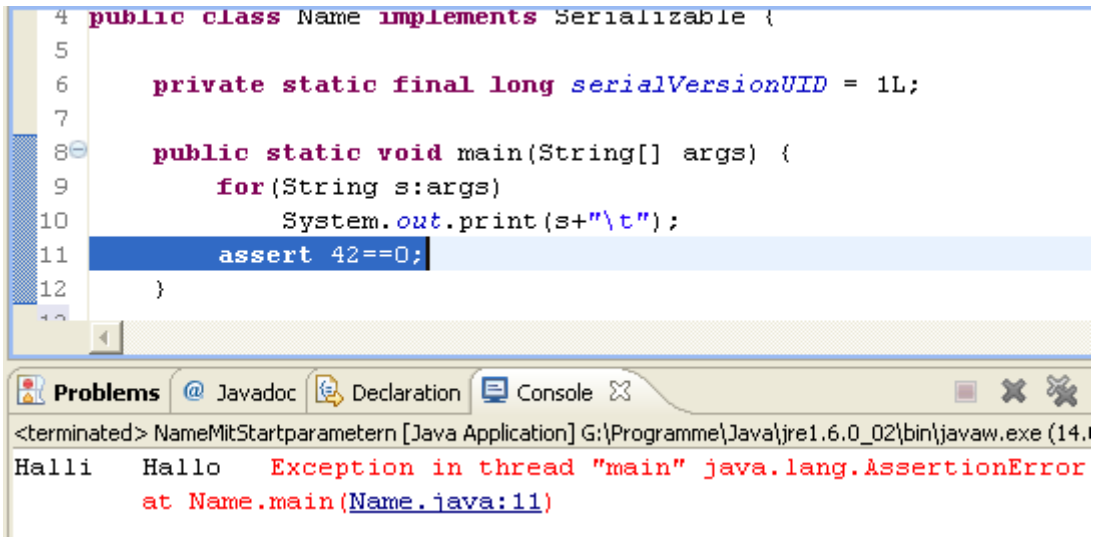
Es rechts die Konfiguration gewählt und es wird wieder der Reiter „(x)= Arguments“ ausgeklickt. Die Option „-ea“ für die virtuelle Maschine unten ergänzt.



Die Ausgabe sieht dann wie folgt aus.

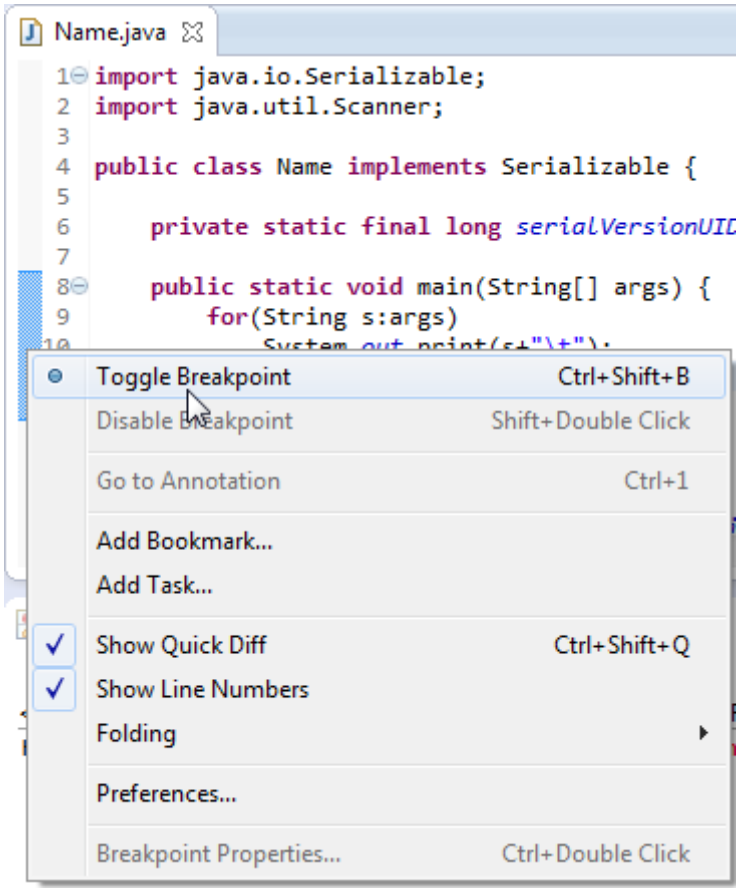


Durch einen Klick auf die markierte Fehlermeldung erreicht man sofort die dazugehörige Fehlerstelle.

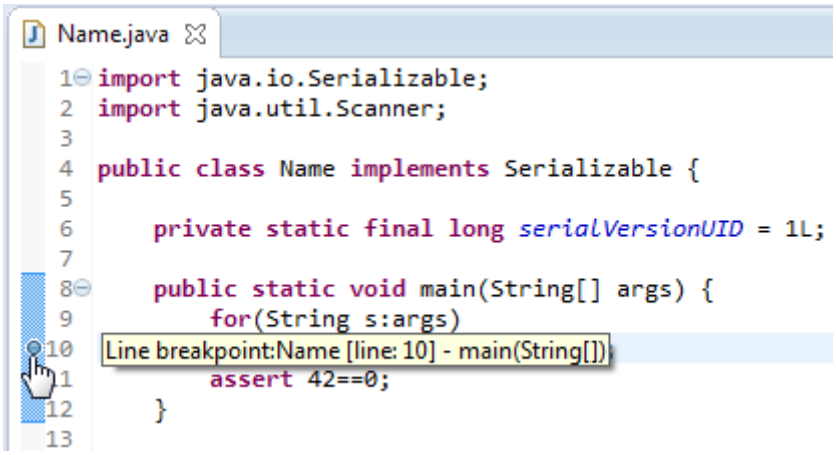


7.6 Kurzeinführung in den Debugger

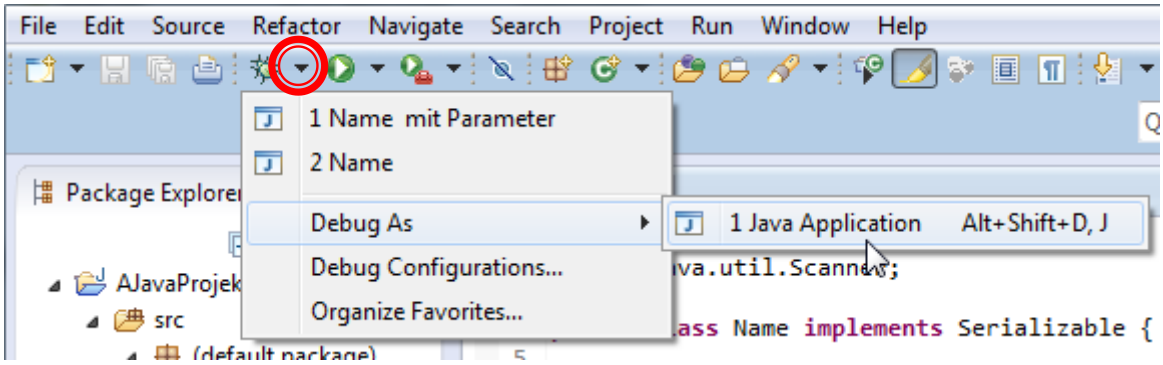
Neben dem bei recht genau identifizierbaren Fehlern sicherlich legitimen Ansatz, bei einem Fehler sich Werte mit einem print auszugeben oder besser einen Logger zu nutzen, besitzt Eclipse einen sehr mächtigen Debugger, mit dem der Programmablauf genauestens verfolgt werden und man jederzeit die genauen 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.



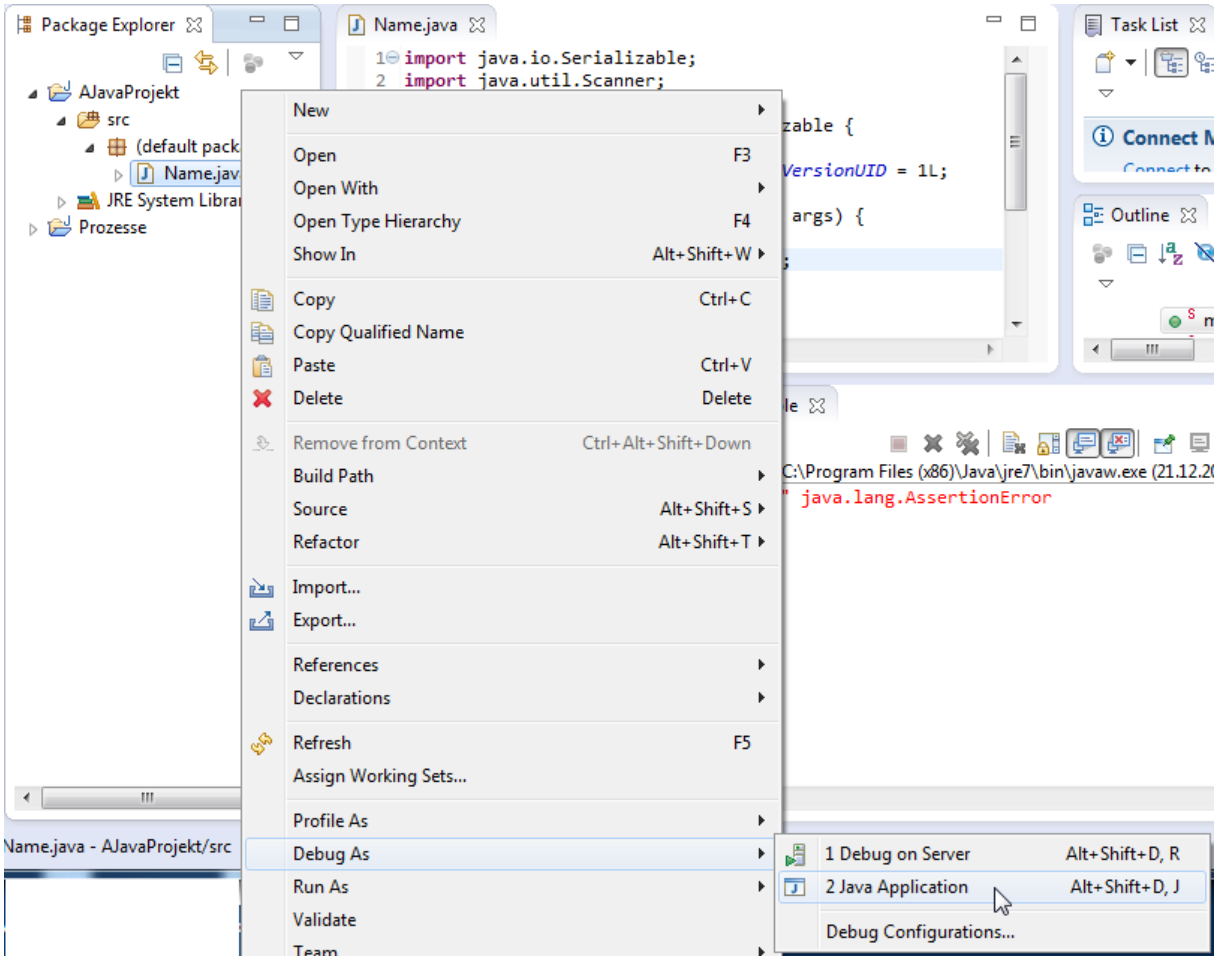
Break-Points sind im Editor links von der Zeile mit einem kleinen Punkt markiert. Schiebt man die Maus auf diesen Punkt, erhält man weitere Informationen.



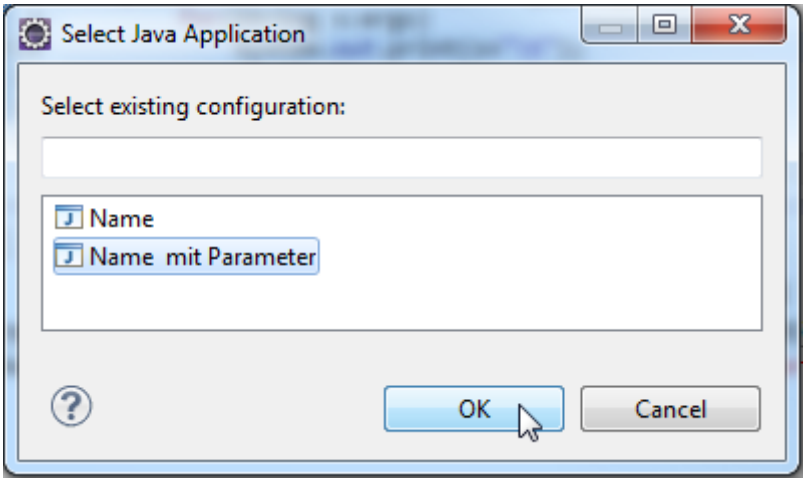
Zum Debuggen muss das Programm beim ersten Mal über den kleinen Pfeil nach unten neben dem Käfer ausgeführt werden.



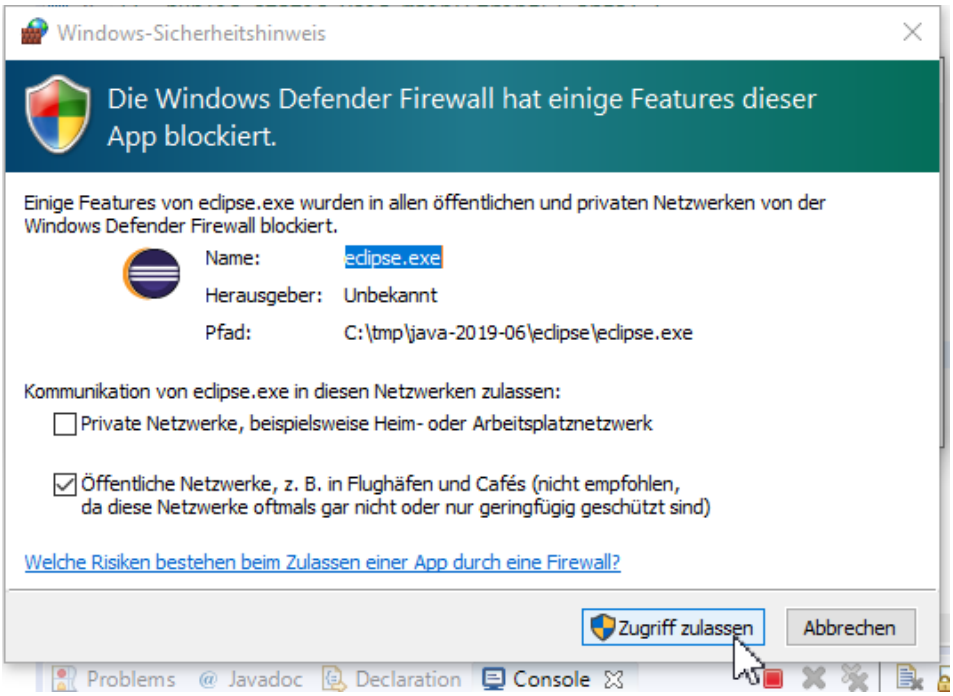
Alternativ kann man wieder über einen Rechtsklick auf der Datei mit „Debug As > Java Application“ den Debugger starten.



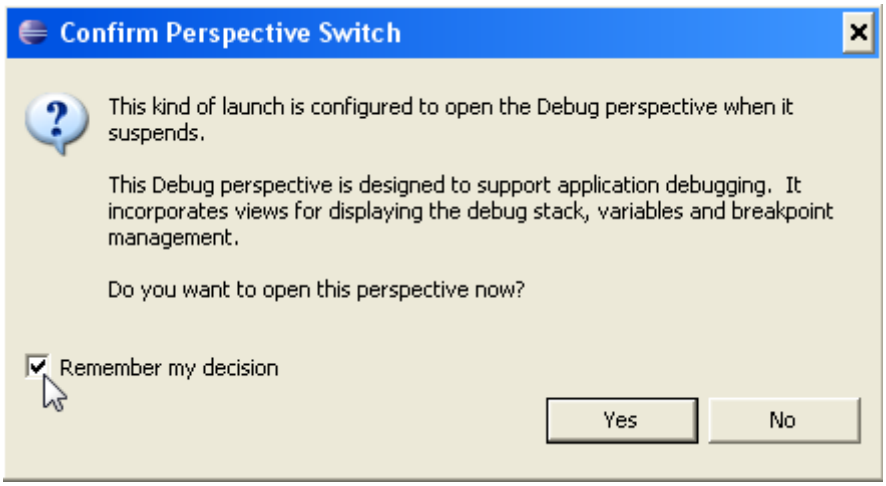
Falls es mehrere Konfigurationen für ein Programm gibt, wird man zur Auswahl einer Konfiguration aufgefordert.



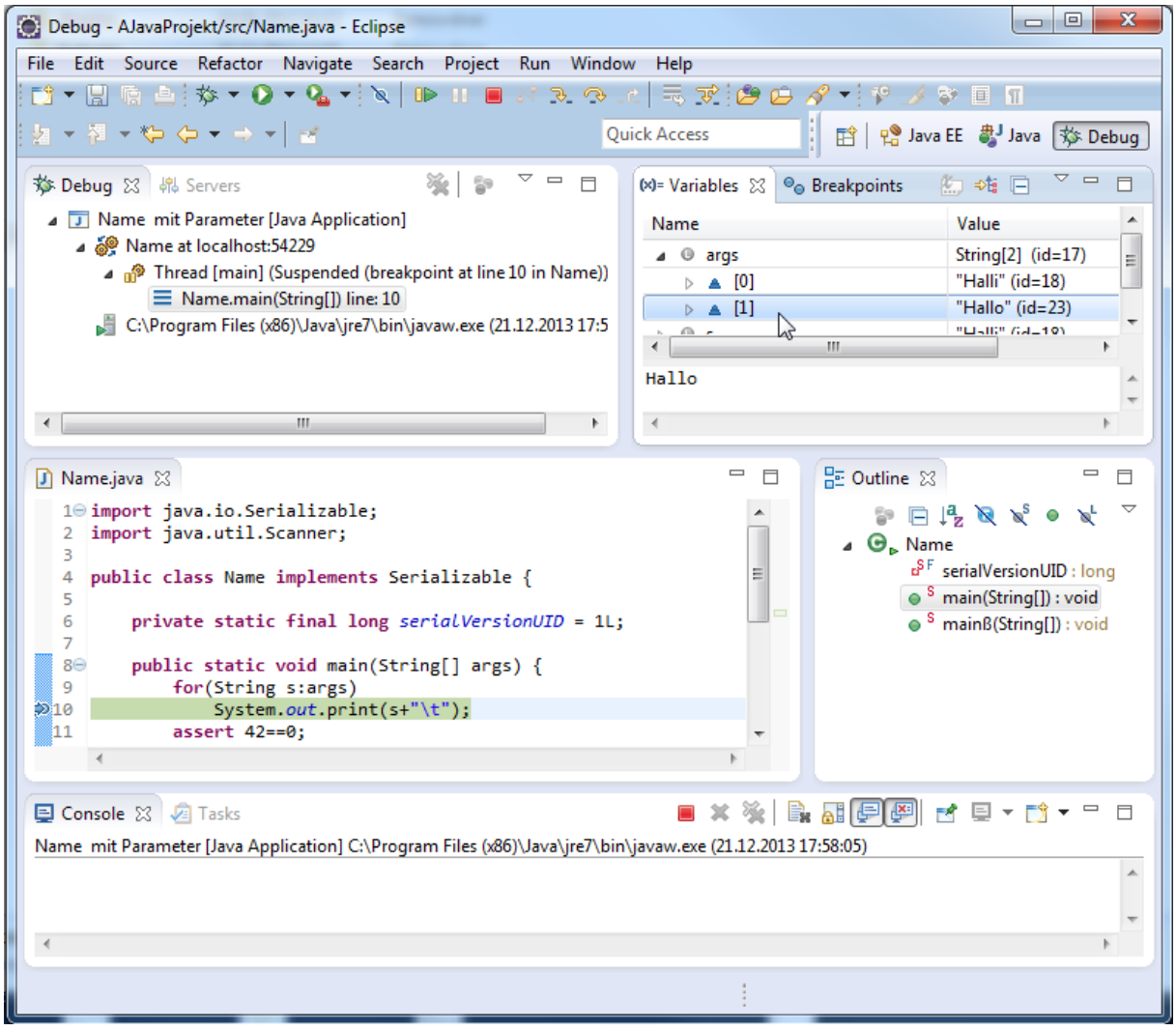
Eventuell muss die Ausführung des Debuggers beim ersten Mal erlaubt werden.



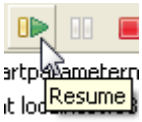
Das Programm hält dann bei der Ausführung beim Breakpoint an und man erhält die folgende Anfrage, die man „für immer“ mit „Remember my decision“ beantworten kann.



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.



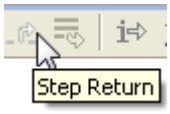
Im Debug-Fenster kann die Programmausführung im Detail gesteuert werden. Es gibt folgende Möglichkeiten.



Die Programmfortsetzung bis zum Erreichen des nächsten Breakpoints.

Der Abbruch des Debugvorgangs.

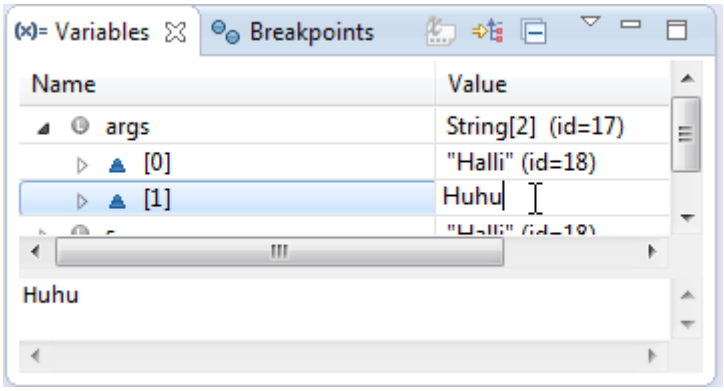
Das Hineingehen in eine aufgerufene Methode, um deren Ablauf zu verfolgen.



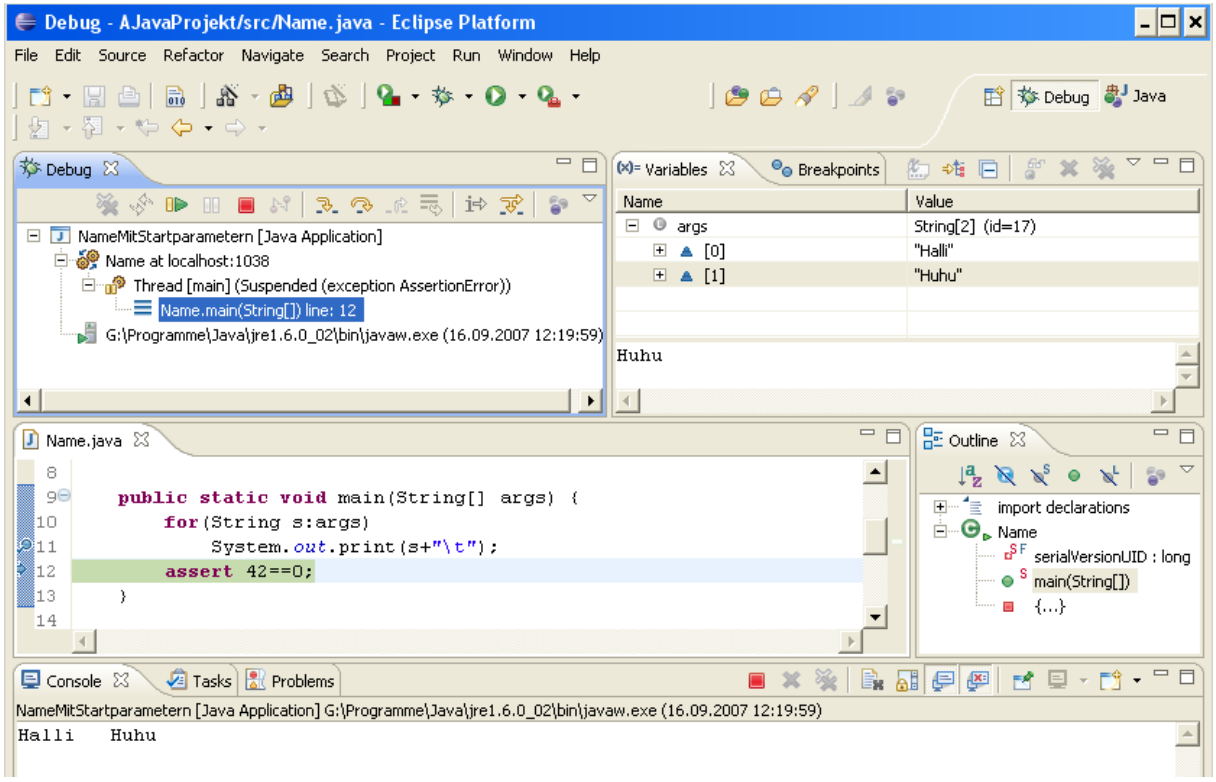
Der Rücksprung aus der aktuell bearbeiteten Methode (die dann abgeschlossen wird). Das Debuggen geht nach dem Methodenaufruf weiter.

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.

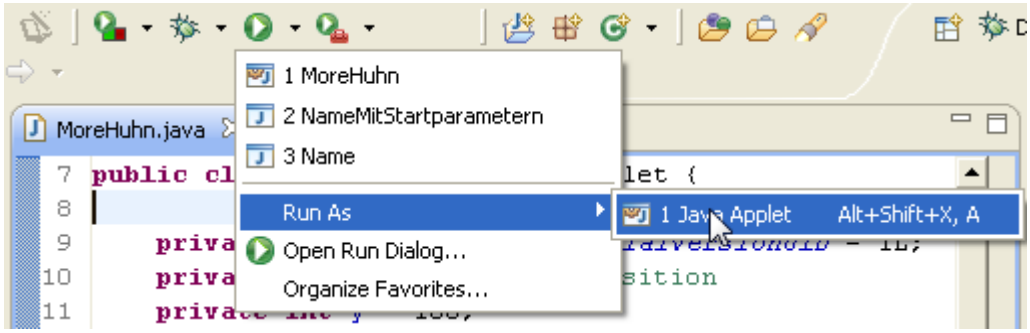


Die Veränderung der Variable wird im Programm berücksichtigt, was folgende Ausgabe zeigt, nachdem zweimal „Resume“ gedrückt wird.

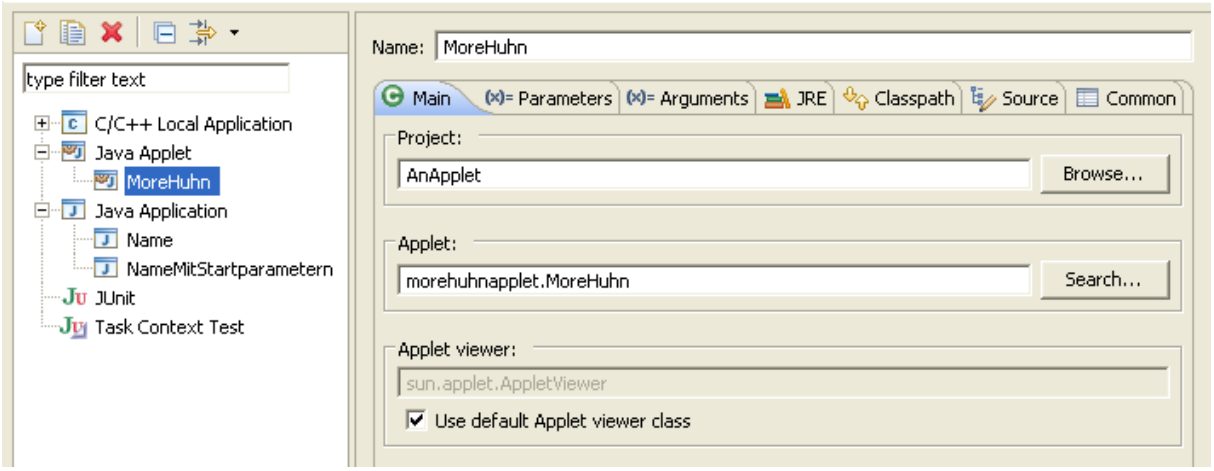


7.7 Starten von Applets (alt)

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.

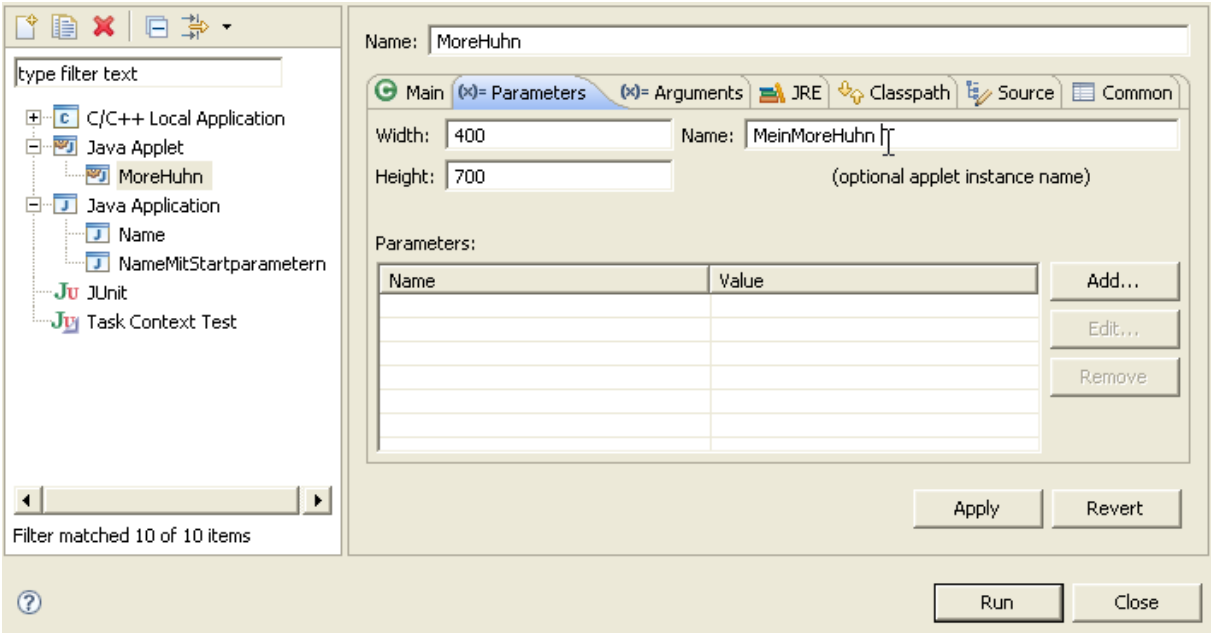


Es wird eine Standardeinstellung genutzt, die zwar änderbar ist, wobei besser die Appletparameter in der Startkonfiguration projektindividuell festgelegt werden. Dazu erfolgt der Start einmal über „Open Run Dialog“.



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.

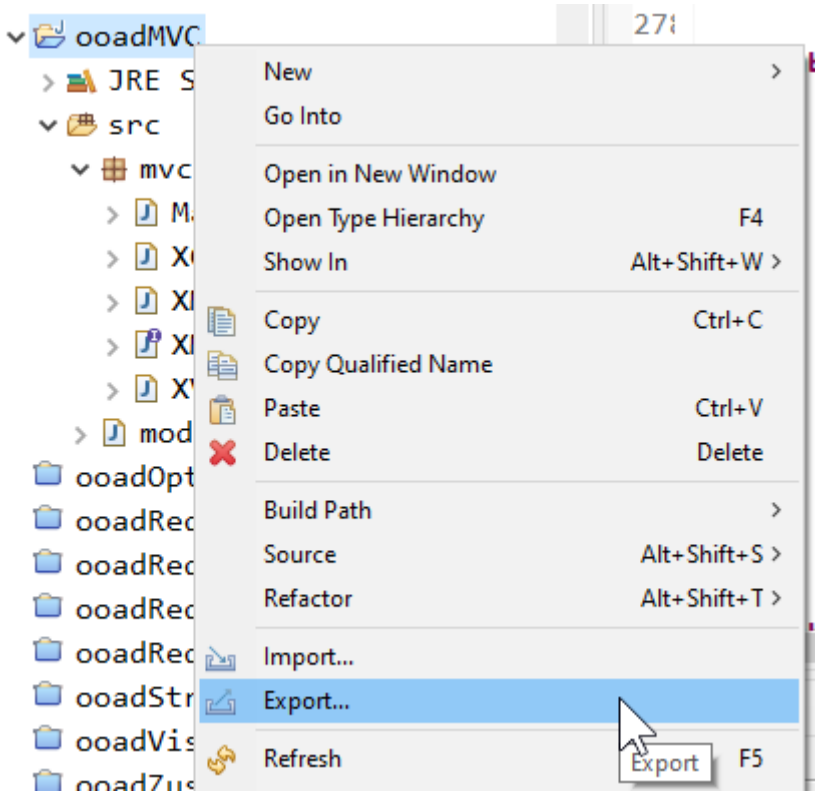
Unter „(x)= Parameters“ kann die Größe des Applets angegeben werden. Weitere Parameter, die auch im HTML-Code stehen können, werden hier mit „Add...“ über die üblichen Parametername/Wert-Paare ergänzt. Das Applet wird über „Run“ gestartet. Diese Einstellungen müssen nur einmal eingegeben werden, danach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



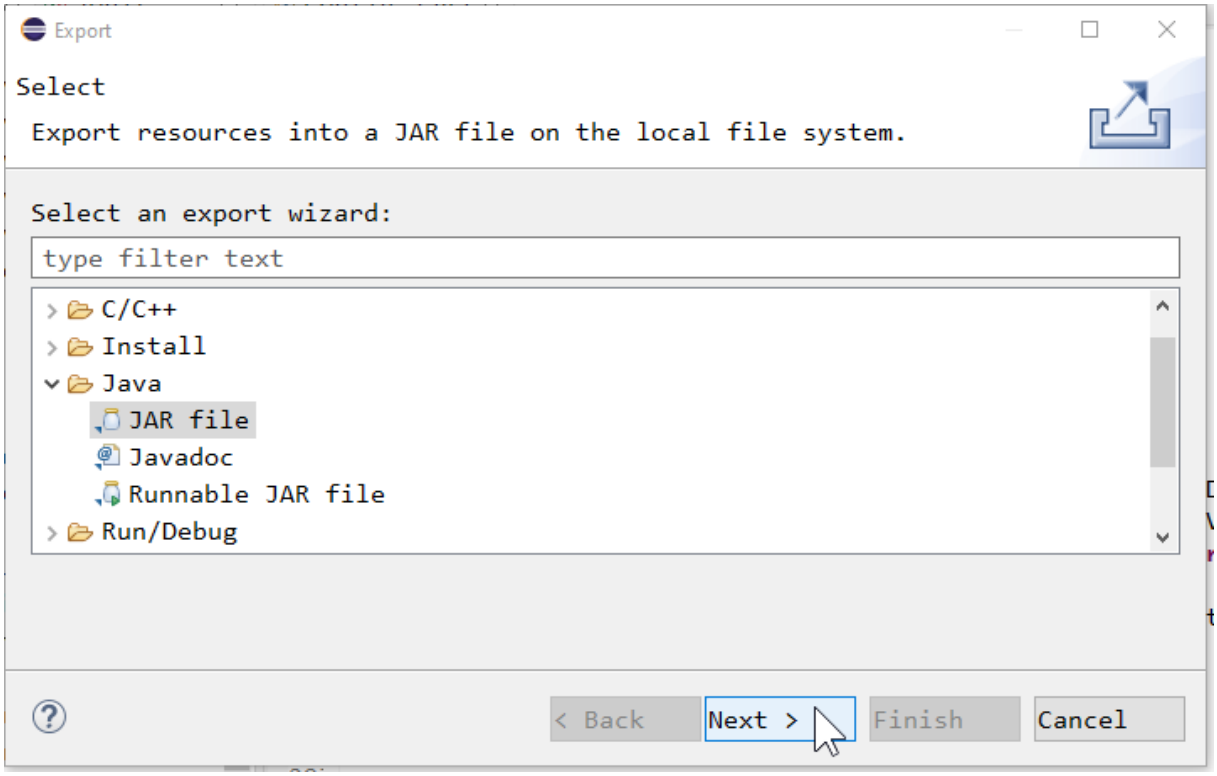
7.8 Packen von JAR-Dateien

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

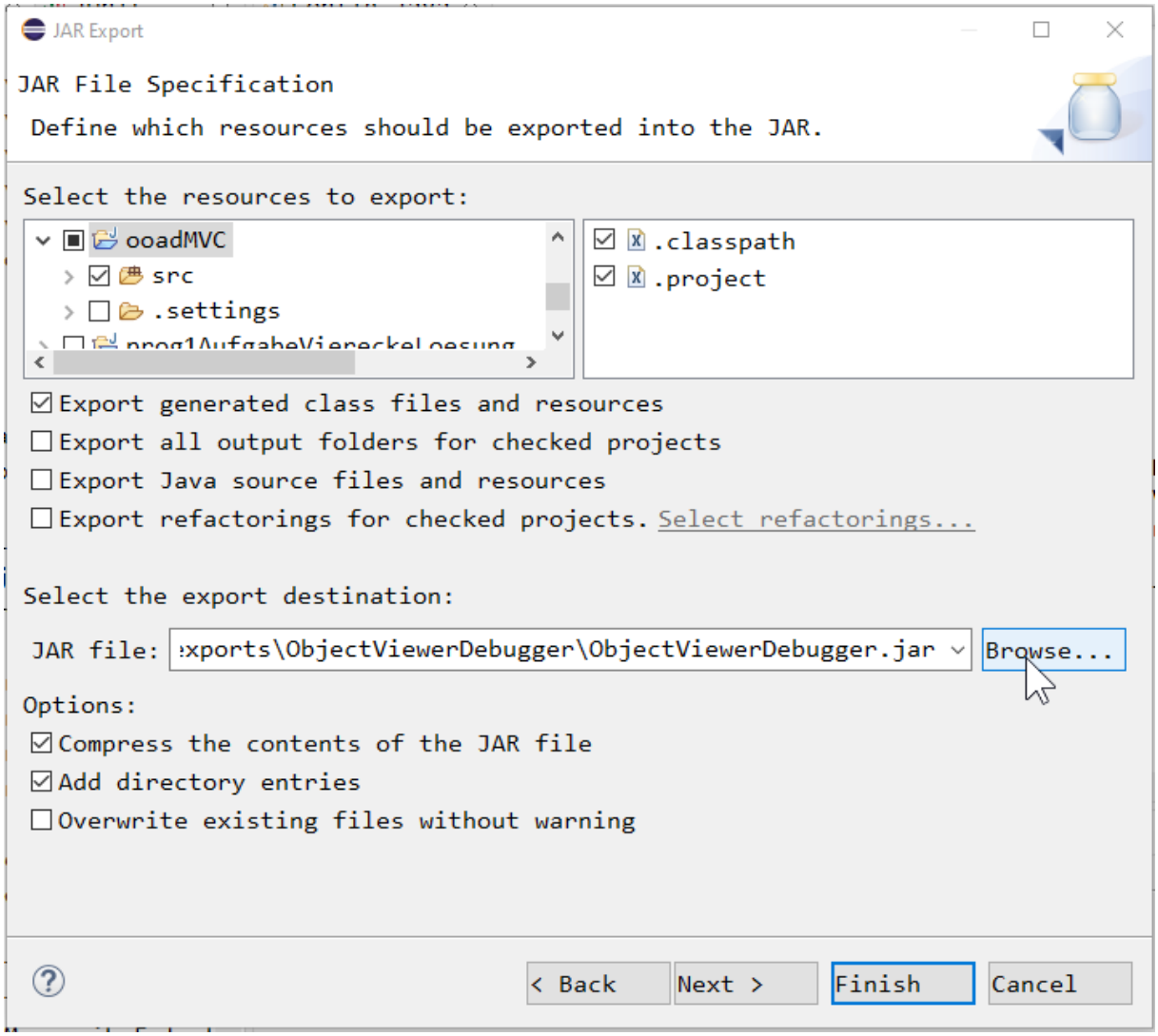
Doppelklick genutzt werden kann, wenn eine Oberfläche gestartet wird. Jar-Dateien werden meist genutzt, um Teil-Software aus anderen Projekten zur Nutzung im aktuellen Projekt auszuliefern. Das Packen beginnt z. B. mit einem Rechtsklick auf dem Projekt, es wird „Export...“ gewählt.



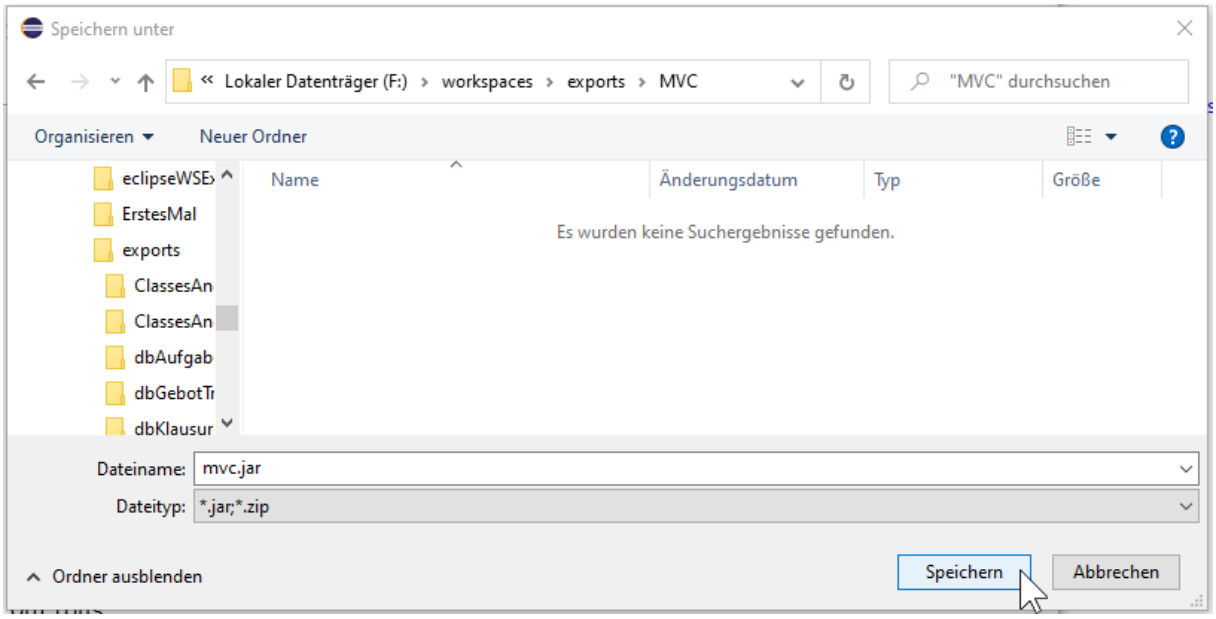
Unter dem Punkt „Java“ dann „JAR file“ auswählen und „Next>“ drücken, bzw. „Runnable JAR File“ falls das Programm alleine lauffähig sein soll. Hier wird zunächst das einfache JAR file und dann das „Runnable JAR file“ betrachtet.



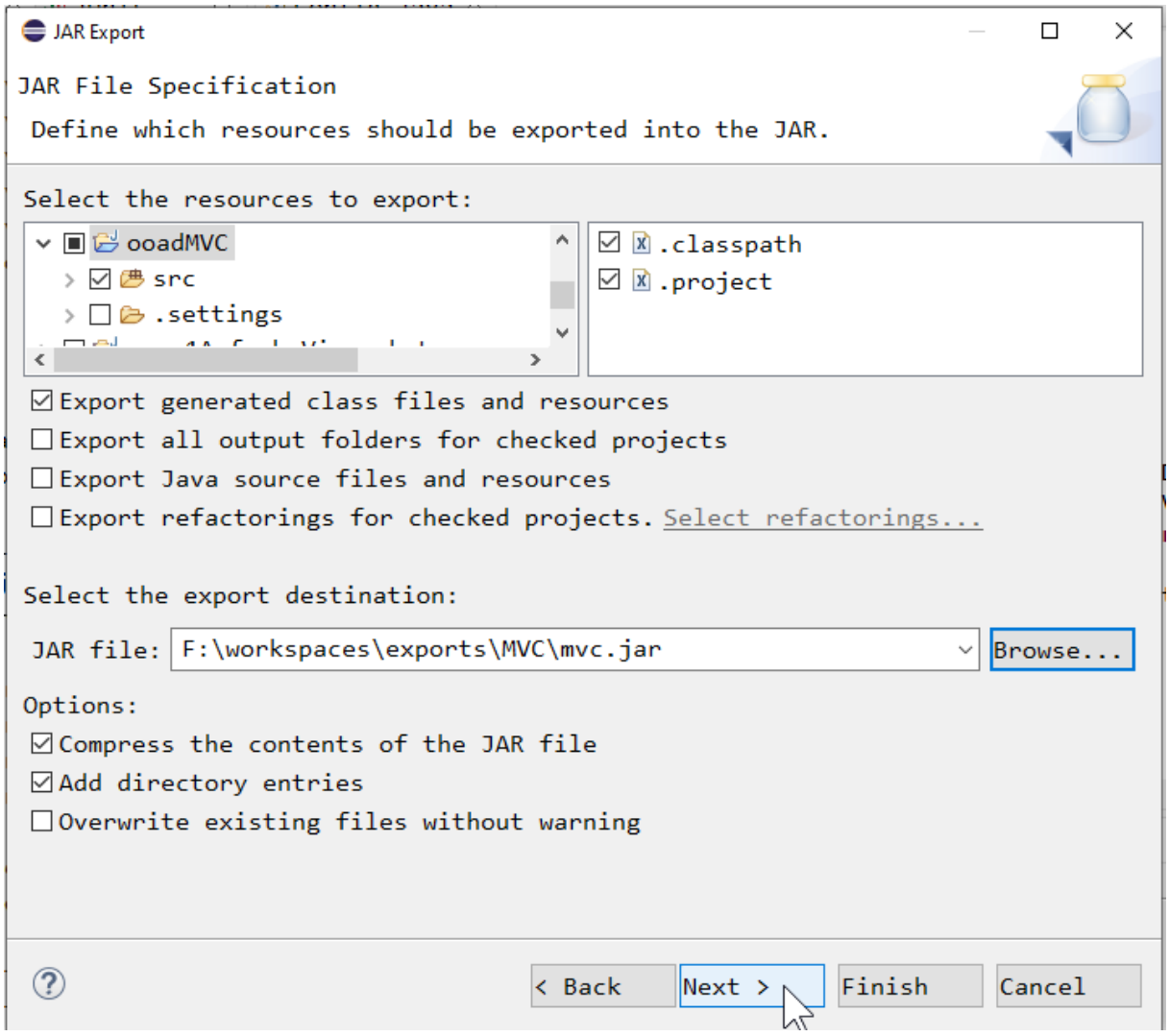
Danach können einige meist selbsterklärende Einstellungen vorgenommen werden. Im oberen Teil des Fensters werden die Dateien zum Packen ausgewählt. Dabei wird bei Endversionen auf die mit einem Punkt beginnenden Einstellungsdateien von Eclipse meist verzichtet. In der Auswahl darunter kann man festlegen, ob nur die ausführbaren Dateien, oder auch der Source-Code mit eingepackt werden soll. Soll es um das nutzbare oder ausführbare Programm gehen, ist der erste Auswahlkasten sinnvoll. Dann wird der Speicherort für die Jar-Datei mit Hilfe des „Browse…“-Knopfs festgelegt.



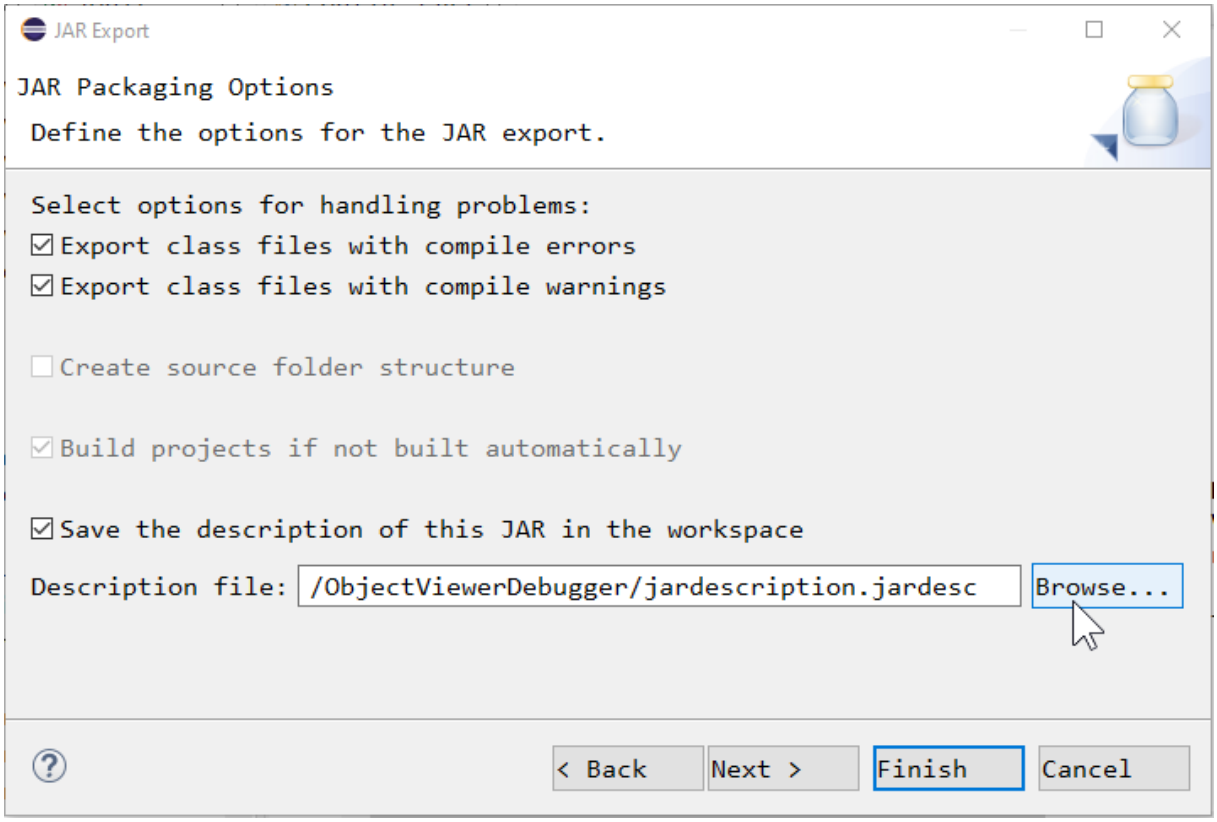
Dieser Speicherort muss außerhalb des Eclipse-Workspaces, zumindest des Projekts liegen, da es sonst Probleme mit Eclipse gibt, die Erstellung bei einer späteren Aktualisierung erneut zu nutzen. Die zu erstellende Jar-Datei soll auch die Endung „.jar“ haben. Am Ende wird „Speichern“ geklickt.



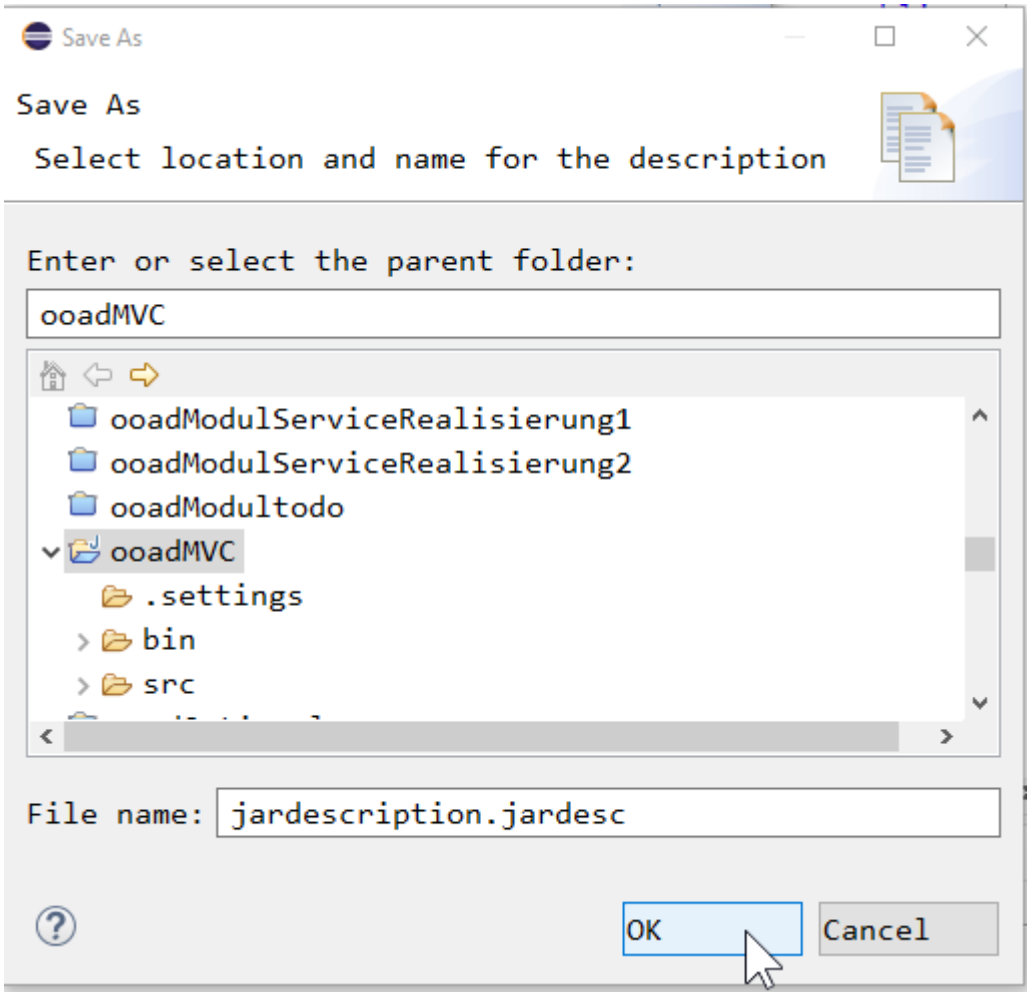
Für weitere Einstellungen wird „Next >“ geklickt.



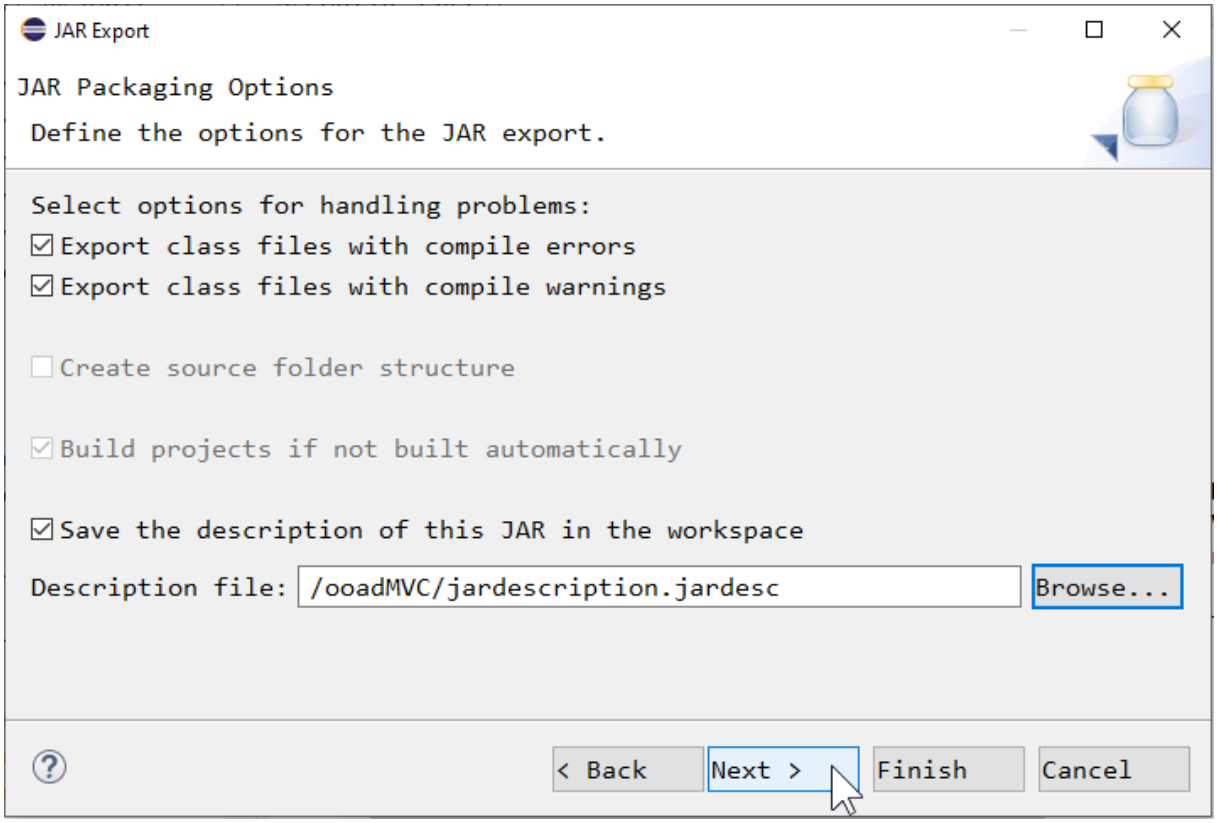
Um sich die Einstellungen bei einer erneuten Erzeugung zu merken, wird „Save the Description …“ angeklickt und dann „Browse…“ gedrückt.



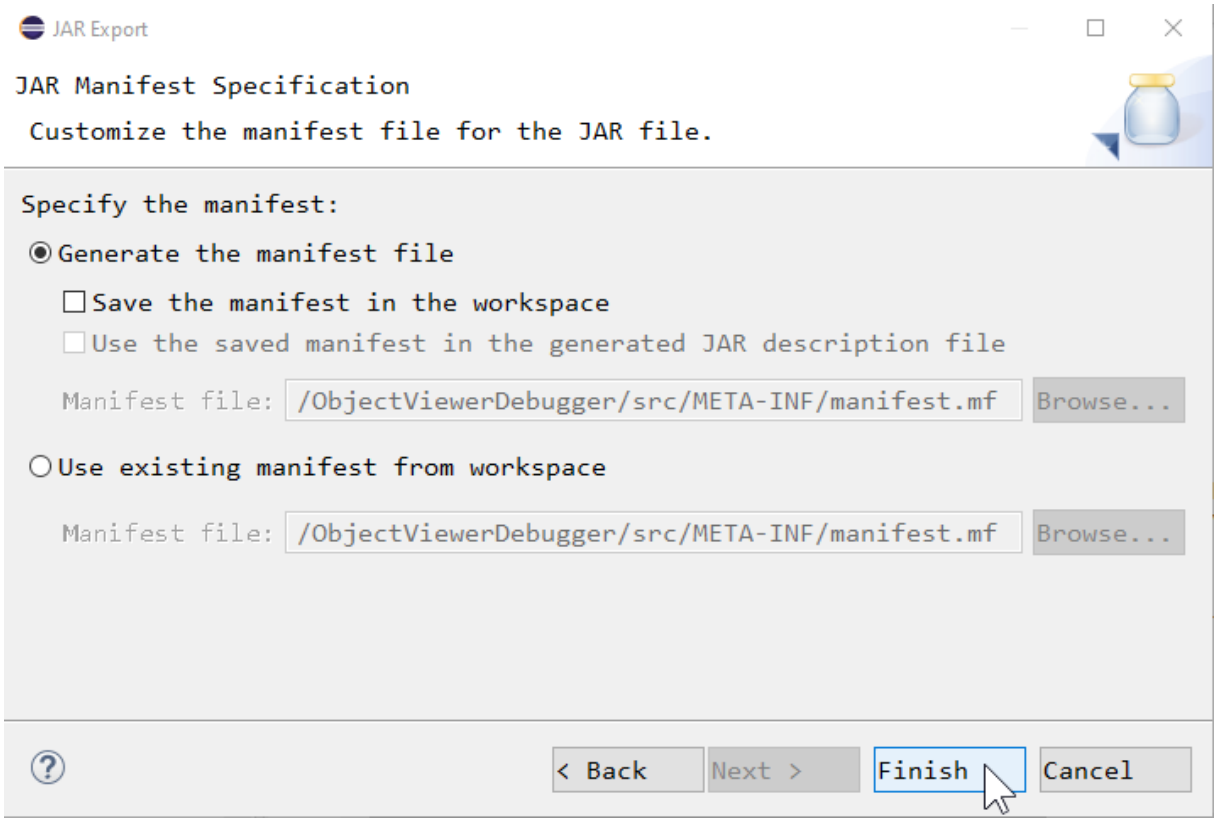
Es wird zum eigentlichen Projekt gesteuert und in dem Verzeichnis eine Datei mit der Endung „.jardesc“, hier „jardescription.jardesc“, abgespeichert.



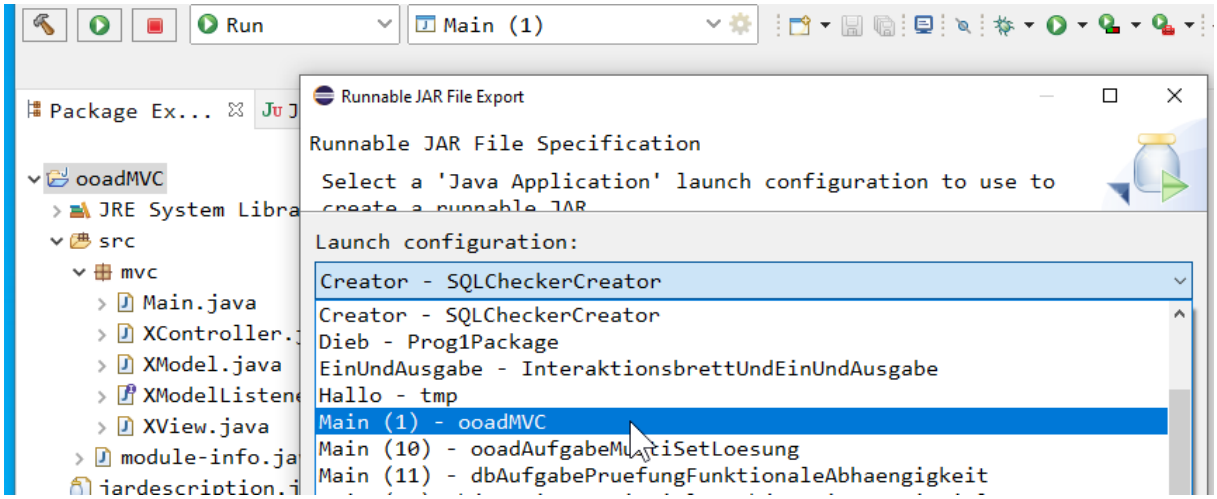
Es wird „Next >“ geklickt.



In einer sogenannten Manifest-Datei kann festgelegt werden, welche Libraries und zusätzliche Ressourcen genutzt werden, weiterhin ist eine Start-Datei angebbar, die hier aber keine Rolle spielt. Handelt es sich um einfaches Projekt, kann das Manifest generiert und die Erstellung mit „Finish“ abgeschlossen werden. Die Jar-Datei ist damit wie jede andere Bibliothek (Library) nutzbar.



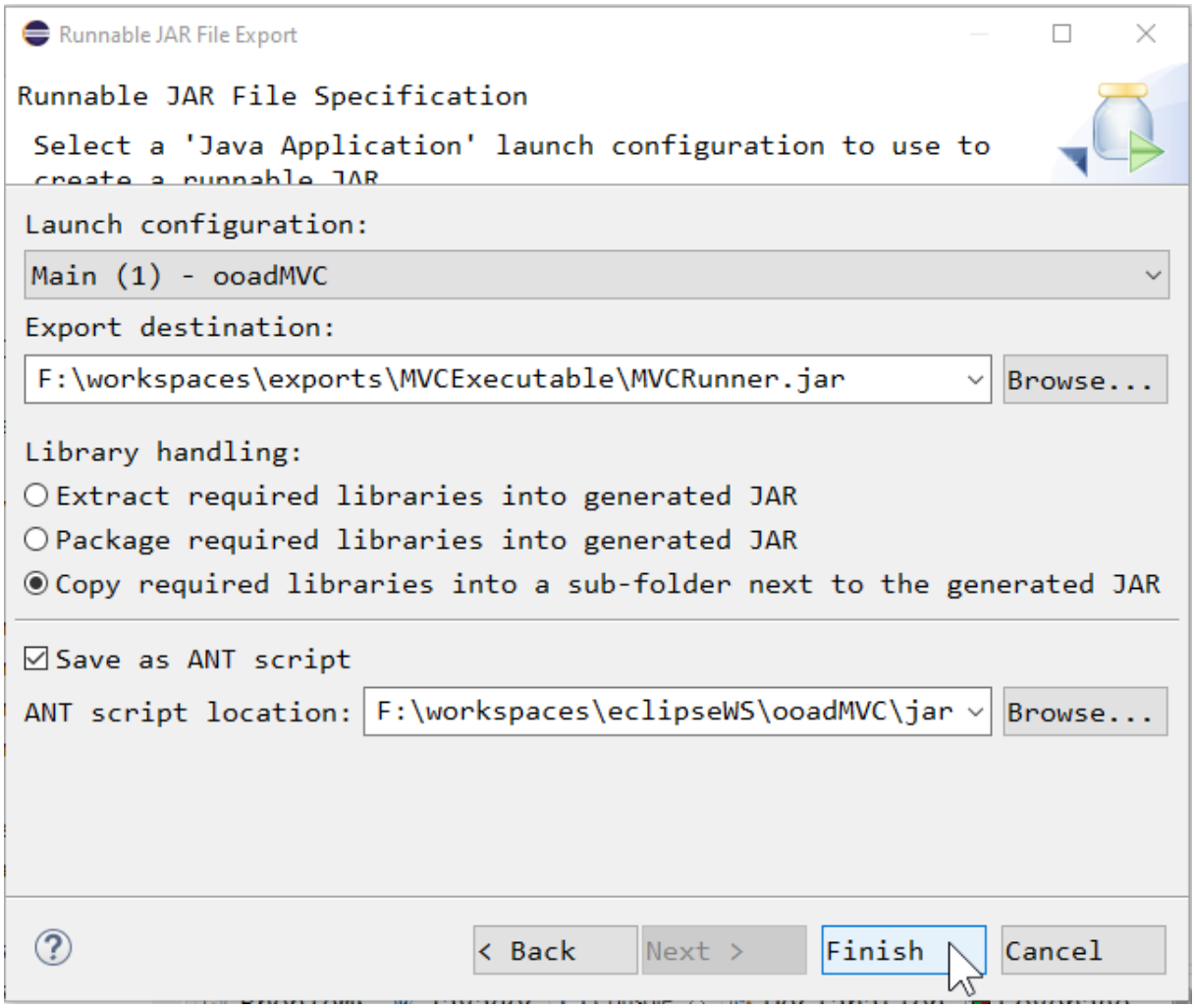
Die Erstellung einer ausführbaren Jar-Datei ist ähnlich. Es ist dabei sinnvoll das Programm zunächst einmal in der später gewünschten Form, also z. B. mit Aufrufparametern, zu starten. Dadurch wird diese sogenannte Run-Konfiguration die oberste in der Auswahl der „Lauch configuration“ unter „Main“, falls der Konfiguration kein Name gegeben wurde..



Wieder wird unter „Browse…“ das Ziel des Exports eingestellt. Weiterhin muss festgelegt werden, wie mit genutzten Bibliotheken umgegangen wird. Mit „Extract“ werden alle jar-Dateien ausgepackt und es entsteht ein Verzeichnisbaum mit „.class“-Dateien, mit „Package“ werden die „Jar-Dateien“ mit in die entstehende Jar-Datei eingebunden. Beide Wege sind bequem in der Nutzung, oftmals aber aus lizenzrechtlichen Gründen nicht nutzbar. Dazu

müssen die Lizenzen der genutzten Bibliotheken analysiert werden. Beim dritten Weg „Copy“ wird ein Zusatzverzeichnis angelegt, in dem sich dann die genutzten Bibliotheken unverändert als Jar-Dateien befinden. Auch hier kann es lizenzrechtliche Probleme in allerdings selteneren Fällen geben. Die Jar-Datei muss dann später zusammen mit dem Verzeichnis, das genau diesen Namen behalten muss, ausgeliefert werden.

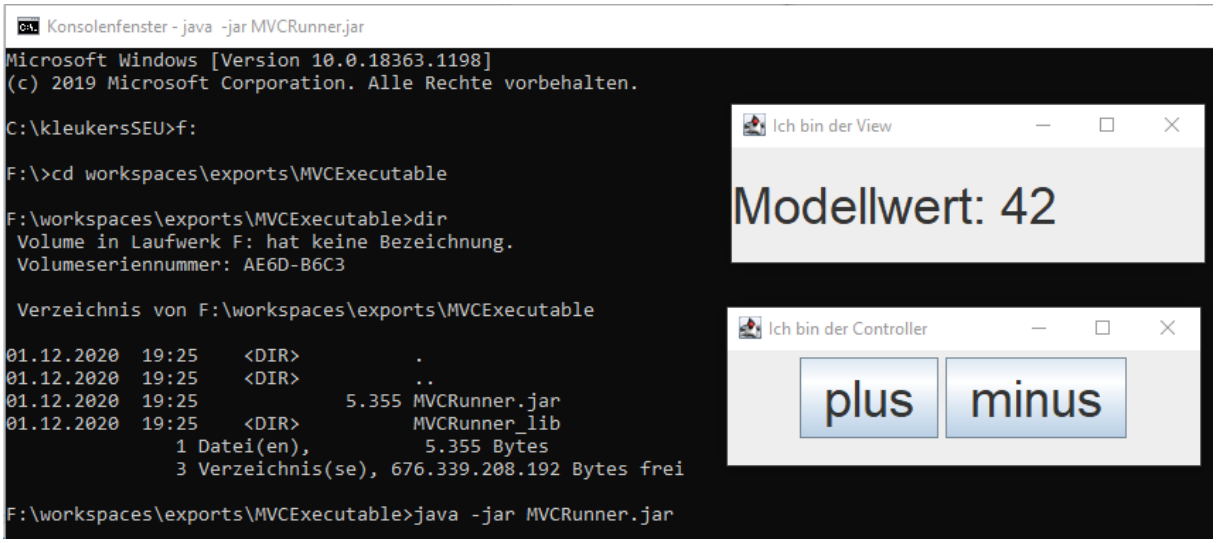
Um die spätere Wiederholung der Generierung zu vereinfachen, ist im Projekt ein „ANT script“ abzulegen. Die Erstellung wird mit „Finish“ abgeschlossen.



Die Jar-Datei ist z. B. dann mit

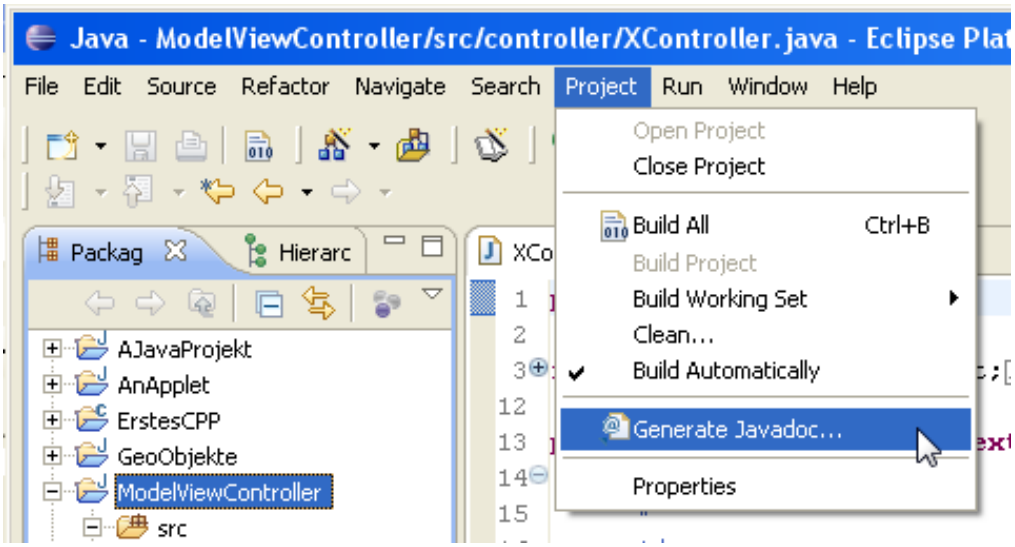
```
java -jar <DateinameMitEndung>
```

im Verzeichnis aufrufbar und kann flexibel weitergegeben werden.



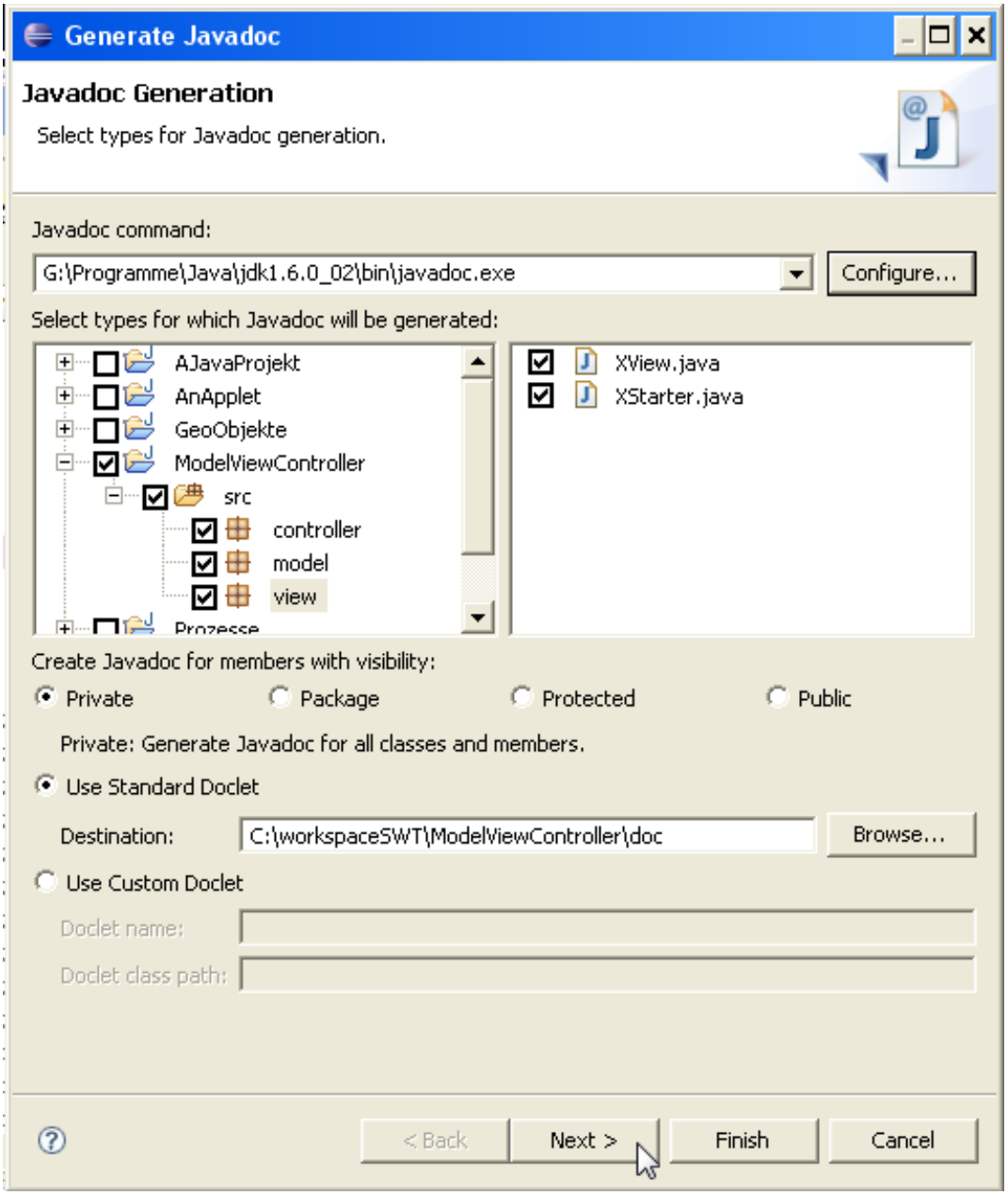
7.9 Dokumentengenerierung

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.

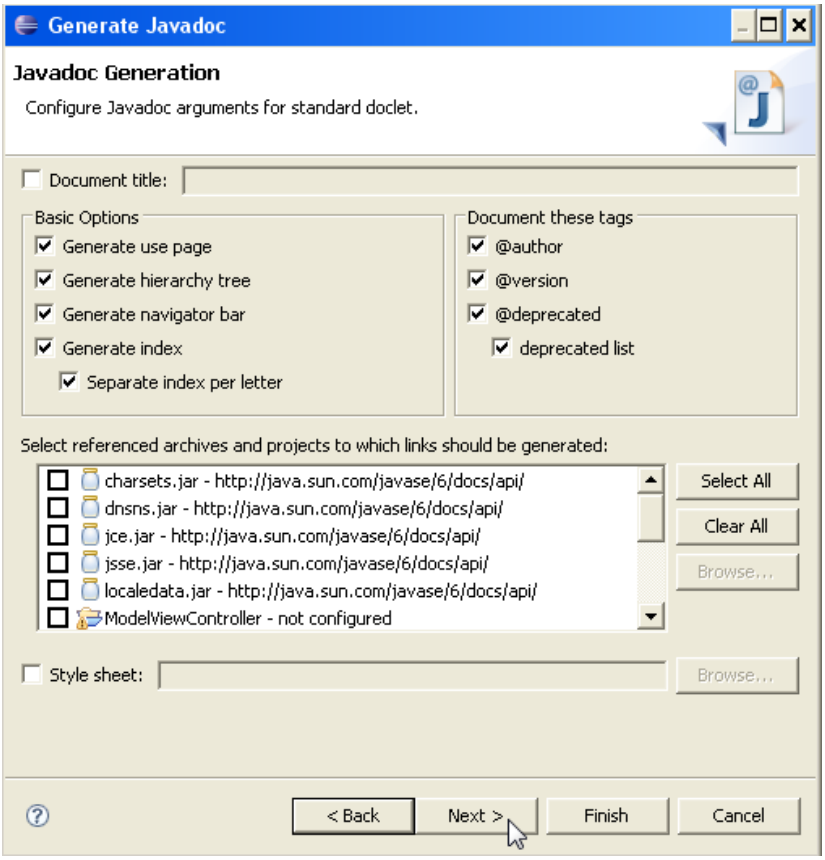


Etwas abhängig von der Java-Installation, muss in der obersten Zeile der Pfad zum genutzten javadoc-Programm eingestellt werden, der meist aber bereits passend angegeben wird.

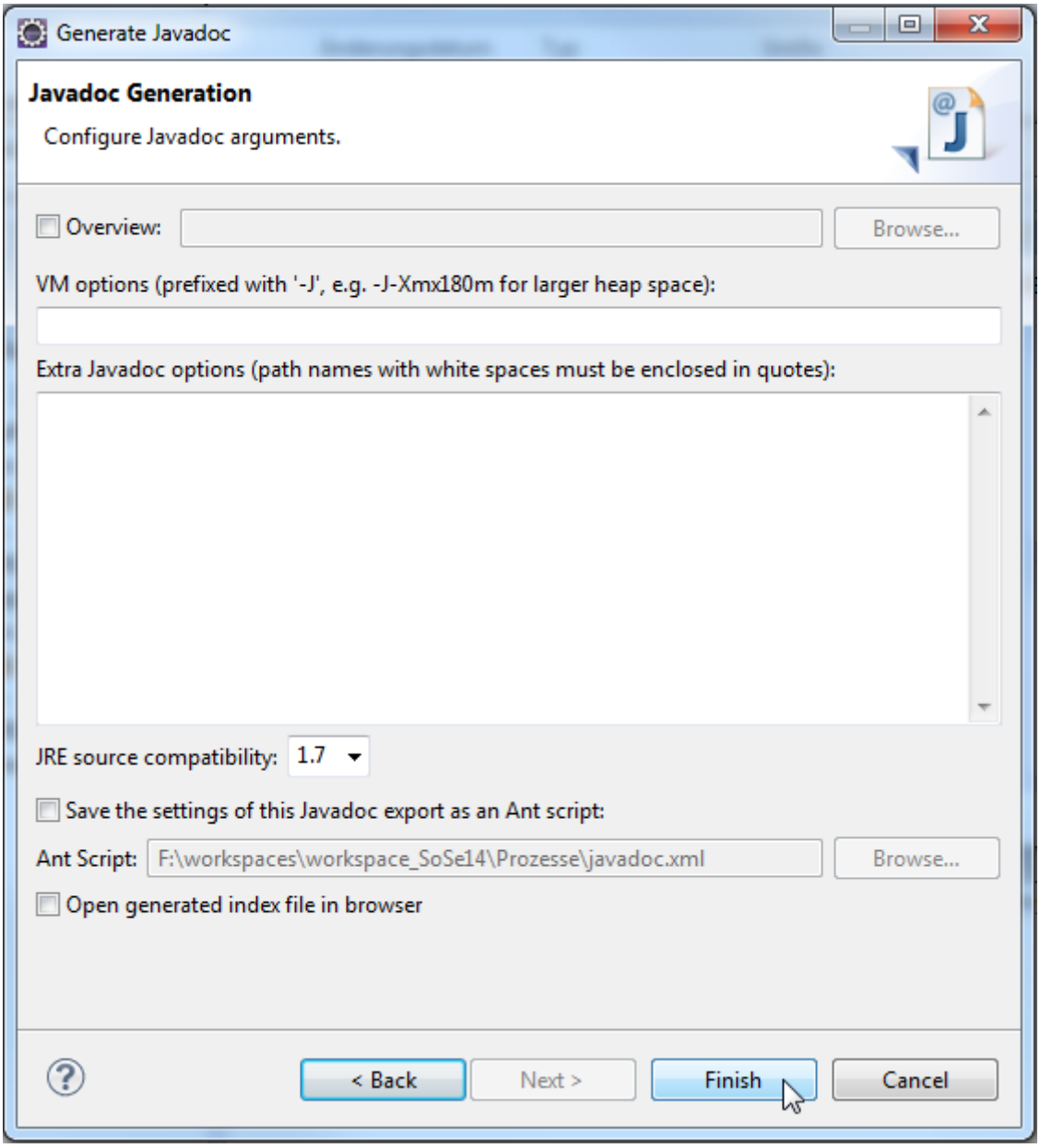
Im oberen Bereich können dann die Details ausgewählt werden, was aus welchen Projekten in die Generierung einfließen soll. Weiter unten muss angegeben werden, welche Sichtbarkeiten beim Generieren genutzt werden sollen. Während der Entwicklung wird dies meist auf private gesetzt, bei Auslieferungen auf 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.



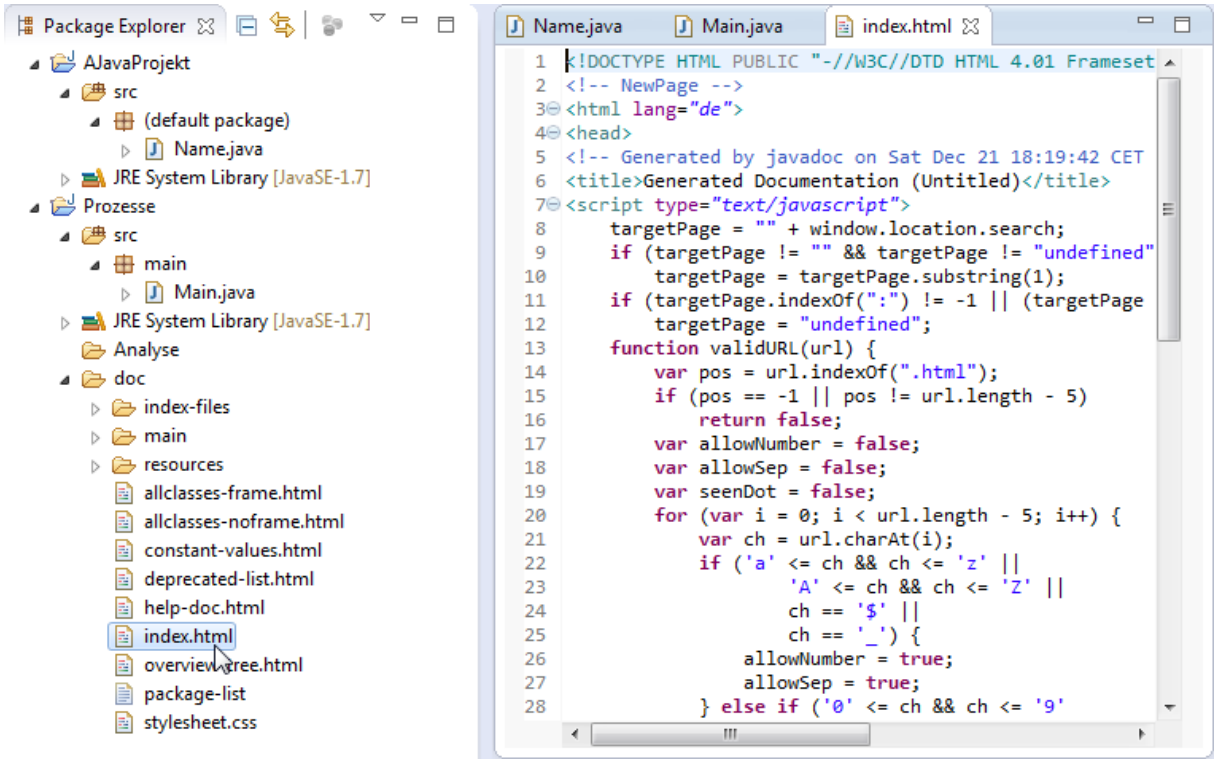
Hier kann u. a. angegeben werden, welche Tags in die Generierung einfließen. Wählt man dann „Next>“, kann man weitere Einstellungen vornehmen.



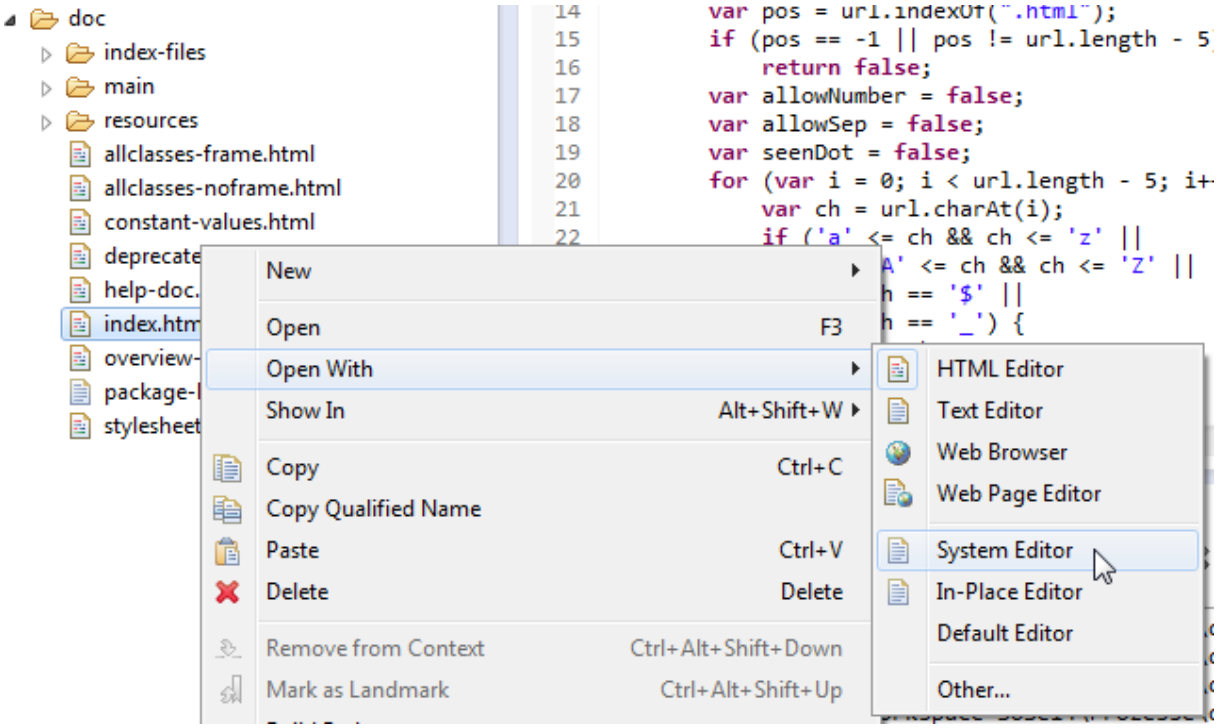
Dabei sind diese Einstellungen meist so übernehmbar und mit „Finish“ abzuschließen.



Mit den vorherigen Einstellungen entsteht im Projekt ein doc-Ordner den man öffnen kann. Doppelklickt man z.B. auf index.html, ist es abhängig vom zugeordneten Programm in Eclipse, was passiert. Ohne weitere Einstellungen wird der HTML-Editor genutzt.



Alternativ kann mit einem Rechtsklick auf der Datei eine Aktion, wie der System-Browser ausgewählt werden. Etwas irritierend, wird die letzte Aktion automatisch zu der Aktion, die bei der Bearbeitung dieser Datei genutzt wird.

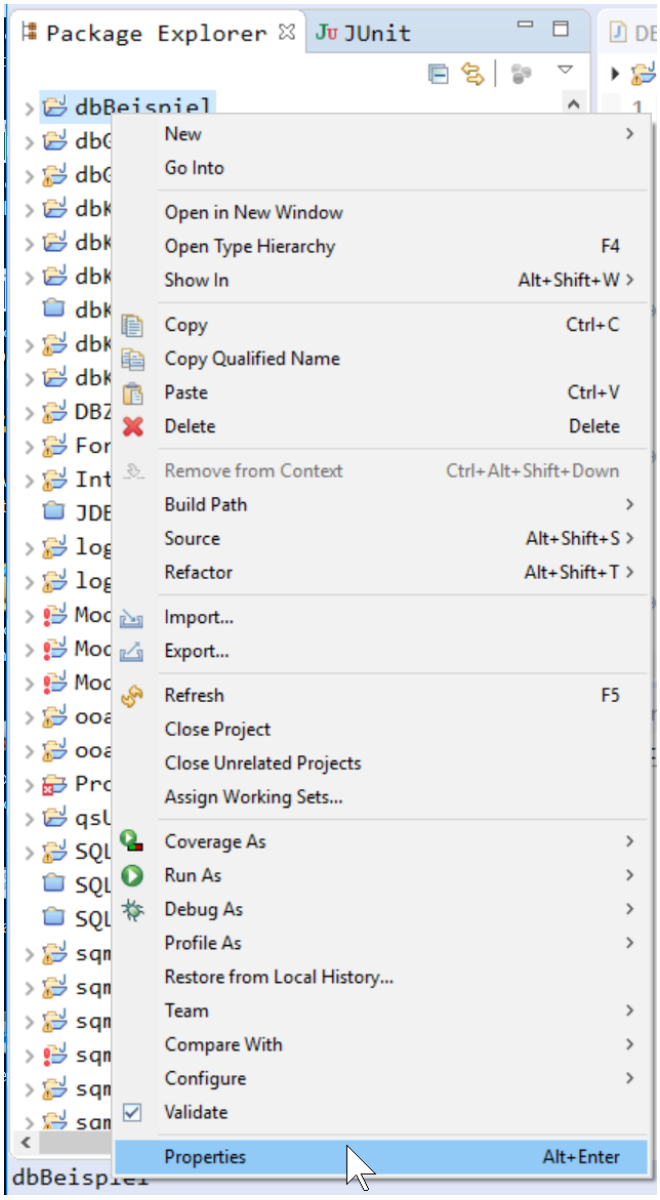


7.10 Nutzung weiterer Jar-Dateien

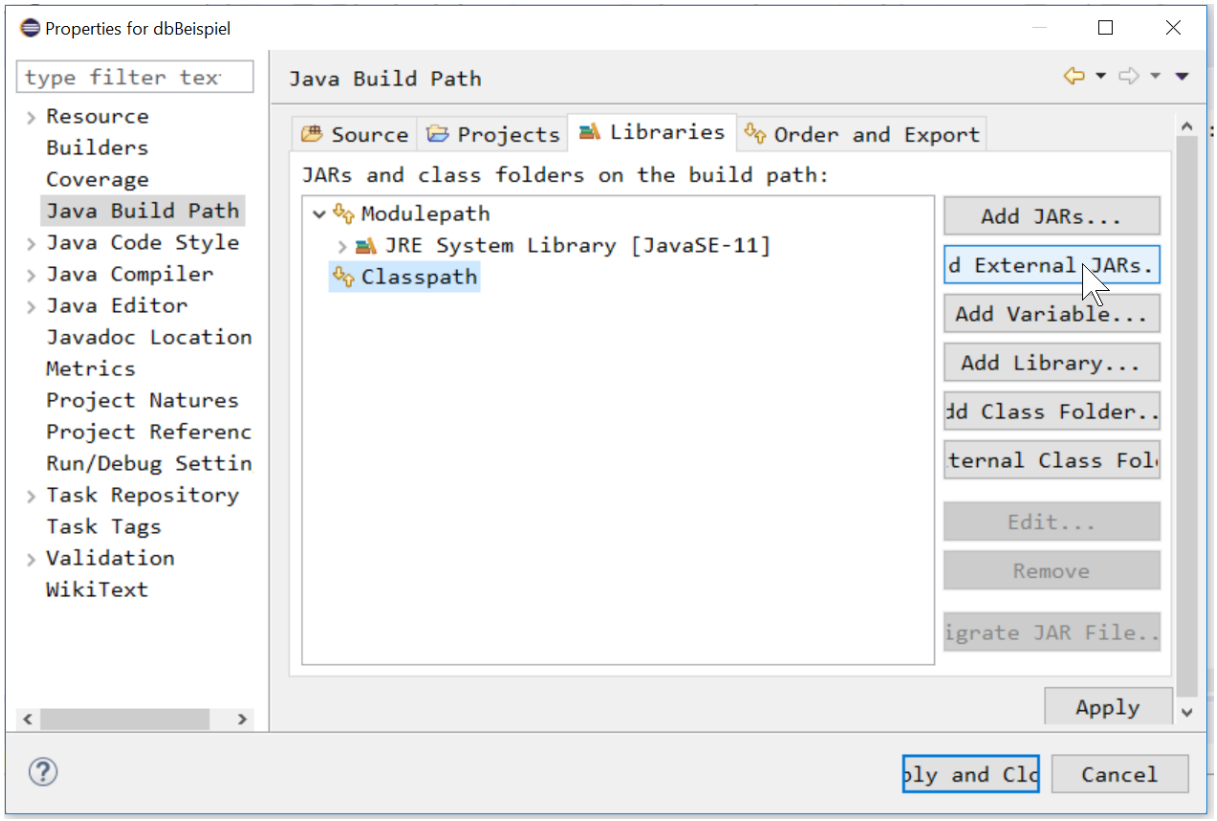
Möchte man weitere Jar-Dateien in einem Projekt nutzen (nicht bearbeiten!), müssen diese Jar-Dateien im Projekt bekannt gemacht werden. Hierzu gibt es mehrere Varianten. Die erste Variante bindet Bibliotheken ein, die sich an einem beliebigen Ort auf dem Rechner befinden können („External Jar“). Der Vorteil ist, dass viele Projekte diese Bibliothek nutzen können und sie einfach gegen eine aktuellere ausgetauscht werden kann. Die zweite Variante ist, dass die Bibliotheken in einem Unterverzeichnis des Projekts verwaltet werden, genauer wird eine Kopie jeder benutzten Bibliothek hier abgelegt. Der Vorteil ist, dass das Projekt unabhängig von anderen Projekten verschiebbar ist. Es kann z. B. als Zip-Datei verschickt und auf einem anderen Rechner ohne weitere Änderungen importiert werden.

Es sei angemerkt, dass in professionellen Projekten weitere Software zur Verwaltung von Bibliotheken eingesetzt wird, was z. B. mit Maven oder Gradle möglich ist.

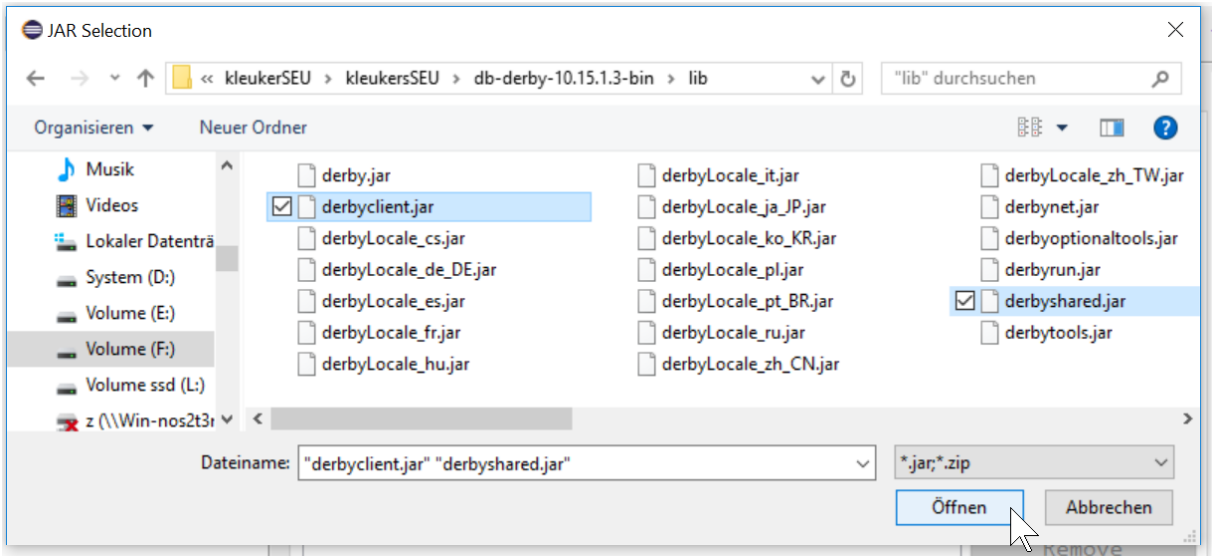
Der erste Weg startet damit, dass man einen Rechtsklick auf das Projekt macht und „Properties“ wählt.



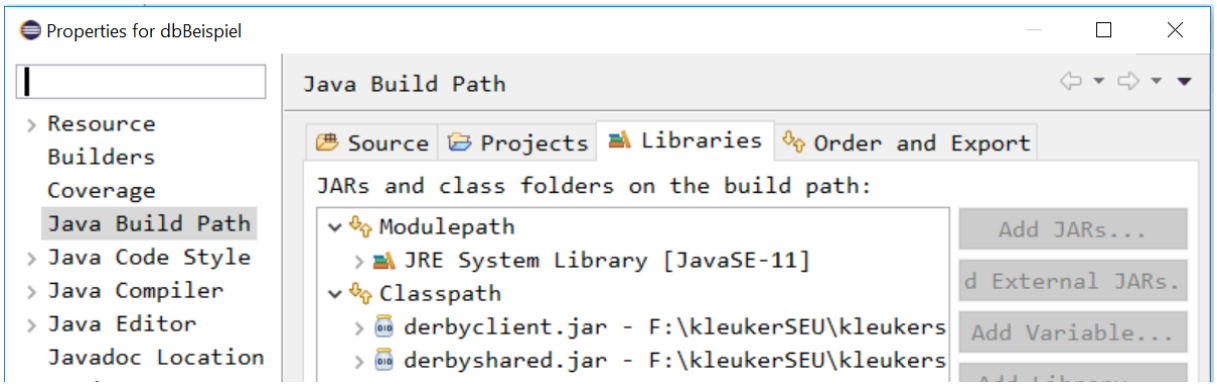
Hier kann unter „Java Build Path“ der Reiter „Libraries“ gewählt, auf „Classpath“ oder „Modulepath“ geklickt und die zu nutzende Jar-Datei unter „Add external JARs...“ ergänzt werden. Wird das Modulsystem von Eclipse im Projekt genutzt, ist immer der „Modulepath“ vorzuziehen.



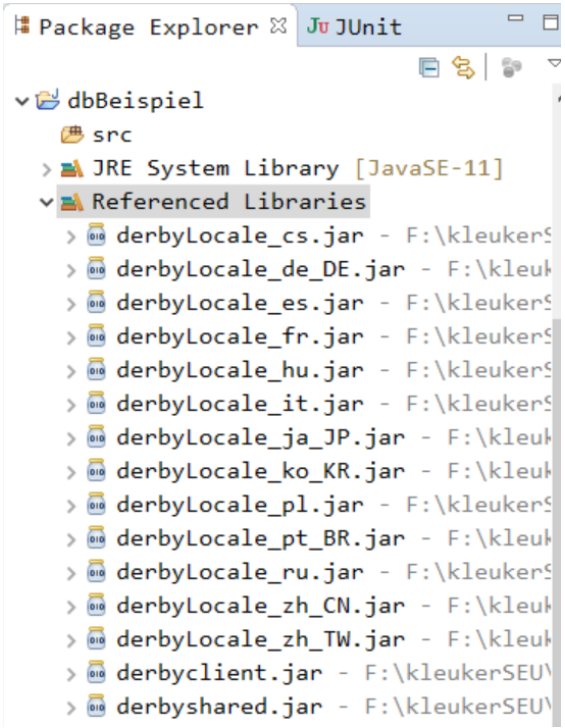
Genauer können hier jar- oder zip-Dateien eingebunden werden. Mit gedrückter Strg-Taste sind mehrere Dateien auswählbar. Die Auswahl wird mit „Öffnen“ und dann „Apply and Close“ abgeschlossen.



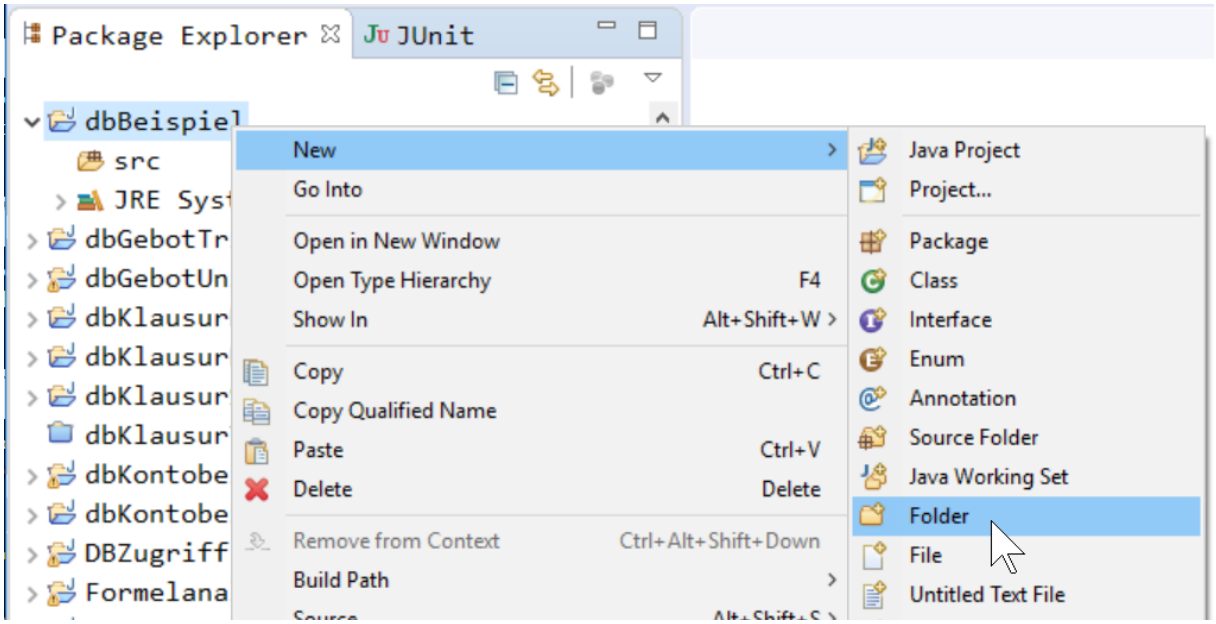
Dies wird dann auch bei den Projekteigenschaften sichtbar.



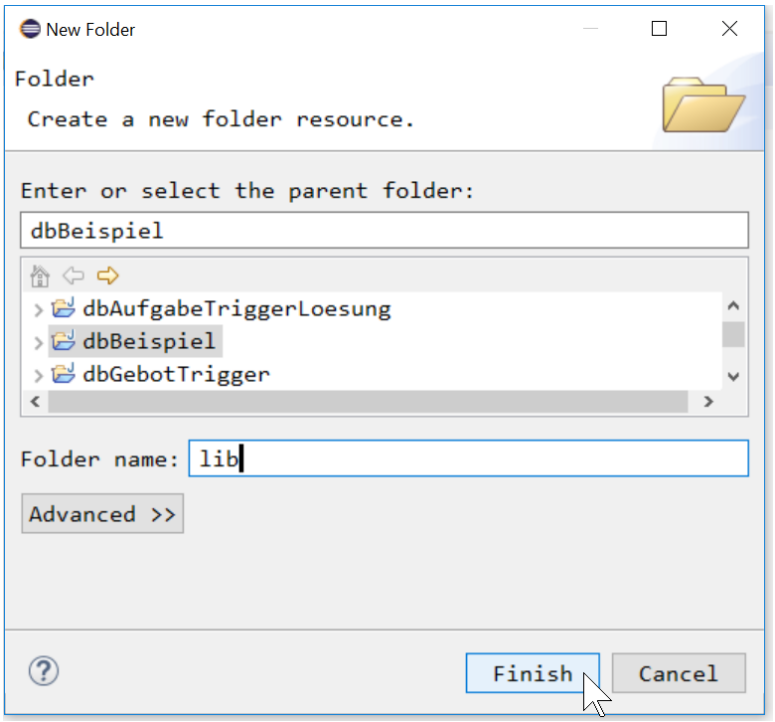
Weiterhin ist die nun nutzbare Bibliothek auch im Projektordner erkennbar. Das Beispiel zeigt, dass sich in einer Jar-Datei weitere Bibliotheken befinden können.



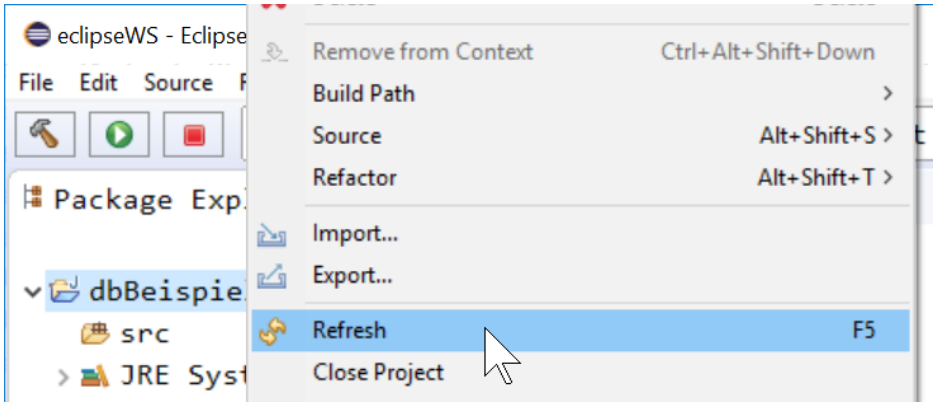
Sollen die Bibliotheken lokal im Projekt verwaltet werden, ist es sinnvoll, zunächst einen Ordner lib anzulegen. Dazu wird ein Rechtsklick auf dem Projekt gemacht und „New > Folder“ ausgewählt.



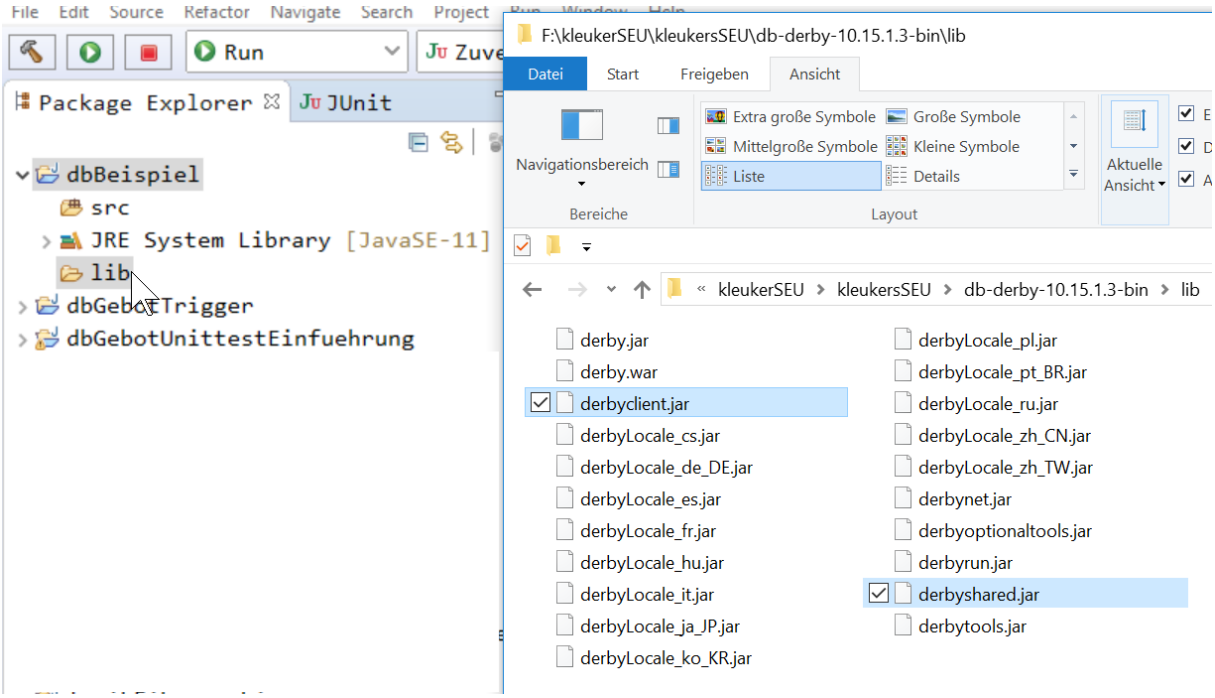
Jetzt wird der Name des Ordners eingegeben und die Erstellung mit „Finish“ abgeschlossen.



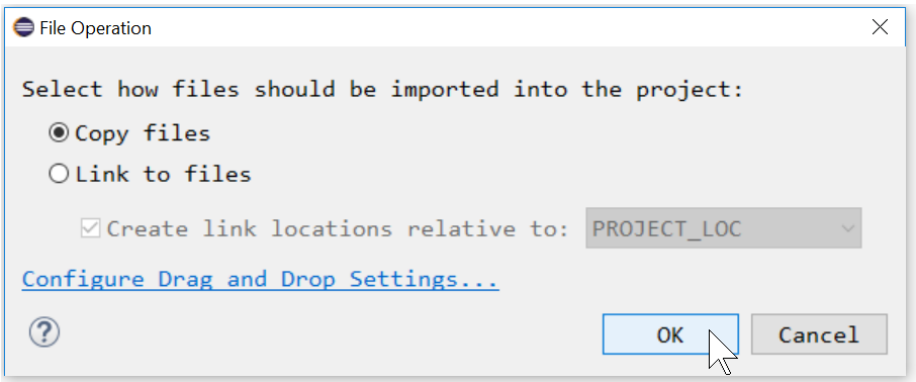
ES gibt mehrere Möglichkeiten die benutzten Bibliotheken in den entstandenen Ordner zu kopieren. Wird einfach das Betriebssystem genutzt, sind die Bibliotheken nicht direkt sichtbar. Dazu muss nach einem Rechtsklick auf dem Projekt zunächst „Refresh“ ausgewählt werden.



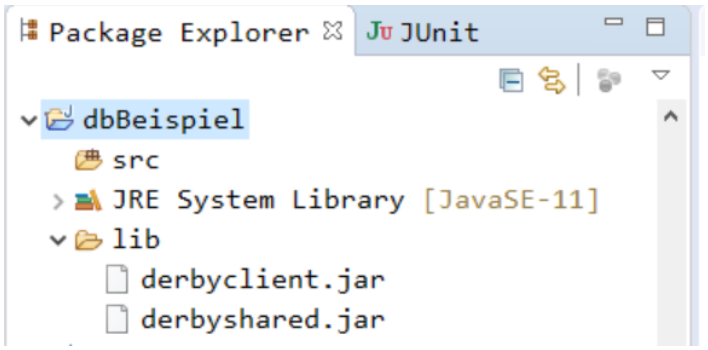
Ein anderer sehr einfacher Weg ist die von Eclipse unterstützte Möglichkeit für „Drag & Drop“ für Ordner in Eclipse.



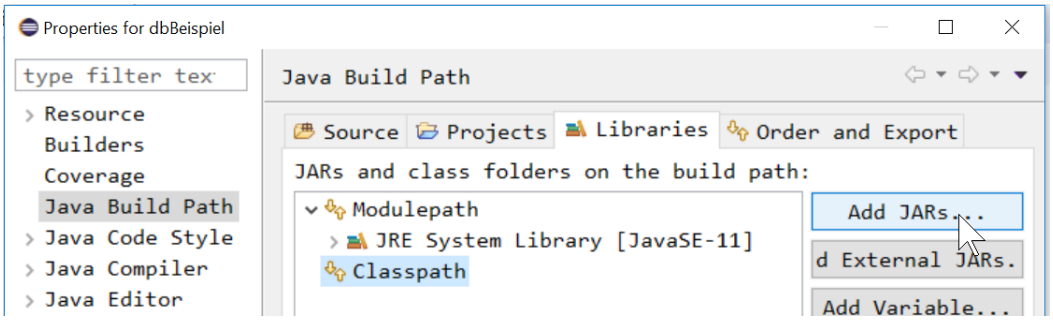
Eclipse stellt dann die Möglichkeit zum Kopieren zur Verfügung, die mit „OK“ abgeschlossen wird.



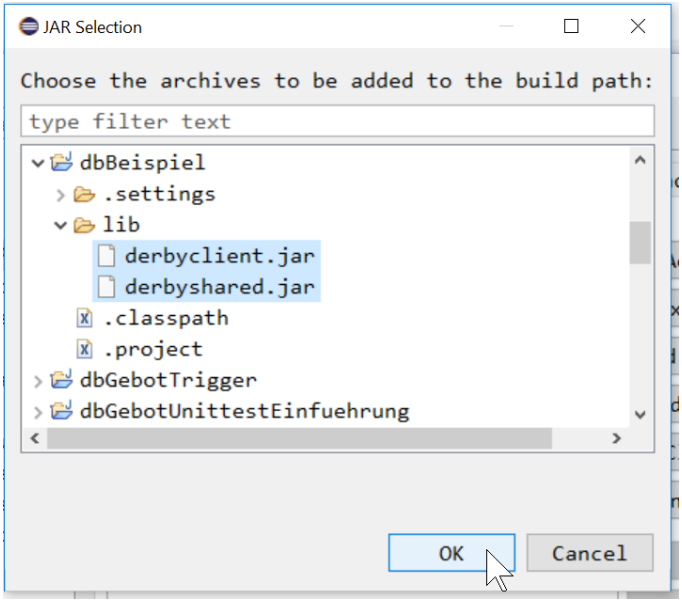
Die Dateien befinden sich jetzt im Ordner, sind aber noch nicht eingebunden.



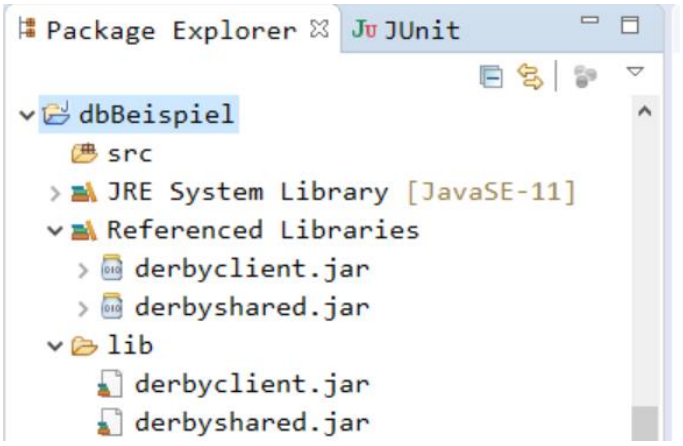
Das Einbinden geschieht ähnlich zum ersten Ansatz, der einzige Unterschied ist, dass jetzt „Add JARs…“, wenn möglich wieder beim „Modulepath“, ausgewählt wird.



Der Auswahldialog zeigt jetzt direkt den Workspace an, in dem zu den einzubindenden Dateien gesteuert wird. Der Schritt endet wieder mit „OK“ und „Apply and Close“.



Die Bibliotheken können jetzt genutzt werden. Das Icon bei den Dateien im Ordner lib verändert sich leicht, wodurch deutlich wird, dass diese Dateien genutzt werden.

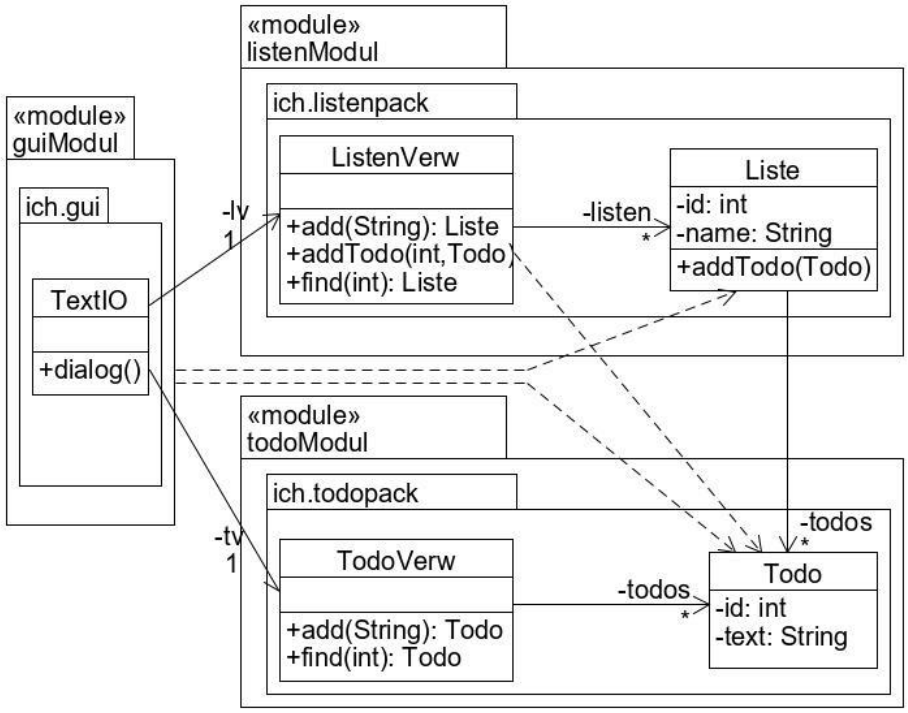


7.11 Einstieg in die Entwicklung von Modulen

Seit Java 9 unterstützt Java die Entwicklungen von Modulen mit denen große Projekte strukturierter, halt modularisierter, entwickelt werden können. Vereinfacht entspricht ein Modul einer Sammlung von Paketen für die auf Modulebene festgelegt werden kann, welches Modul auf welches andere Modul, genauer dessen Pakete, zugreifen darf. Dies löst einige mögliche Probleme, schafft aber neue Probleme bei der Weiterentwicklung älterer Software und der Verwendung von Bibliotheken, so dass selbst die Empfehlung neue Projekte mit Modulen aufzubauen zum Zeitpunkt der Erstellung dieses Textes nicht uneingeschränkt gegeben werden kann.

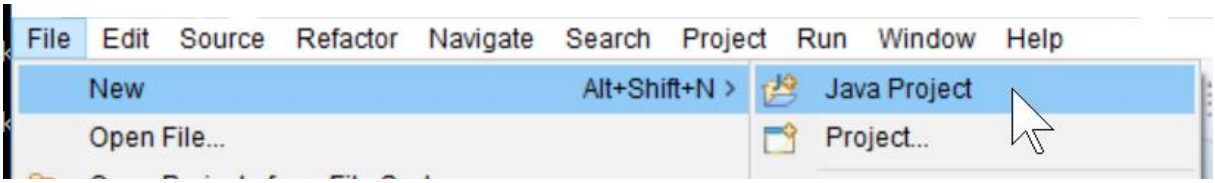
Neben der strukturierten Aufbereitung der Sichtbarkeit von Paketen unterstützen Module auch eine Art Service-Begriff basierend auf Interfaces, der im Folgeabschnitt behandelt wird. Am Ende diese Kapitels steht ein kurzer Hinweis, wie Eclipse die Umstellung eines nicht mit Modulen entwickelten Systems auf Module unterstützt.

Konkret wird in diesem Kapitel zunächst nur skizziert, wie eine Umsetzung des folgenden Klassendiagramms erfolgt, ohne auf Details einzugehen. Die Motivation dieser Software ist eine Aufgabenverwaltung, wobei einzelne Aufgaben in Todo-Objekten stehen. Mehrere Todos können in einer Liste, also Aufgabenliste verwaltet werden. Für die Klassen Todo und Liste gibt es jeweils Verwaltungsklassen, die jeweils alle zu gehörigen Objekte verwalten. Es ist so u. a. möglich, dass ein Todo-Objekt mehreren Liste-Objekten zugeordnet werden kann. Diese Möglichkeit wird durch eine Oberfläche, hier vereinfacht in einer Klasse TextIO, geschaffen. Das Klassendiagramm zeigt, dass alle Klassen in Paketen organisiert sind und dass die Pakete zu Modulen gehören, im konkreten einfachen Fall nur ein Paket pro Modul. Da die UML keine eigene Notation für Module kennt, sind diese als Paket mit Stereotyp <<module>> dargestellt. Auf sinnvolle Namensregeln wurde hier verzichtet, Modulnamen sollten wie Paketnamen aufgebaut sein.

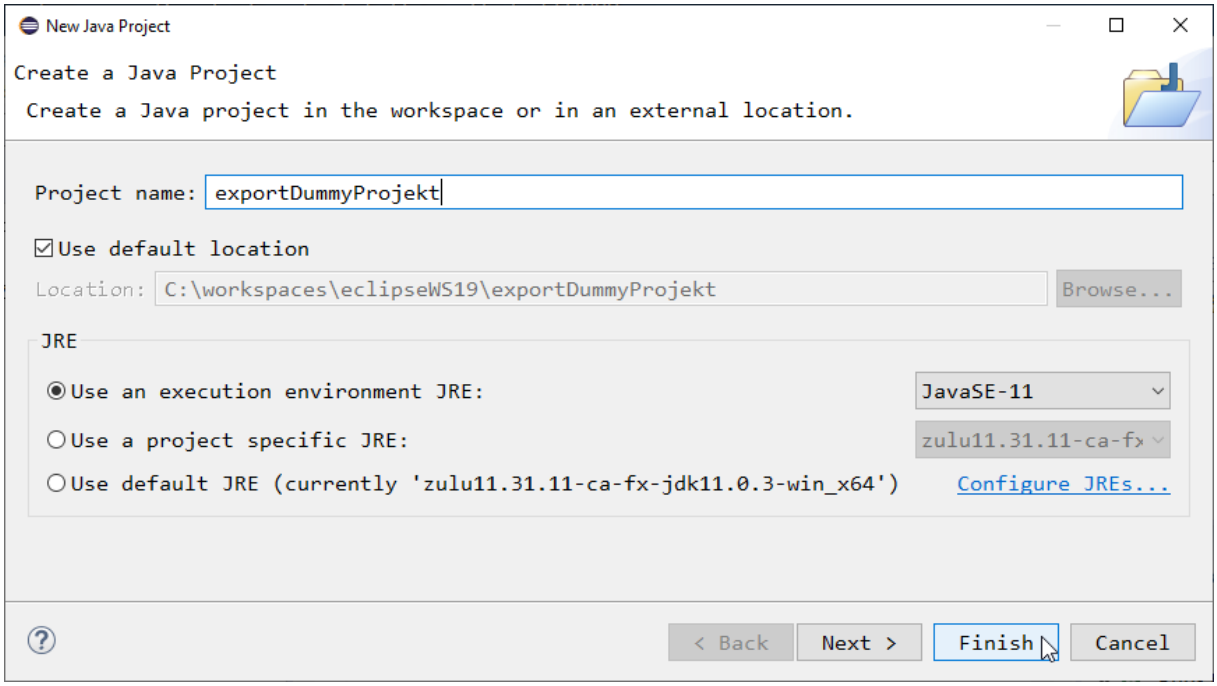


Hier wird ein möglicher Weg zur Entwicklung mit Modulen gezeigt, dessen Fokus auf einer einfachen Struktur für Anfänger liegt. Dazu gehört das Anlegen eines Hilfsprojekts, dass ausschließlich dazu dient, die aus den Modulen übersetzten Jar-Dateien zu enthalten. Diese Dateien dürfen generell fast überall, nur nicht in den Projekten selbst gespeichert werden. Dieser Trick mit dem Hilfsprojekt führt dazu, dass die Jar-Dateien sich im Eclipse-Workspace befinden und so lokal aus Projekten zugreifbar sind. Der Workspace kann damit leicht verschoben werden, was bei externen Jar-Dateien nicht der Fall sein muss.

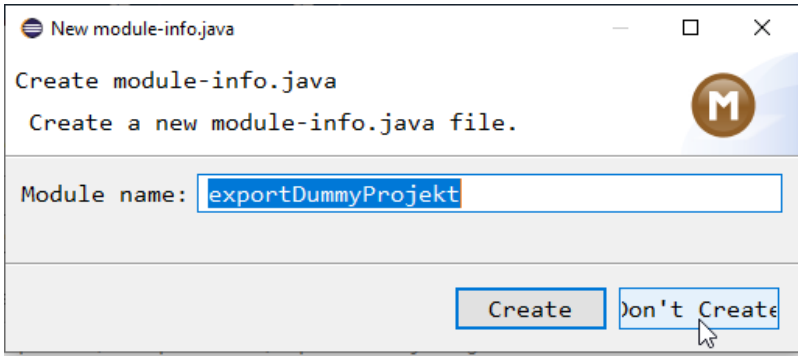
Es wird ein Rechtsklick auf „File“ gemacht und „New > Java Project“ angeklickt. Falls „Java Project“ nicht angezeigt wird, muss der Umweg über „Projct…“ gemacht werden.



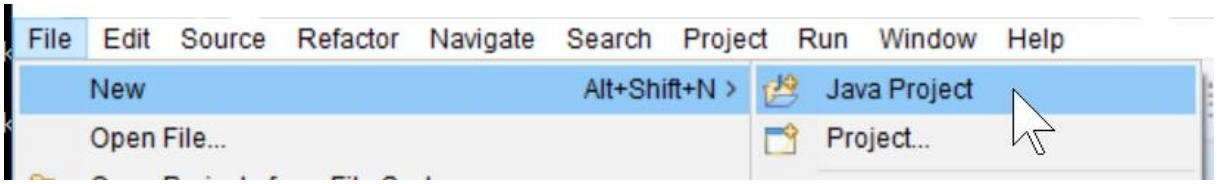
Als Name wird oben „exportDummyProjekt“ eingetragen und unten „Finish“ geklickt.



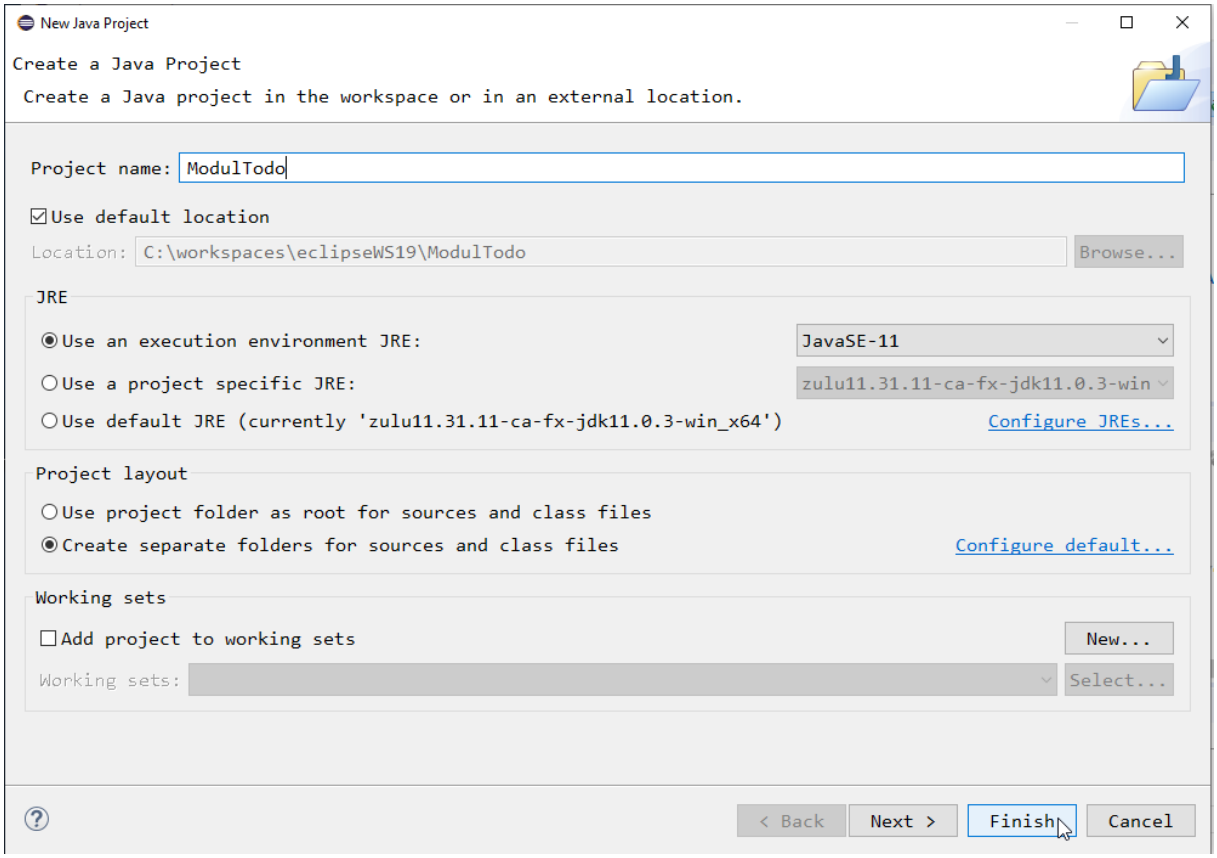
Nur für dieses Projekt wird kein Modul angelegt und einfach rechts-unten „Don't Create" geklickt.



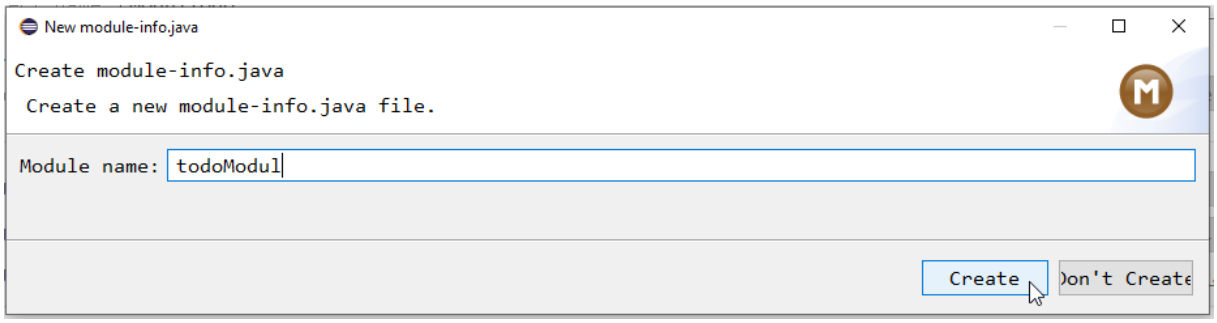
Für jedes Modul wird ein eigenes Projekt angelegt, deren Namen frei wählbar sind. Für das unterste Modul sieht der Weg wie folgt aus. Es wird ein Rechtsklick auf „File" gemacht und „New > Java Project" angeklickt.



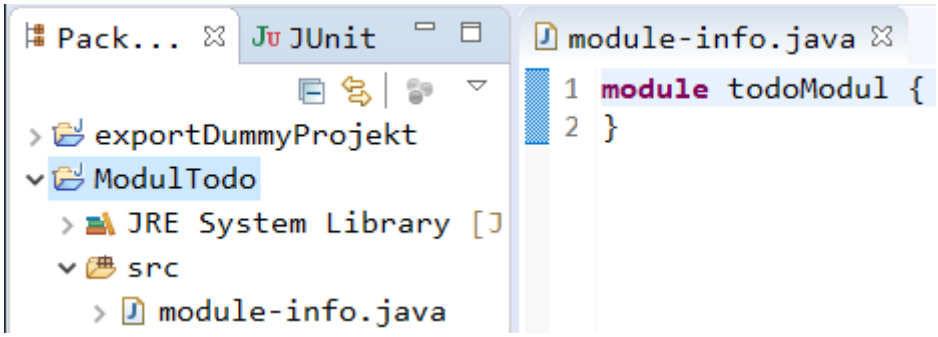
Als Name wird oben „ModulTodo" eingetragen und unten „Finish" geklickt.



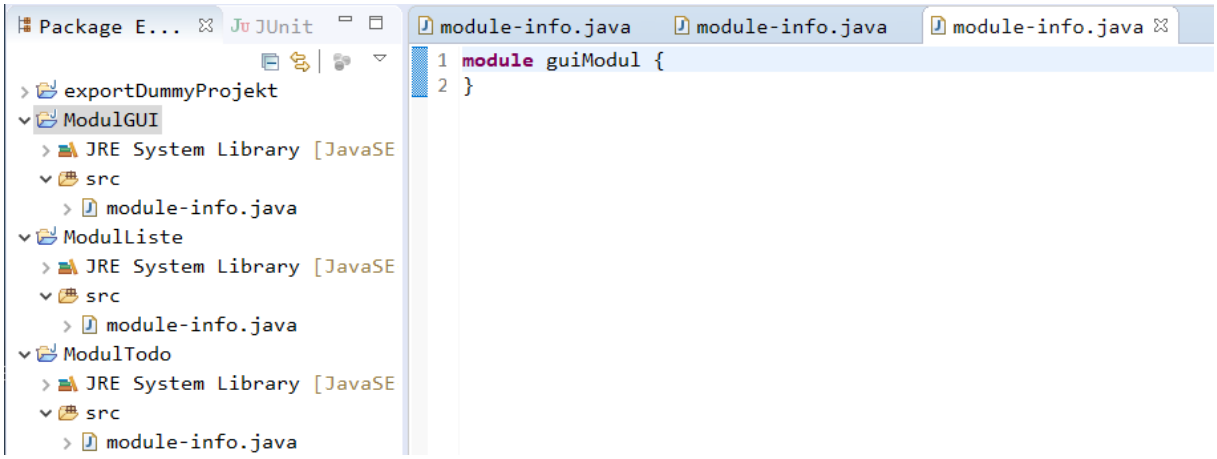
Jetzt wird ein Modulname aus dem Klassendiagramm eingetragen und „Create“ geklickt.



Es entsteht ein neues Projekt und die zentrale Modul-Konfigurationsdatei module-info.java.



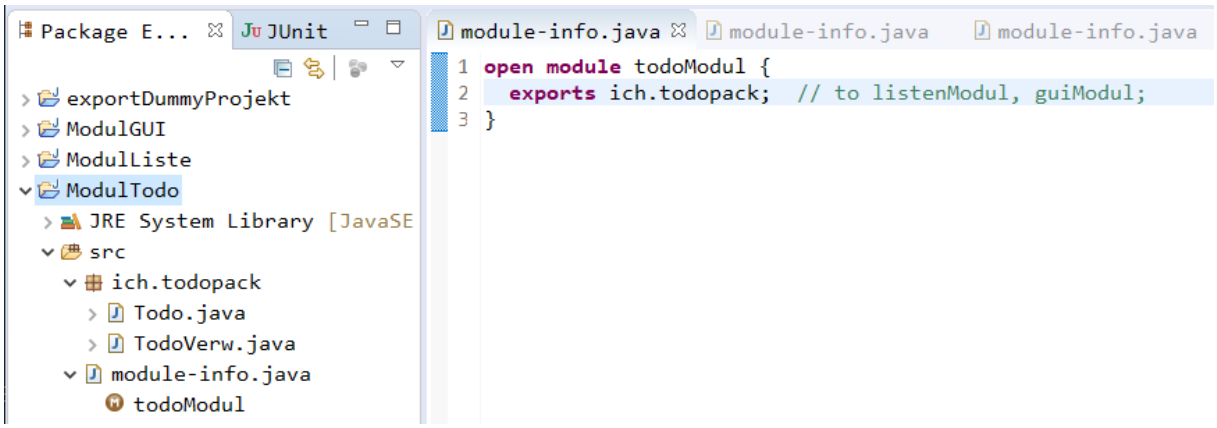
Es werden die Projekte ModulListe und ModulGUI mit den Modulen listenModul und guiModul in gleicher Form angelegt.



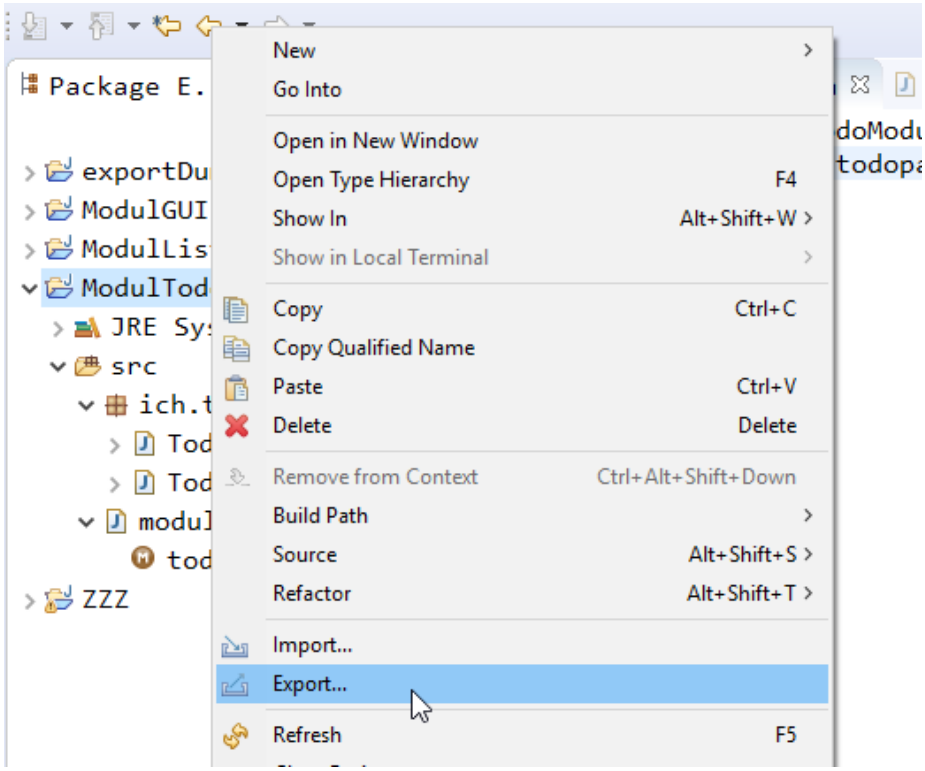
Jetzt findet die übliche Programmierung statt. Danach muss die modul-info erweitert werden. Beim todoModul soll das programmierte Paket in allen anderen Modulen nutzbar sein, dazu wird der Eintrag

```
exports ich.todopack;
```

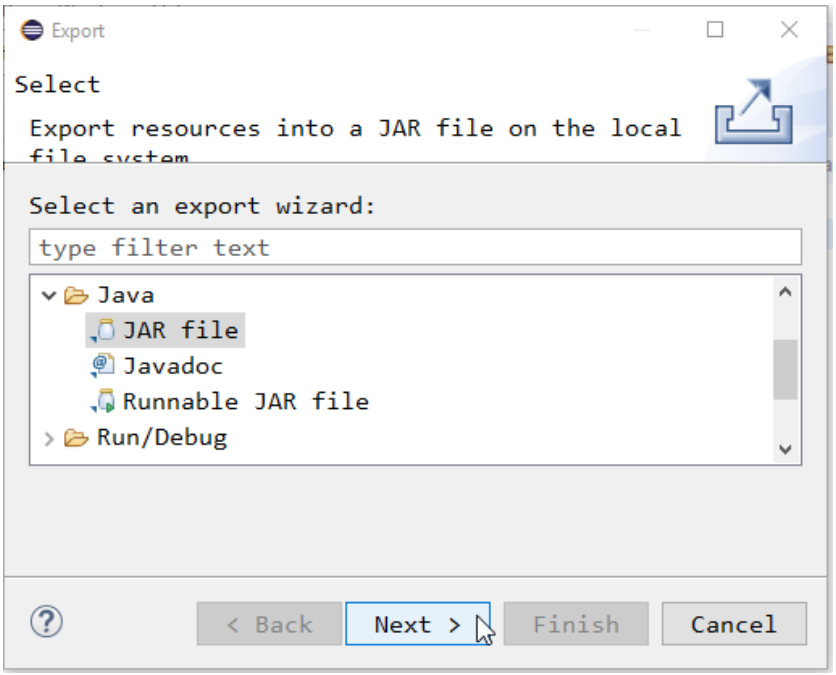
ergänzt. Weiterhin wurde am Anfang das Schlüsselwort „open“ hinzugefügt, damit Klassen dieses Modul u. a. von anderen Bibliotheken und Werkzeugen aus nutzen können. Die Nutzung erfolgt oft per Reflection die durch das Schlüsselwort „open“ ermöglicht wird und zumindest einige der am Anfang angedeuteten Probleme verhindert. Sollen nicht alle Module zugreifen können, kann obige Zeile durch „to“ und eine Liste von kommaseparierten Modulnamen ergänzt werden. Ausschließlich Module dieser Liste können dann dieses Modul nutzen.



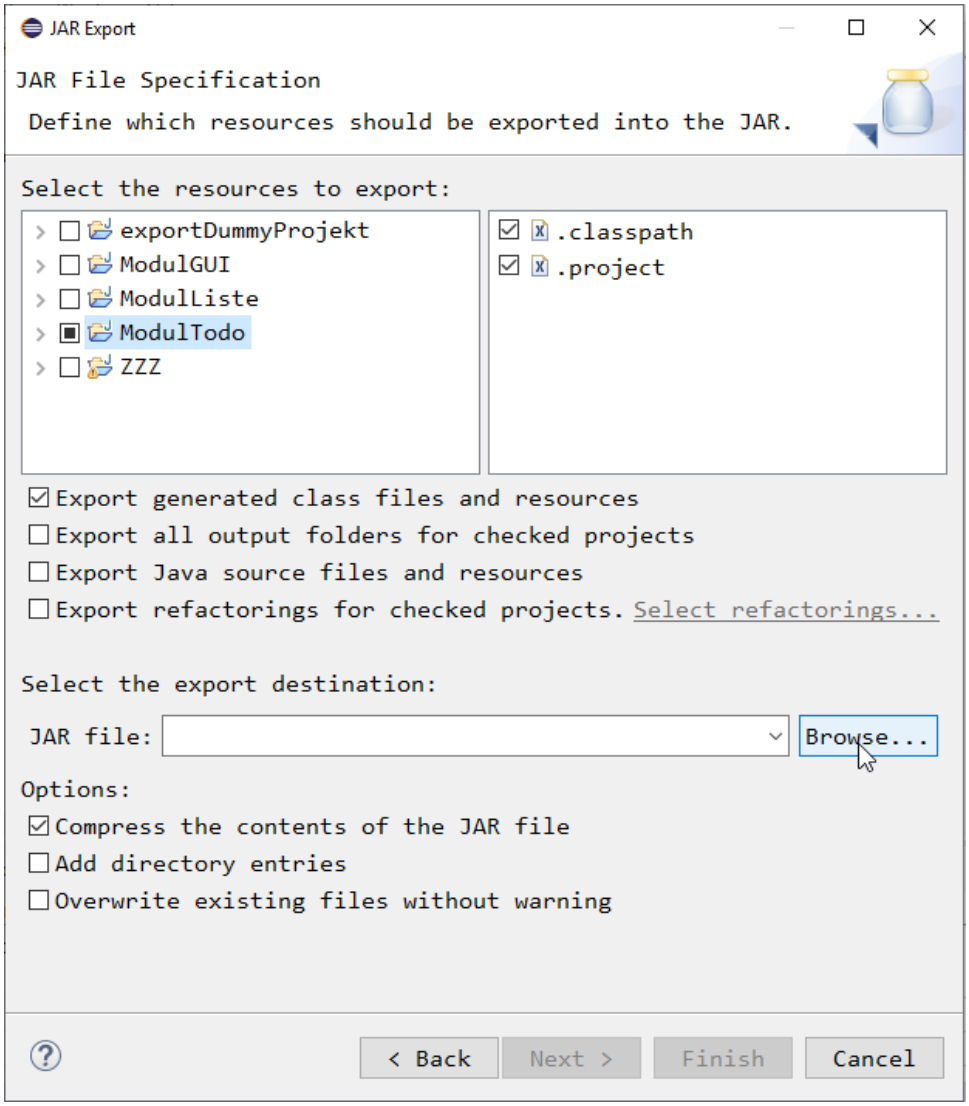
Im nächsten Schritt wird aus dem Modul eine Jar-Datei erzeugt, die dann von anderen Modulen genutzt werden kann. Dazu wird ein Rechtsklick auf dem Projekt gemacht und „Export…“ gewählt.



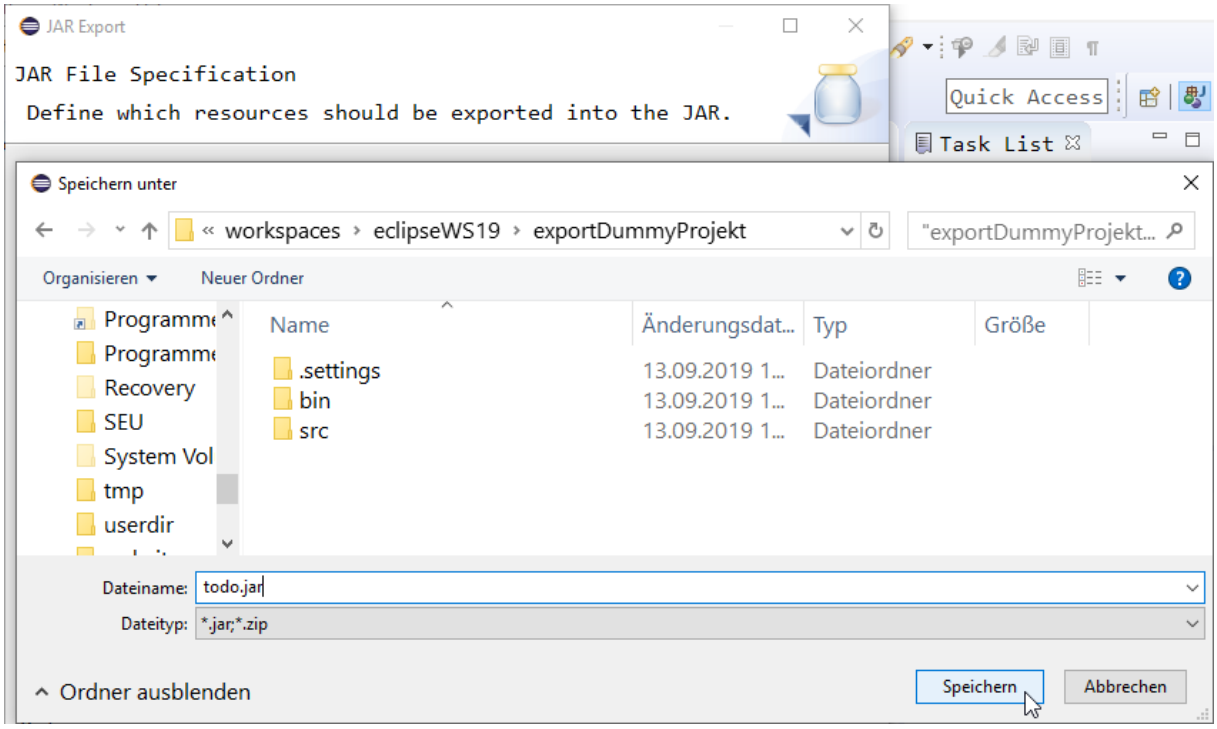
Es wird „Java“ aufgeklappt, „JAR File“ ausgewählt und „Next >“ geklickt.



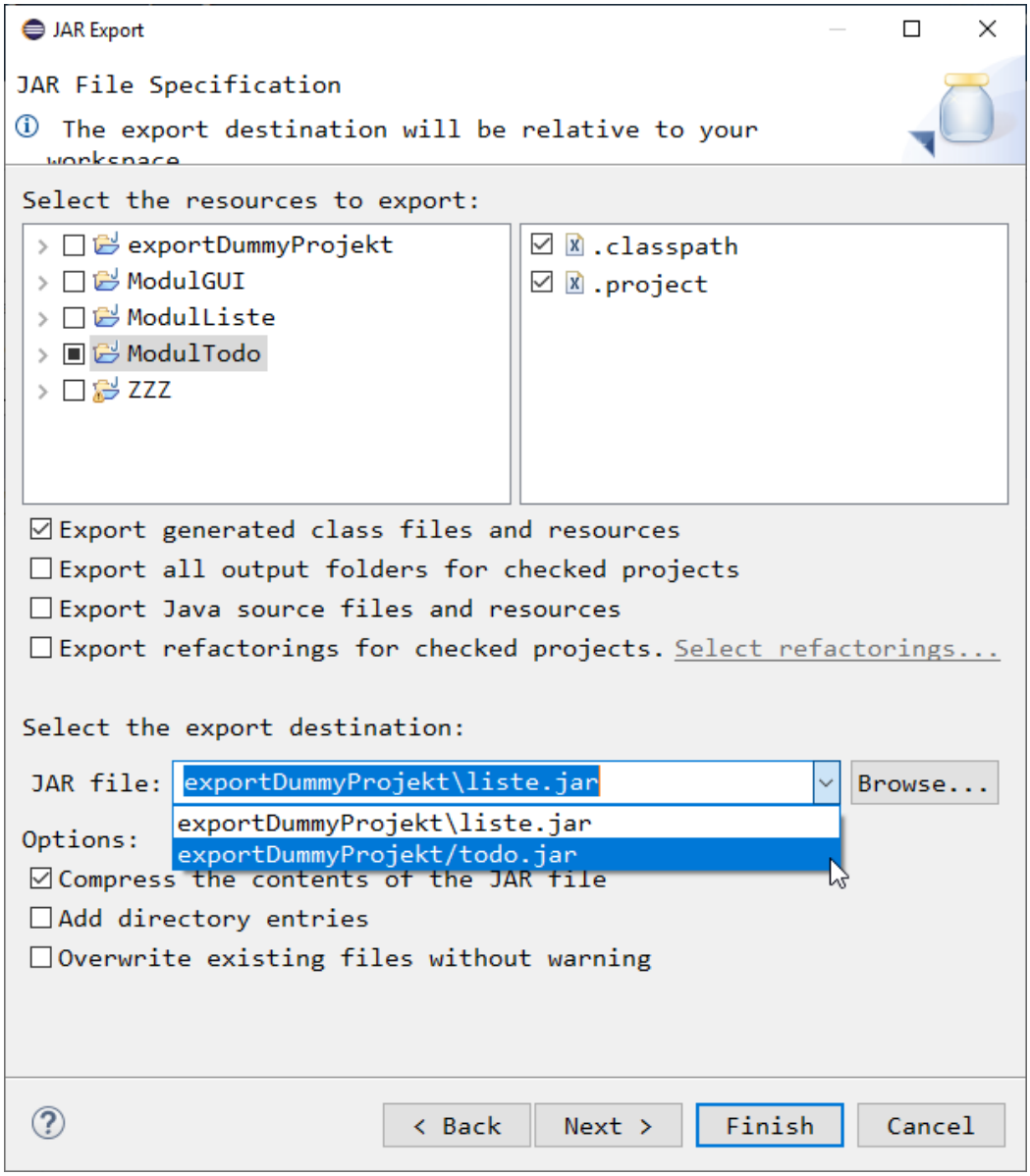
Die vorgeschlagenen Einstellungen sind in Ordnung, nur der Dateiname unter „JAR file:“ ist einzutragen. Dazu wird auf „Browse…“ geklickt.



Es wird zum anfänglich eingerichteten Hilfsprojekt manövriert und unten ein Filenamen vergeben. Die Aktion wird mit „Speichern“ abgeschlossen.



Werden Änderungen am Code vorgenommen, muss dieser Export immer wiederholt werden. Leider merkt sich Eclipse nicht den Pfad zur Jar-Datei, es ist immer der zuletzt genutzte Pfad eingetragen. Dies ist bei der parallelen Arbeit an mehreren Modulen sinnlos. Zum Glück befindet sich neben dem Feld ein Pfeil zum Aufklappen einer Auswahl-Box, die die zuletzt genutzten Pfade enthält.



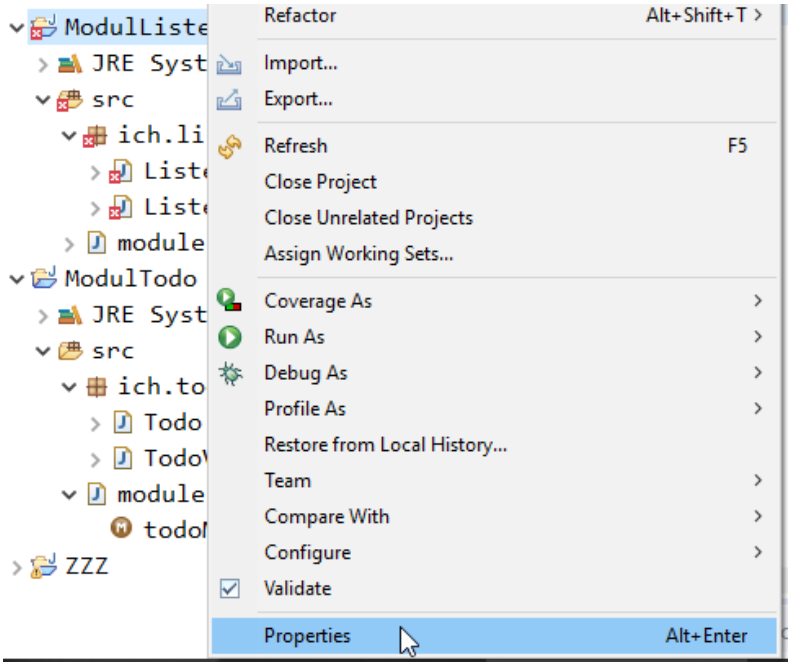
Nun wird ModulListe programmiert. Dieses Projekt nutzt das vorher angegebene Modul, was in der module-info einzutragen ist. Der einfachste korrekte Eintrag wäre

```
requires todoModul;
```

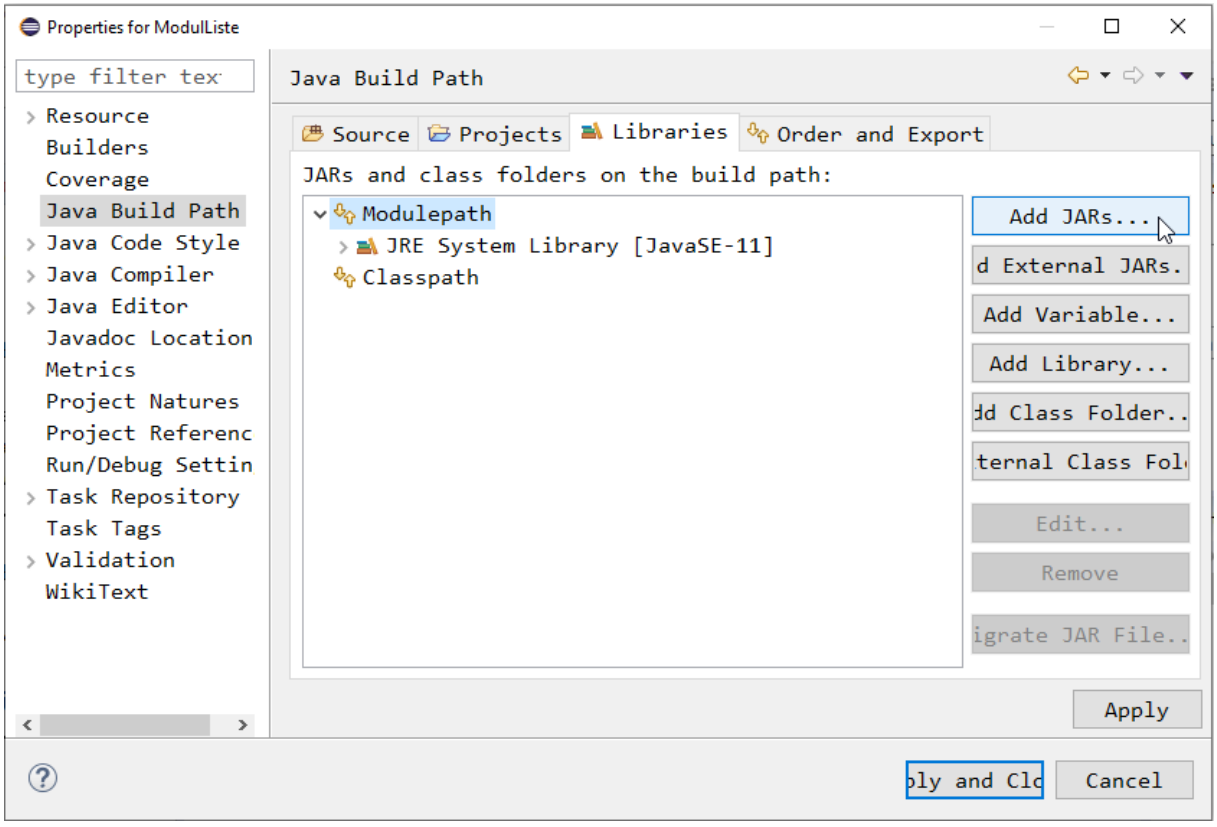
Da Nutzer dieses Moduls auch das Todo-Modul benötigen, könnte dies in einer weiteren Zeile als „exports“ ergänzt werden. Flexibler ist das Schlüsselwort „transitive“, was Nutzern dieses Moduls den Zugriff auf von diesem Modul benutzte Module ermöglicht. Der zugehörige Befehl sieht wie folgt aus.

```
requires transitive todoModul;
```

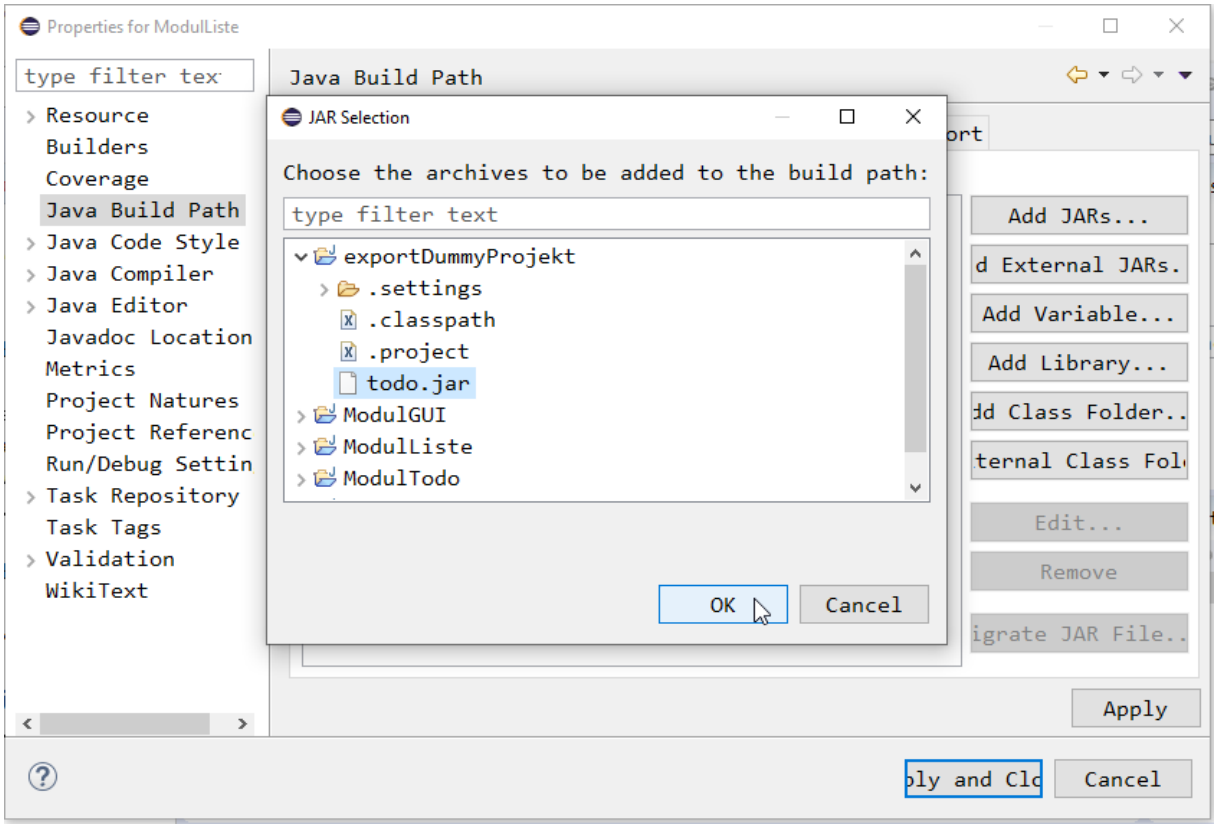
Weiterhin muss das benutzte Modul eingebunden werden. Dazu wird ein Rechtsklick auf dem Projekt gemacht und unten „Properties“ gewählt.



Es wird rechts auf „Java Build Path“, dann der Reiter „Libraries“, dann auf „Modulepath“ und dann rechts auf „Add JARs…“ geklickt.

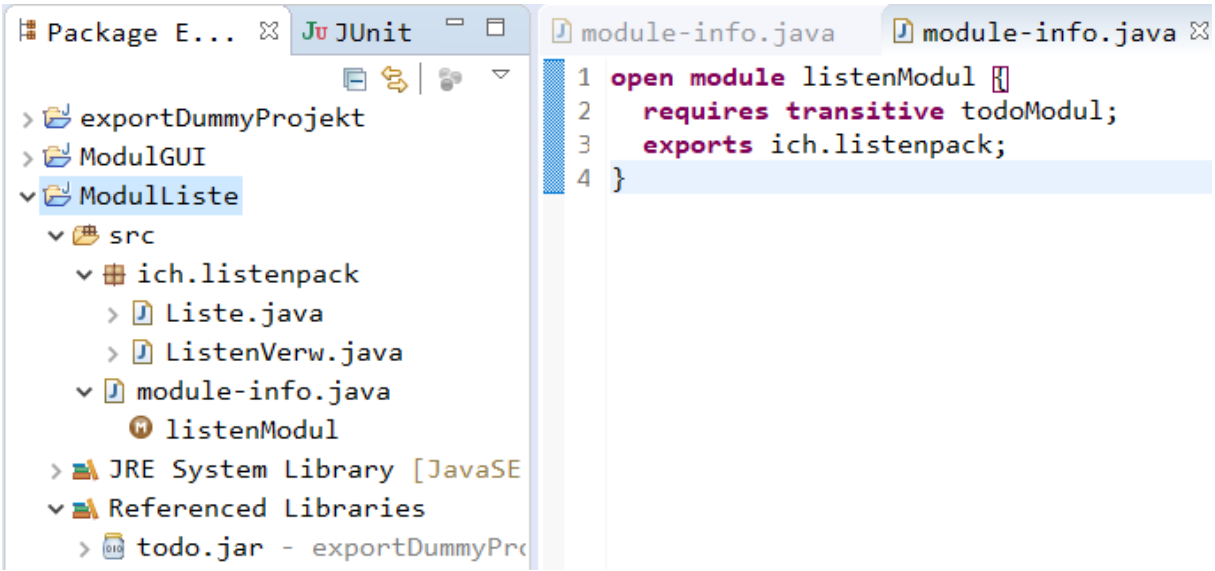


Jetzt wird zur gerade erstellten Jar-Datei gesteuert und „Ok“ geklickt. Zum Abschluss wird „Apply and Close“ geklickt.



Die Datei module-info.jar für dieses Modul sieht wie folgt aus.

```
open module listenModul {
  requires transitive todoModul;
  exports ich.listenpack;
}
```

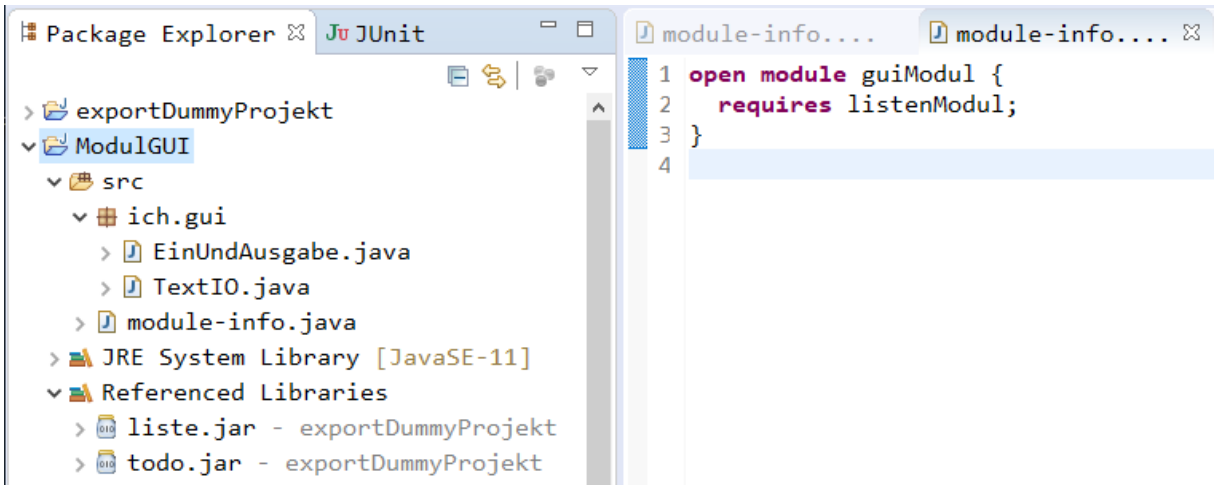


Die Jar-Datei liste.jar wird analog zum vorherigen Beispiel erstellt.

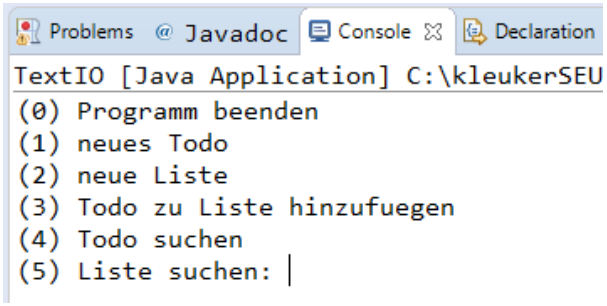
Die module-info.java für ModulGUI sieht wie folgt aus.

```
open module guiModul {
  requires listenModul;
}
```

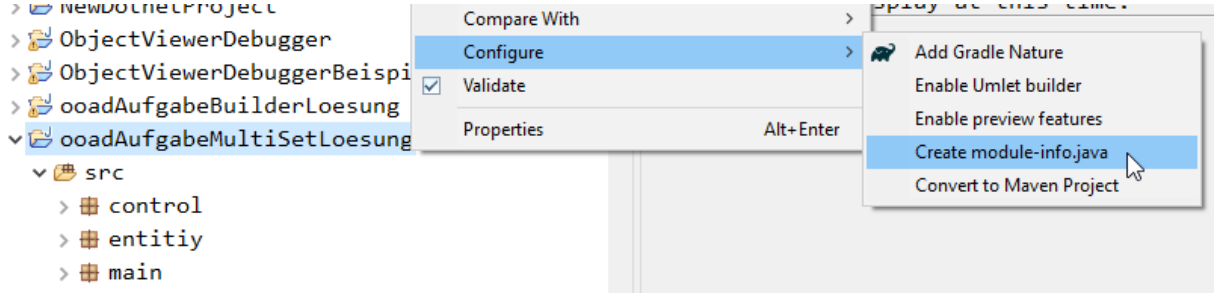
Beim Einbinden der Jar-Dateien in den Module-Path ist zu beachten, dass beide Jar-Dateien einzubinden sind. Da sonst kein Zugriff auf die Klasse Todo möglich wäre.



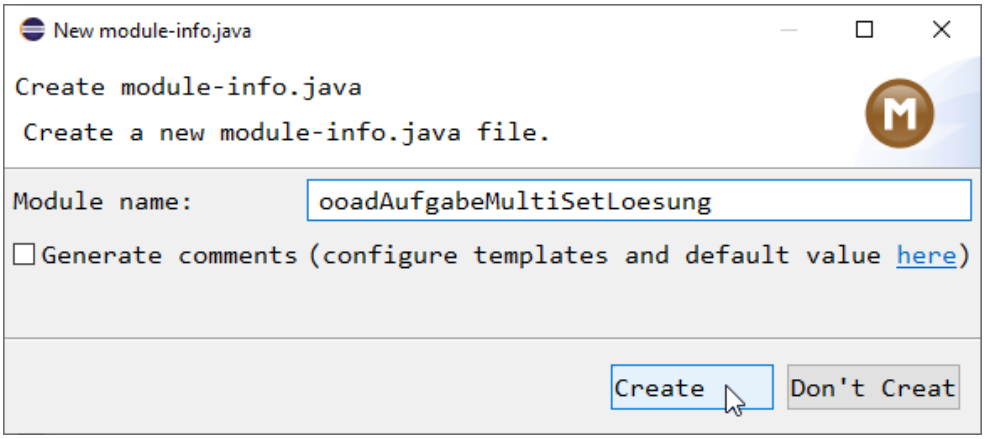
Danach kann das modulare Programm normal gestartet werden.



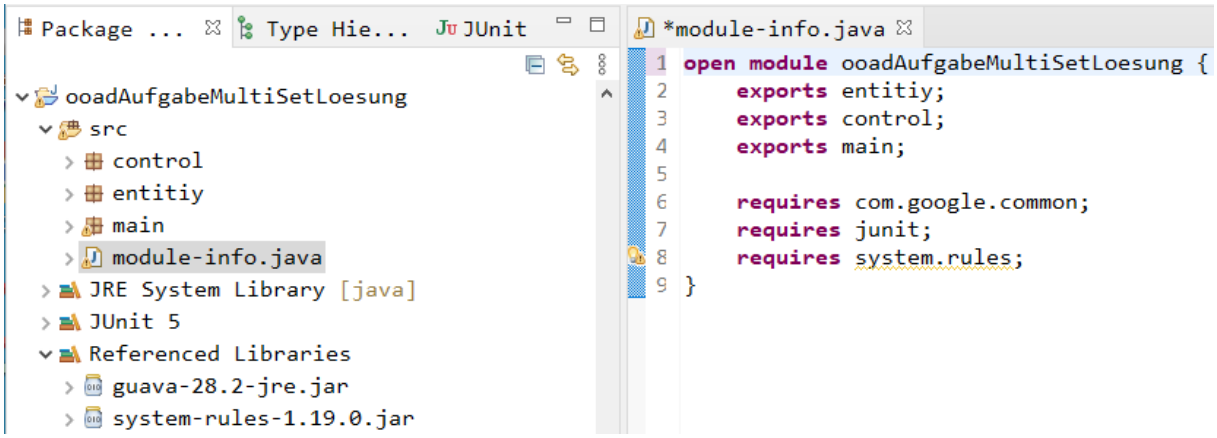
Sollte ein Projekt zunächst ohne Module entwickelt worden sein, unterstützt Eclipse zumindest etwas bei der Umstellung. Dazu wird ein Rechtsklick auf dem umzustellenden Projekt gemacht und unten „Configure > Create module-info.java“ ausgewählt.



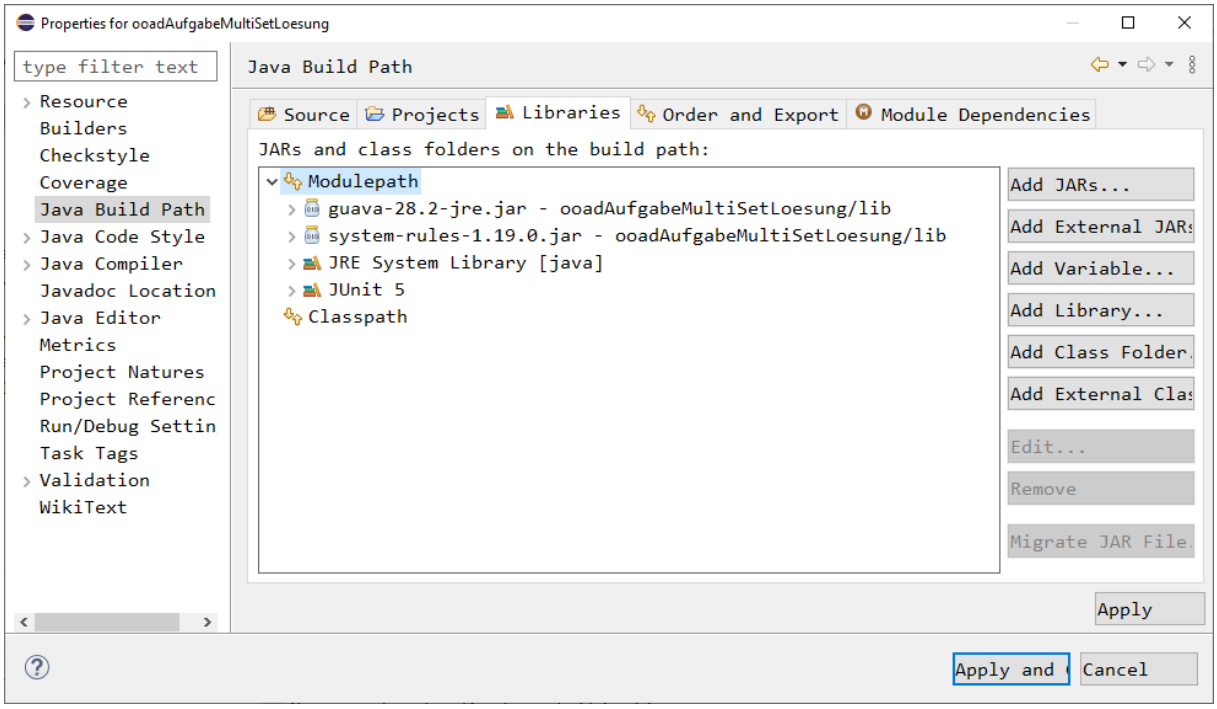
Es muss ein Name für die Modulbeschreibung als „Module name“ angegeben werden. Die Namensregeln sind identisch mit den Regeln für Paketnamen, werden aber in der folgenden Abbildung ignoriert. Die Erstellung wird mit „Create“ abgeschlossen.



Es wird automatisch die module-info.java angelegt, bei der es sinnvoll ist am Anfang ein „open“ zu ergänzen. Sind nicht alle genutzten Bibliotheken auf Module umgestellt, kann es passieren, dass die Umstellung nicht möglich ist.- Mit etwas Glück, wie im folgenden Fall, ist die Bibliothek trotzdem nutzbar und erhält automatisch einem aus dem Namen der jar-Datei genierten Namen, was in der Default-Einstellung mit einer Warning markiert ist.



Die Umstellung ist damit noch nicht abgeschlossen. Es gibt zur Nutzung von Bibliotheken zwei verschiedene Möglichkeiten, wie diese Bibliotheken gefunden werden können. Die alte Möglichkeit ist der Class-Path, die neuere ist der Module-Path. Generell ist es anzustreben, dass alle Bibliotheken im Module-Path stehen, da nur diese untereinander auf die jeweiligen Inhalte zugreifen können. Klassen aus Bibliotheken im Class-Path können nur Klassen nutzen, die ebenfalls zu Bibliotheken im Class-Path gehören. Eine Änderung der Einstellung erfolgt über den bereits bekannten Reiter, der nach einmem Rechtsklick auf dem Projekt und der Auswahl „Properties“ ganz unten unter dem rechten Eintrag „Java Build Path“ und dem Reiter „Libraries“ steht. Eclipse versucht alle Bibliotheken in den „Modulepath“ einzutragen, was im folgenden Fall gelingt. Bibliotheken können auch mit Drag-And-Drop zwischen den beiden Path-Varianten hin und her geschoben werden. Durch einen Klick auf „Modulepath“ können über die Funktionalität auf der rechten Seite u. a. weitere Bibliotheken in den Module-Path eingebunden werden.



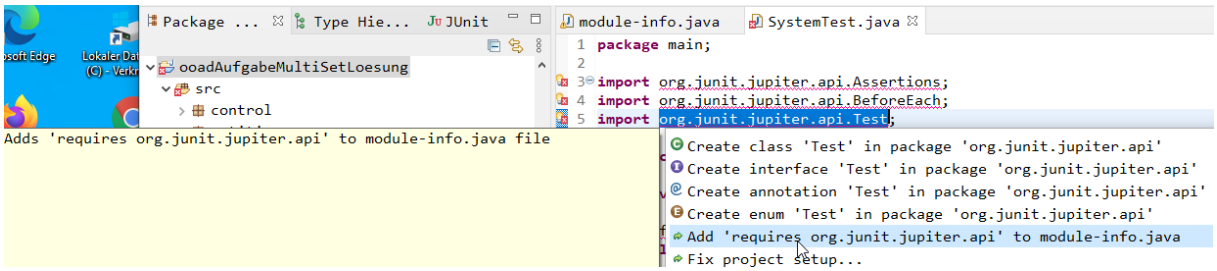
Bei der Programmierung mit Modulen ist immer zu beachten, dass genutzte Klassen nicht nur, wie im folgenden Beispiel, in den import-Zeilen erwähnt werden, sondern dass ein Eintrag in module-info.java ergänzt werden muss.

module-info.java | SystemTest.java

```
1  package main;
2
3  import org.junit.jupiter.api.Assertions;
4  import org.junit.jupiter.api.BeforeEach;
5  import org.junit.jupiter.api.Test;
6
7  public class SystemTest {
8
9      private Zugriffsdialog dialog;
10
11     @BeforeEach
12     public void setUp(){
13         this.dialog = new Zugriffsdialog();
14     }
15
16
17     @Test
18     public void testBasis(){
19         Assertions.assertEquals(42, 6*9, " sollte so sein");
20     }
21 }
```

The type org.junit.jupiter.api.Test is not accessible

In diesem Fall kann die automatische Fehlerkorrektur von Eclipse meist helfen. Wird die Meldung angeklickt, werden Lösungsvorschläge generiert und die hier vorgeschlagene Ergänzung ausgewählt.

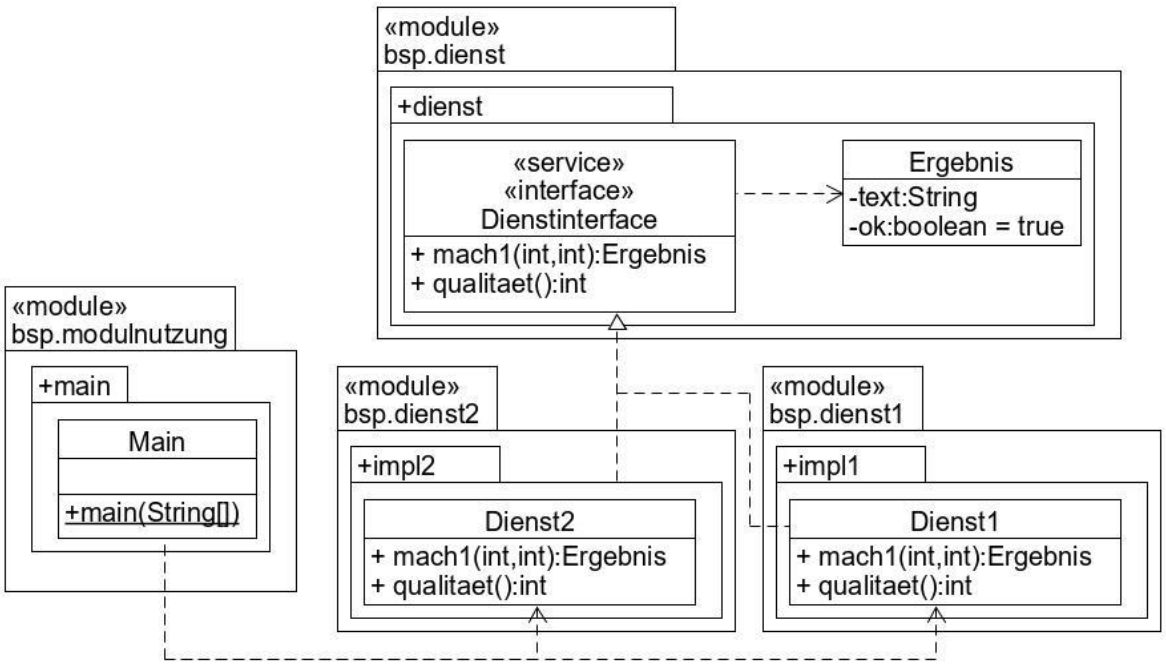


Eclipse ergänzt dann automatisch den folgenden Eintrag.

```
9    requires org.junit.jupiter.api;
```

7.12 Einstieg in die Entwicklung von Services mit Modulen

Neben der Aufteilung von Paketen in Modulen ermöglicht der Module-Path auch die Spezifikation und Realisierung sogenannter Services, die ursprünglich mit Java 6 eingeführt worden. Die Besonderheit ist, dass zur Laufzeit geprüft werden kann, welche Service-Implementierungen vorliegen. Dieser Abschnitt basiert auf „7.11 Einstieg in die Entwicklung von Modulen“, da diese Ideen aufgegriffen werden. Wieder wird hier nur ein Beispiel skizziert, dass die Regeln für den Zusammenbau zeigt, wozu das im folgenden Klassendiagramm beschriebene Beispiel genutzt wird.

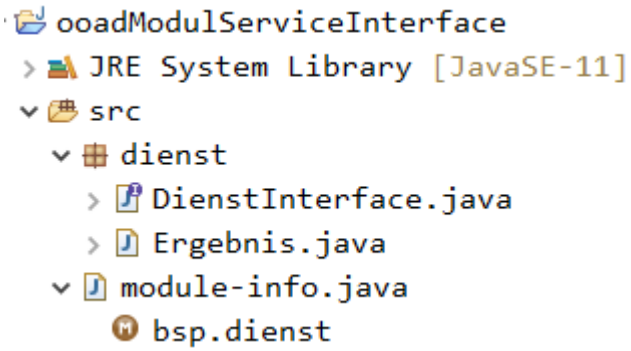


Ein Service ist ein einfaches Interface, dass generell in einem beliebigen Paket liegen kann, typisch zusammen mit seinen Hilfsklassen, hier Ergebnis, und in ein eigenes Modul ausgelagert wird.

```
public interface DienstInterface {
	public Ergebnis mach1(int x, int y);
	public int qualitaet();

}
```

Der Modul-Deskriptor exportiert die notwendigen Pakete.

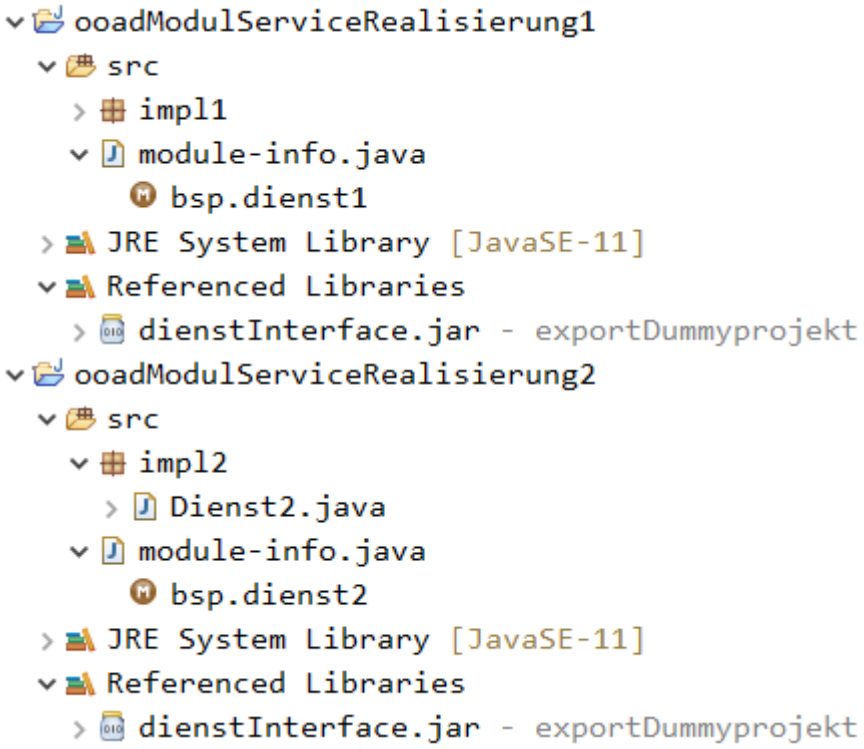
```
open module bsp.dienst {
	exports dienst;

}
```



Zum Modul wird, wie vorher beschrieben eine Jar-Datei erzeugt.

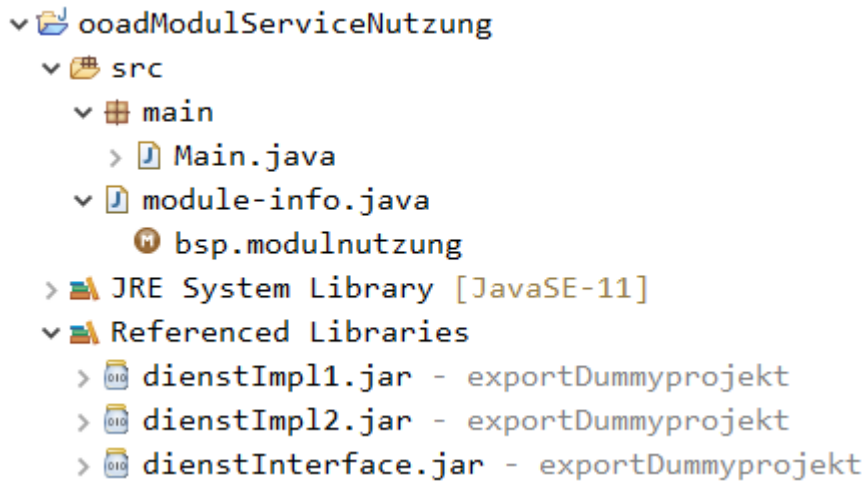
Die Modulimplementierungen in den Paketen impl1 und impl2 realisieren jeweils das Interface. Dies wird explizit auch im Modul-Deskriptor vermerkt. Dazu dient das Schlüsselwort „provides“, dem folgt der Name der implementierenden Klasse, gefolgt vom Schlüsselwort „with“ und einer kommaseparierten Liste der realisierten Services, also Interfaces. Es wird jeweils eine Jar-Datei erzeugt.

```
open module bsp.dienst1 {
	requires transitive bsp.dienst;
	exports impl1;

	provides dienst.DienstInterface with impl1.Dienst1;

}


open module bsp.dienst2 {
	requires transitive bsp.dienst;
	exports impl2;

	provides dienst.DienstInterface with impl2.Dienst2;

}
```



Der Service-Nutzer muss im Modul-Deskriptor dies explizit mit dem Schlüsselwort „uses“ gefolgt vom Namen des Services, also Interface, angeben.

```
open module bsp.modulnutzung {
      requires bsp.dienst1;
      requires bsp.dienst2;

      uses dienst.DienstInterface;
}
```



Die Besonderheit ist, dass im Programm abgefragt werden kann, welche Serviceimplementierungen es gibt und die JVM Service-Objekte zur Verfügung stellt, ohne

dass diese vom Nutzer selbst erzeugt werden müssen. Der Ansatz kann als eine Variante von Dependency Injection angesehen werden.

```
package main;

import java.util.ServiceLoader;
import dienst.DienstInterface;
import dienst.Ergebnis;

public class Main {
  public static void main(String[] args) {
    ServiceLoader<DienstInterface> sl
        = ServiceLoader.load(DienstInterface.class);
    for(DienstInterface di: sl) {
      Ergebnis erg = di.mach1(1, 42);
      System.out.println("Service: " + di.getClass()
          +" Q: " + di.qualitaet() + " erg: " + erg.getText());
    }
  }
}
```

Eine mögliche Ausgabe kann wie folgt aussehen.

```
Service: class impl2.Dienst2 Q: 42 erg: 42
Service: class impl1.Dienst1 Q: 20 erg: 43
```

Neben der direkten Implementierung des Interfaces besteht die Möglichkeit eine Factory-Methode zu nutzen. Diese Methode liefert dann genau ein Objekt als Ergebnis, das die Schnittstelle des Service realisiert. Die Klassenmethode muss wie folgt aussehen.

```
public static „Service“ provider()
```

Im Moduldekriptor wird wieder angegeben, werden den Service realisiert.

```
module bsp.dienstFactory {
  requires transitive bsp.dienst;
  exports impl3;
  provides dienst.DienstInterface with impl3.DienstFactory;
}
```

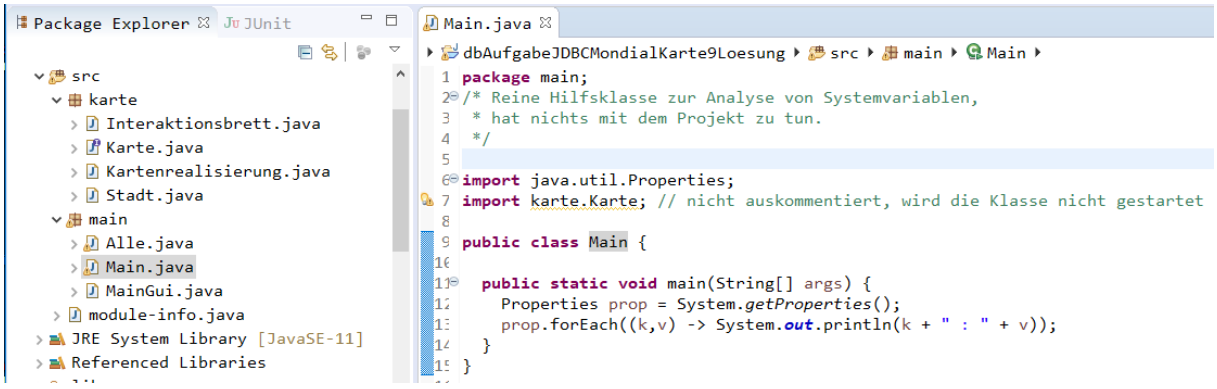
Eine Beispielklasse könnte wie folgt aussehen. Natürlich könnten z. B. weitere Informationen zur Objekterstellung aus einer Konfigurationsdatei gelesen werden.

```
package impl3;

import dienst.DienstInterface;
import dienst.Ergebnis;

public class DienstFactory {

     public static DienstInterface provider() {
```

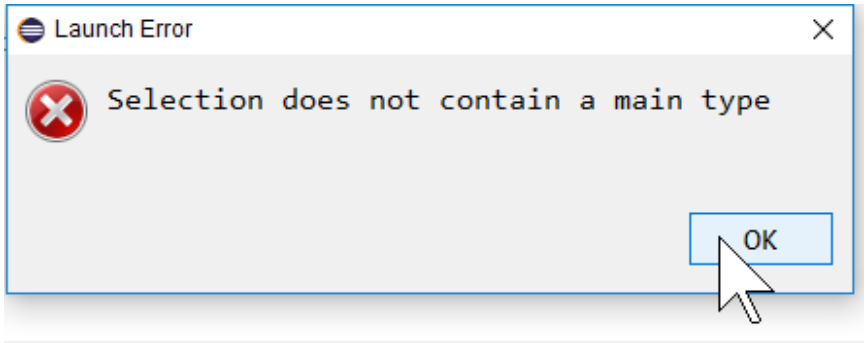
```
            return new DienstInterface() {

                  @Override
                  public Ergebnis mach1(int arg0, int arg1) {
                        return new Ergebnis();
                  }

                  @Override
                  public int qualitaet() {
                        return 100;
                  }
            };
      }
}
```

7.13 Startproblem: „Selection does not contain a main type“

Gerade im Zusammenhang mit der Modularisierung kann es passieren, dass der einfache Start einer Klasse mit einer korrekten main-Methode nicht funktioniert. Die folgende Abbildung zeigt eine einfache Klasse zur Ausgabe vom Systemvariablen, die einen überflüssigen import enthält, der natürlich die Ausführbarkeit nicht beeinflusst.



Wird die Klasse jetzt über einen Rechtsklick mit „Run As > Java Application“ gestartet, wird folgende Fehlermeldung ausgegeben.



Wird jetzt nur der überflüssige import auskommentiert, läuft das Programm ohne Probleme.

Main.java

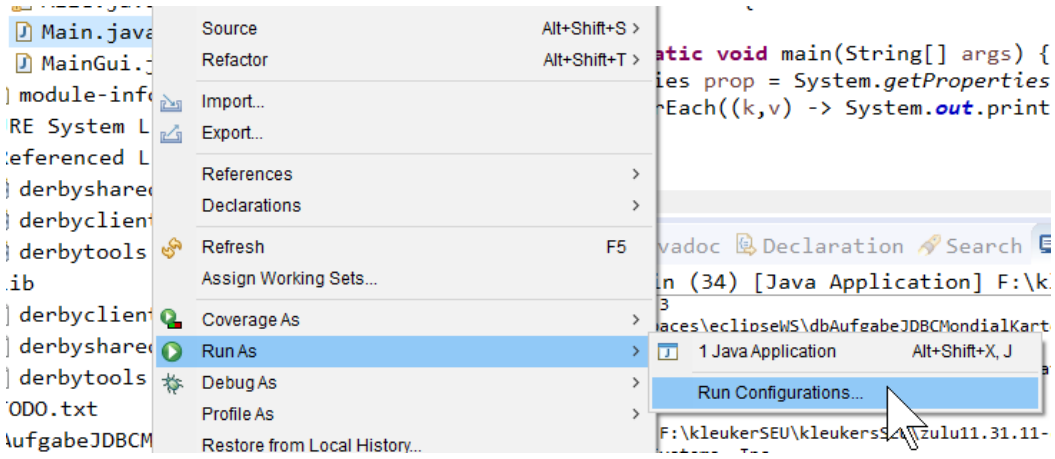
dbAufgabeJDBCMondialKarte9Loesung ▸ src ▸ main ▸ Main ▸

```
1  package main;
2  /* Reine Hilfsklasse zur Analyse von Systemvariablen,
3   * hat nichts mit dem Projekt zu tun.
4   */
5
6  import java.util.Properties;
7  //import karte.Karte; // nicht auskommentiert, wird die Klasse nicht gestartet
8
9  public class Main {
10
11   public static void main(String[] args) {
12     Properties prop = System.getProperties();
13     prop.forEach((k,v) -> System.out.println(k + " : " + v));
14   }
15 }
```

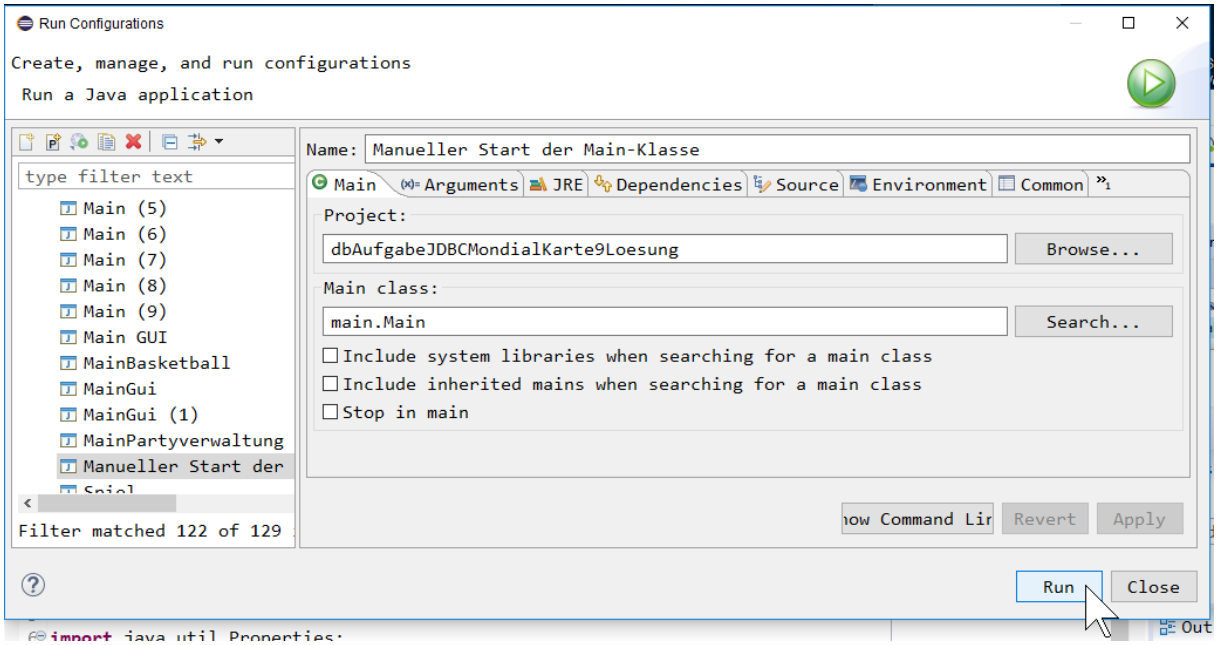
Problems | Javadoc | Declaration | Search | Console | Coverage | Call Hierarchy

```
<terminated> Main (34) [Java Application] F:\kleukerSEU\kleukersSEU\zulu11.31.11-ca-
java.version : 11.0.3
user.dir : F:\workspaces\eclipseWS\dbAufgabeJDBCMondialKarte9Loesung
os.arch : amd64
java.vm.specification.name : Java Virtual Machine Specification
java.awt.printerjob : sun.awt.windows.WPrinterJob
sun.os.patch.level :
java.library.path : F:\kleukerSEU\kleukersSEU\zulu11.31.11-ca-fx-jdk11.0.3-win_x64\bin;C:\WINDOWS\Sun\Java\b
java.vendor : Azul Systems, Inc.
java.vm.info : mixed mode
java.vm.version : 11.0.3+7-LTS
sun.io.unicode.encoding : UnicodeLittle
java.class.version : 55.0
```

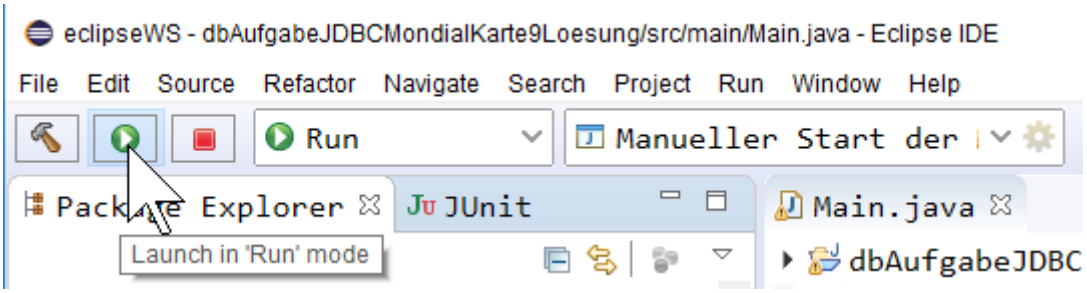
Hier liegt leider ein Eclipse-Fehler vor, der zumindest teilweise erst in der Version 4.14 (2019-12) behoben wurde. Als Workaround klappt meist der Versuch explizit eine Start-Konfiguration zu erstellen, wie es auch in „“ beschrieben ist. Es wird z. B. ein Rechtsklick auf der Klasse gemacht und „Run As > Run Configurations …“ ausgewählt.



Es wird oben hinter „Name:“ ein sprechender Name für die Konfiguration eingegeben. Es muss das passende Projekt ausgewählt sein. Der Wert kann meist übernommen werden. Unter „Main class“ wird von Hand der vollqualifizierte Name der Klasse eingetragen. Die Konfiguration wird mit „Apply“ zunächst gesichert, um sie dann mit „Run“ auszuführen.



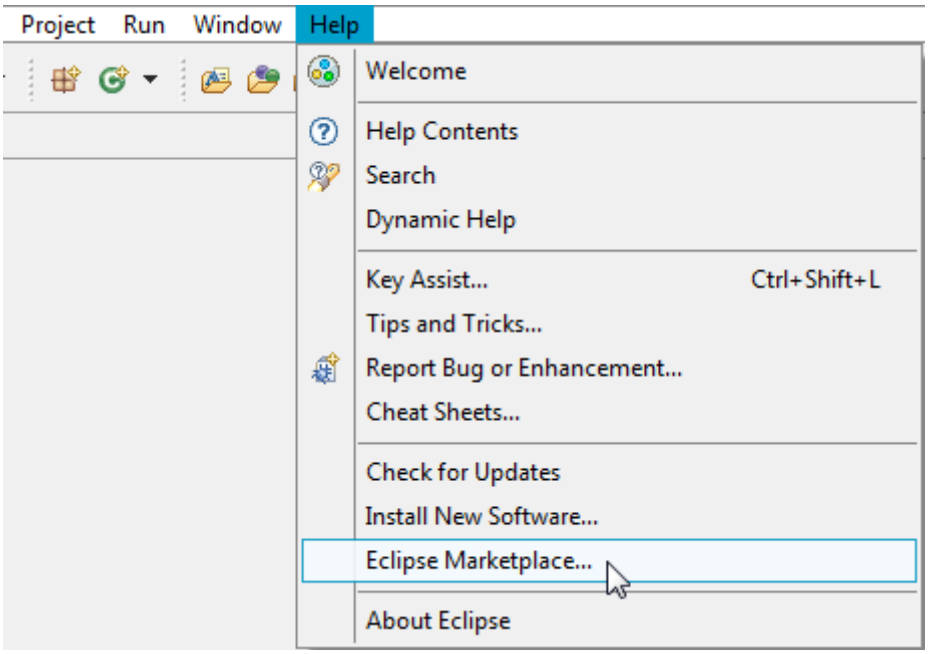
Die Konfigurationen sind danach auch oben über den Launcher auswählbar und werden durch einen Klick auf das Run-Symbol gestartet.



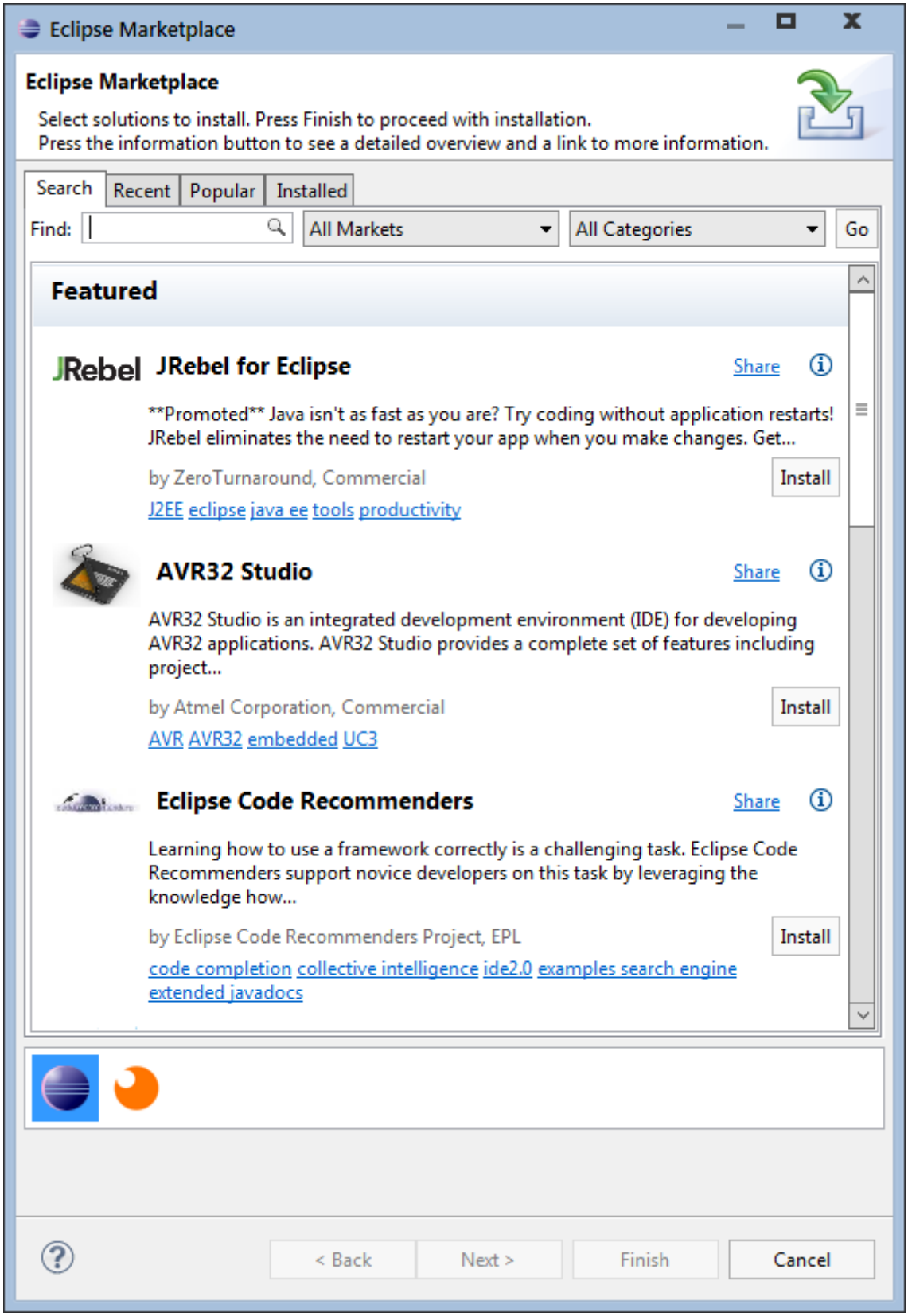
8 Installation von Plugins über Marketplace (4.11.0) am Beispiel SVN

Vergleichbar zu Betriebssystemen, beginnend mit Android und iOS hat sich dar Ansatz durchgesetzt, sogenannte Marktplätze für Kunden anzubieten, in denen sie Programme bzw. Erweiterungen erwerben können. Diese Idee wird auch mit Eclipse umgesetzt und stellt eine zusätzliche Variante dar, Programmerweiterungen zu installieren. Dieser Ansatz wird hier exemplarisch für die Suche nach einer Integration des Versionsmanagementsystems Subversion (SVN) beschrieben.

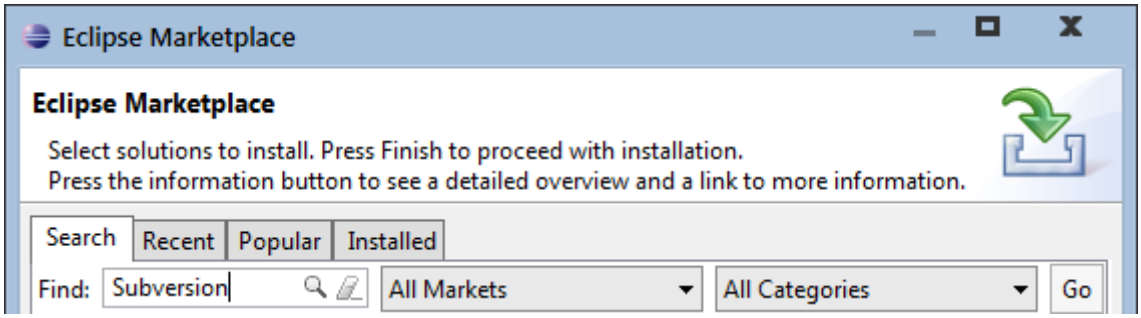
Der Market Place befindet sich unter „Help > Eclipse Marketplace“



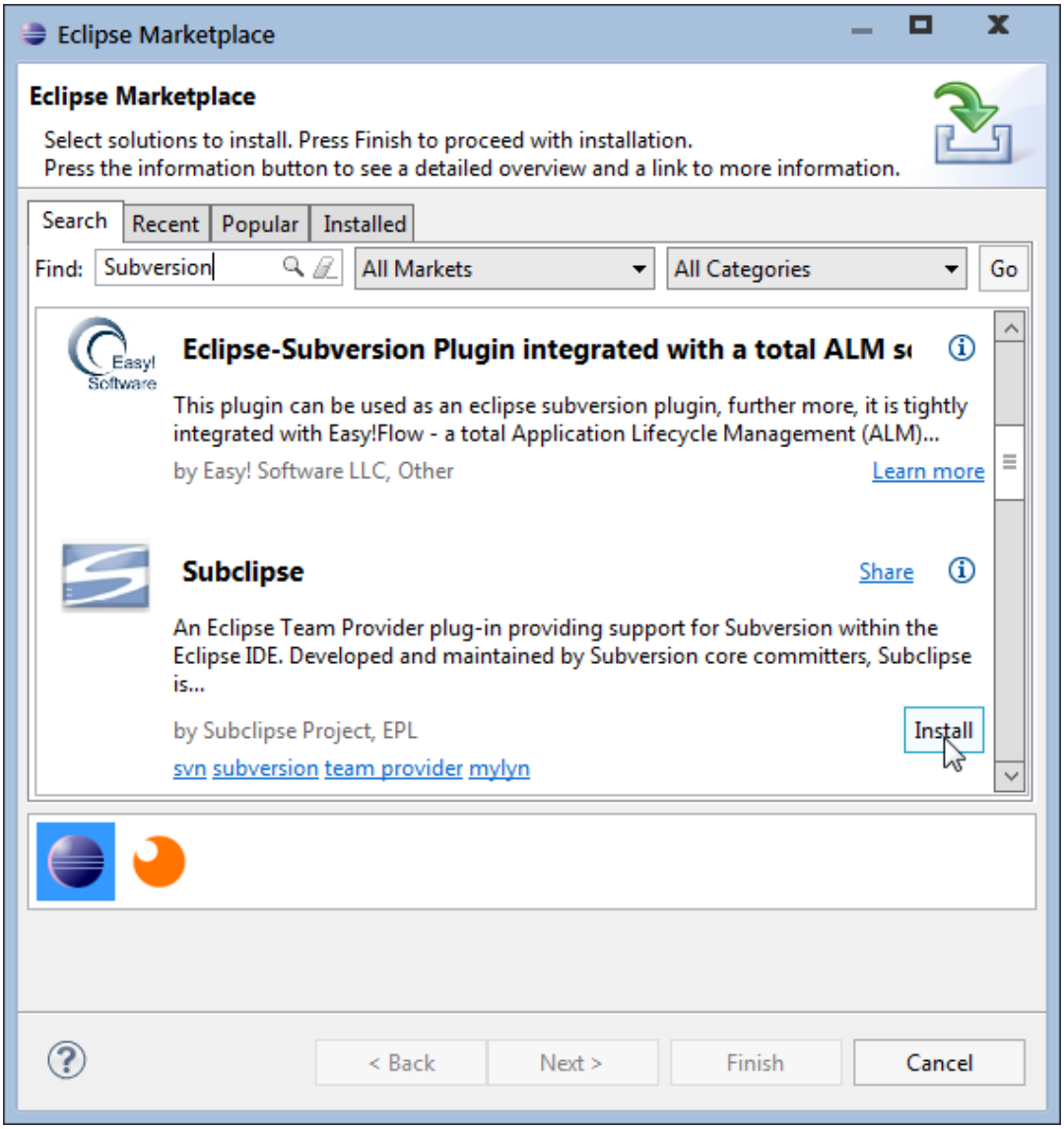
Am Anfang wird eine Menge von Angeboten, auch vielen kommerziellen, geladen. Das Stöbern ist für Entwicklungsumgebungen eher untypisch, kann aber auch zu interessanten Ergebnissen führen. Trotzdem sei daran erinnert, dass jede Erweiterung, auch wenn sie nicht genutzt wird, Eclipse verlangsamt.



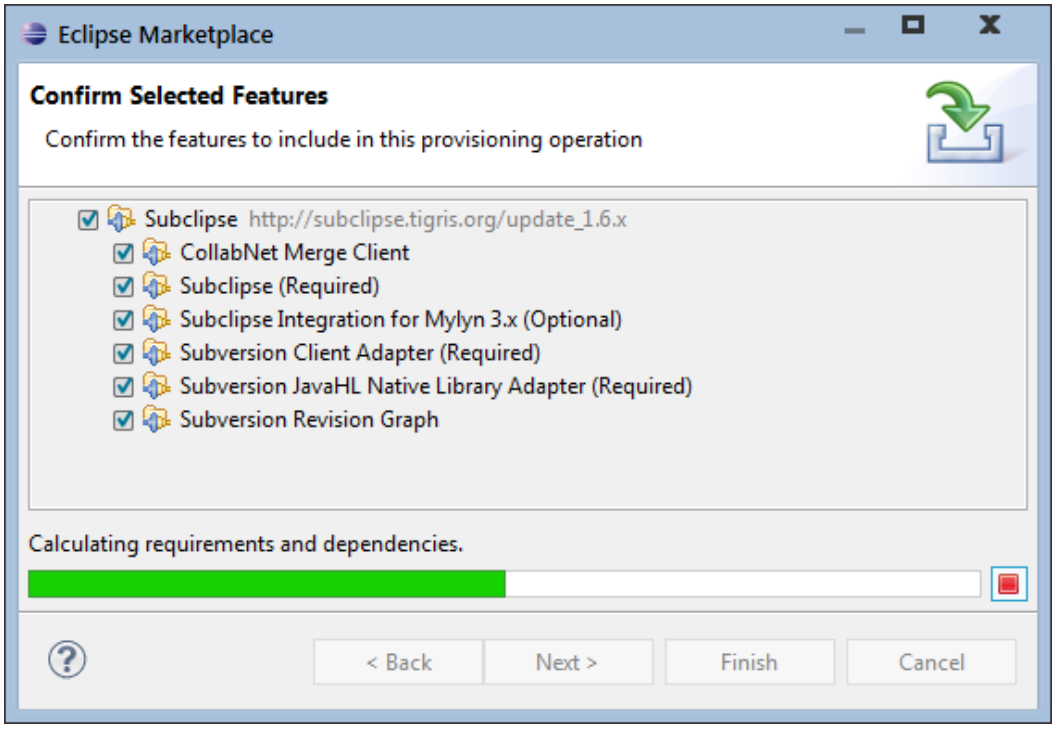
Die oberen Reiter bieten einige Möglichkeiten zur Suche, hier wird konkret der Reiter „Search“ genutzt, „Subversion“ eingegeben und die Suche gestartet.



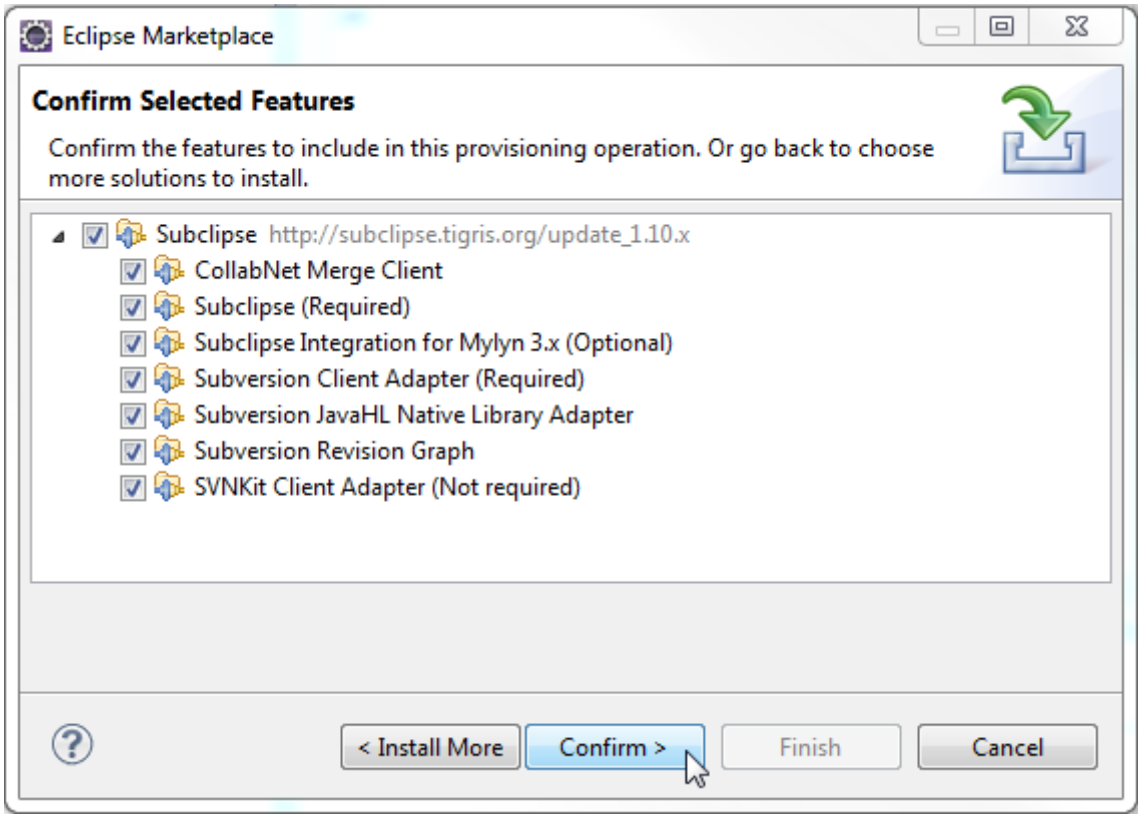
Man erhält eine größere Anzahl an Ergebnissen, die man ohne weitere Kenntnisse kaum nutzen kann. Neben den Lizenzen werden auch die Hersteller und ein Kommentar zur Funktionalität angegeben. Das hier sinnvolle Plugin Subclipse (Alternative wäre Subversive) befindet sich dabei eventuell nicht an erster Stelle. In Projekten ist es deshalb auch üblich, dass konkrete Plugins vorgegeben werden, die dann auch mit anderen Ansätzen installiert werden können (siehe Kapitel 9). Zur Installation wird der „Install“-Knopf auf der rechten Seite gedrückt.



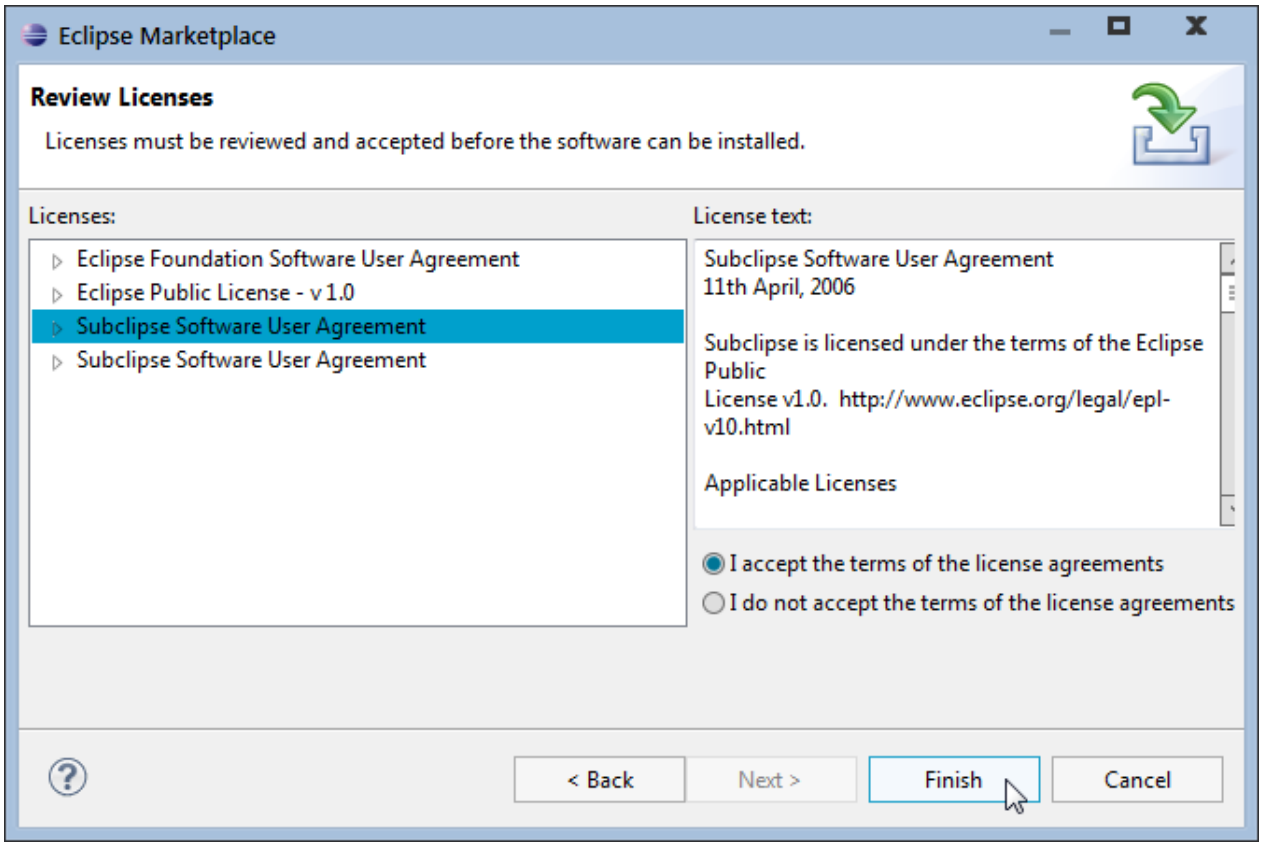
Es werden Abhängigkeiten von weiteren Erweiterungen berechnet, die dann zusammen installiert werden können.



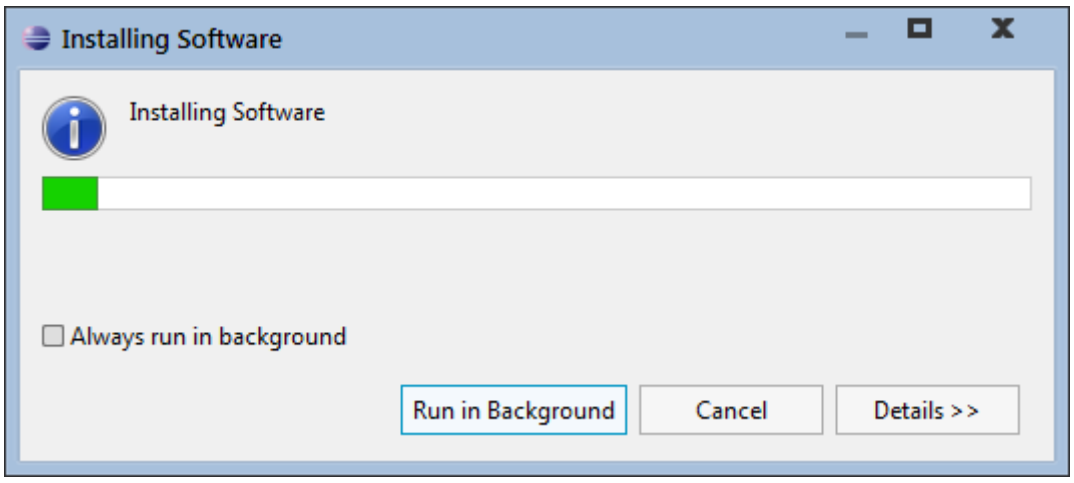
Die Berechnung kann dauern, danach wird einfach „Confirm>“ gedrückt.

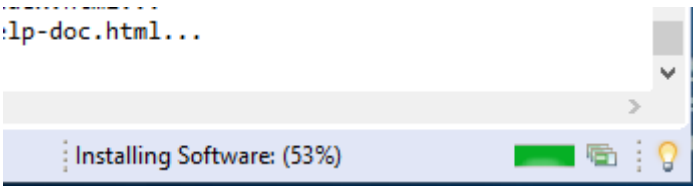


Es werden die zugehörigen Lizenzen auf der linken Seite eingeblendet, die man jeweils nach dem Lesen auf der rechten Seiten bestätigen („I accept…“) muss, danach wird „Finish“ gedrückt.

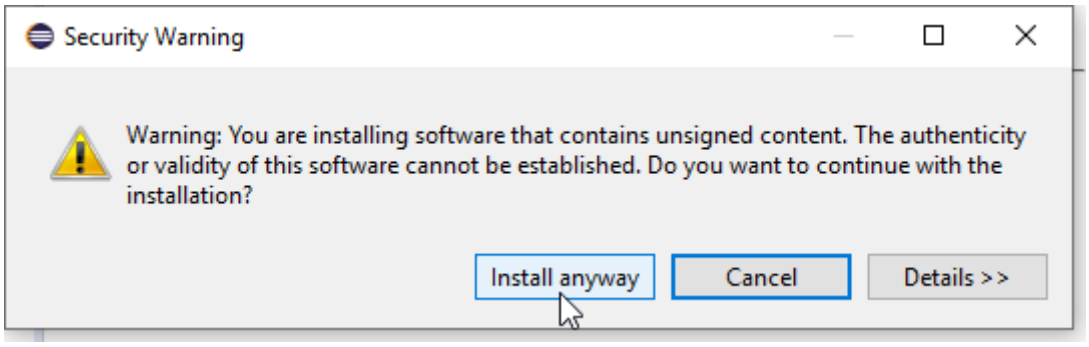


Die Installation kann wieder dauern.

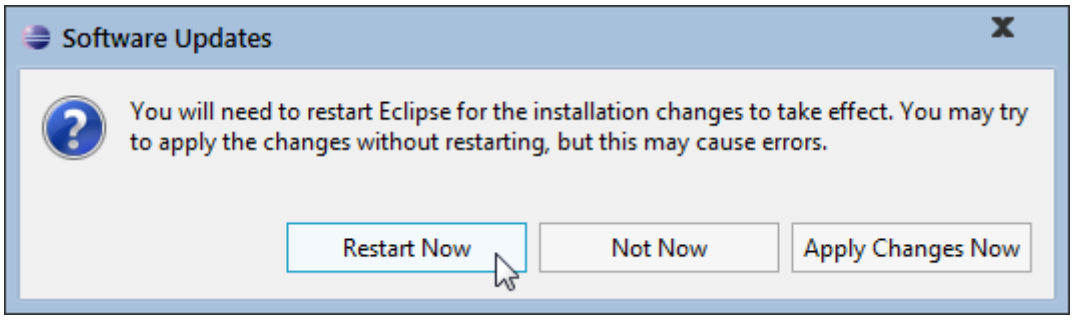




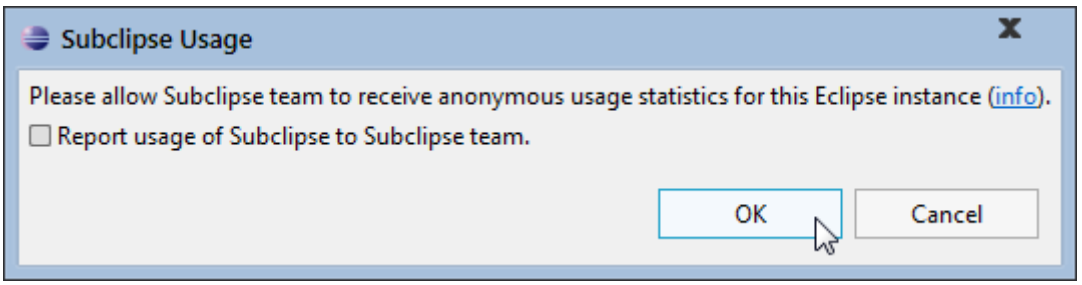
Da das Signieren von Software zur Sicherstellung, dass es sich nicht um Schad-Software handelt, Geld kostet, verzichten einige Unternehmen daruf, so dass man ggfls. folgende Anfrage bestätigen muss.



Generell ist es sinnvoll und meist auch notwendig, Eclipse neu zu starten. Nach dem Neustart sollte man prüfen, ob die Installation erfolgreich war.

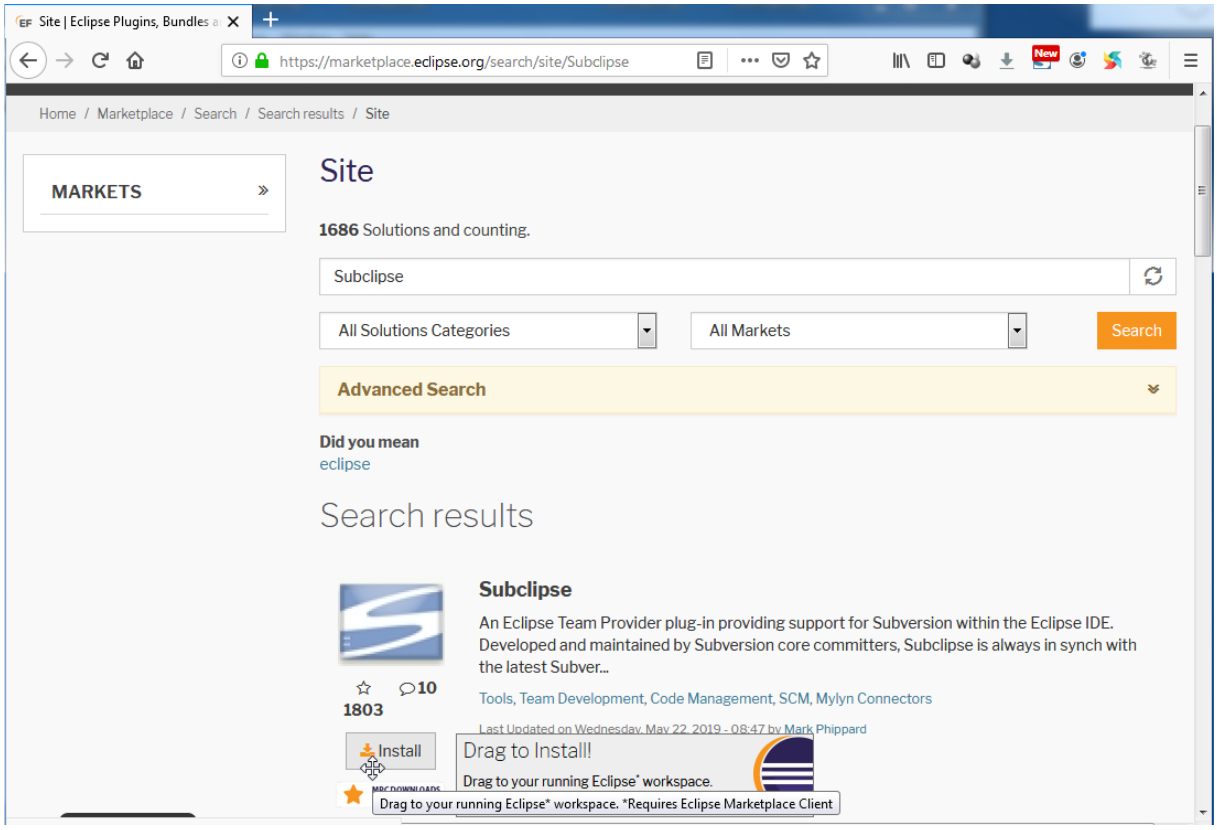


Eventuell gibt es beim Neustart Anmerkungen oder Fragen, die kritisch gelesen werden sollten.

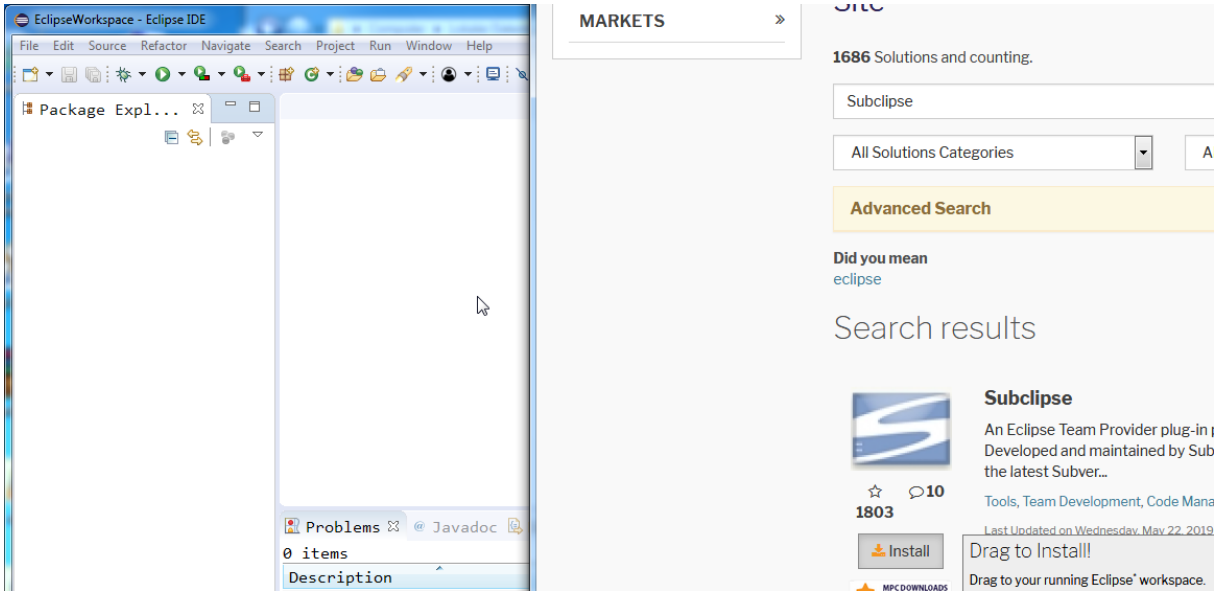


Weitere Informationen zur Nutzung von Subclipse stehen im Kapitel 16 .

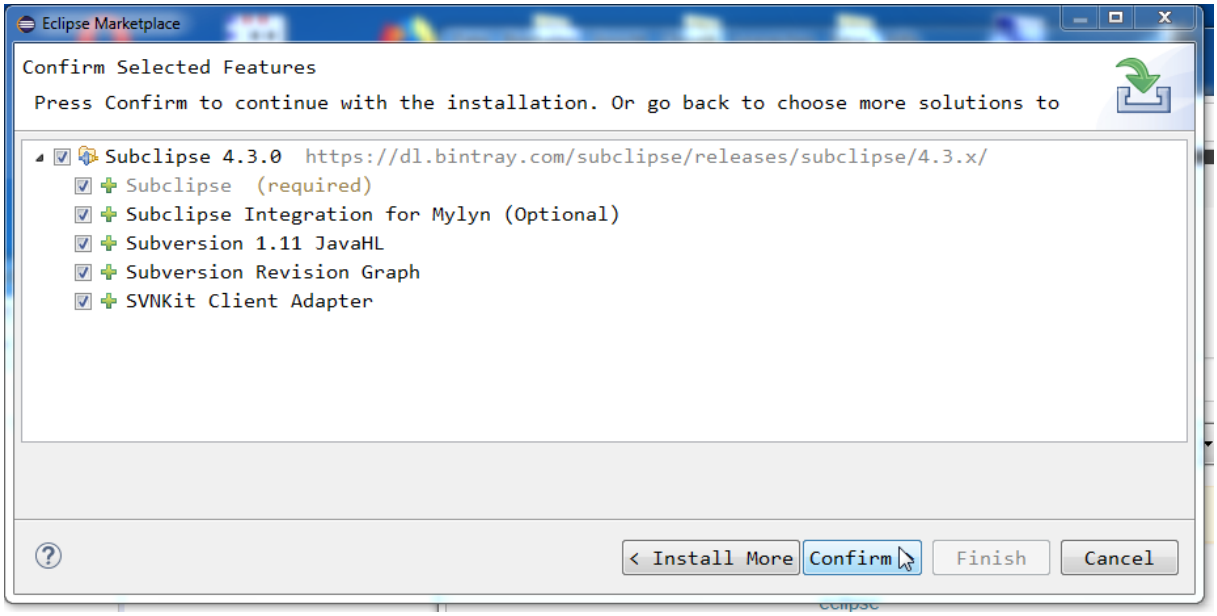
Die Installation kann auch über die Web-Seite des Marketplace erfolgen, was hier ebenfalls für Subclipse (https://marketplace.eclipse.org/search/site/Subclipse) gezeigt wird. Auf der Web-Seite ist ein Install-Button, der als Mouse-Over beschreibt, wie die Installation auch ablaufen kann.



Der Install-Button wird jetzt auf das geöffnete Eclipse mit gedrückter linker Maustaste gezogen.



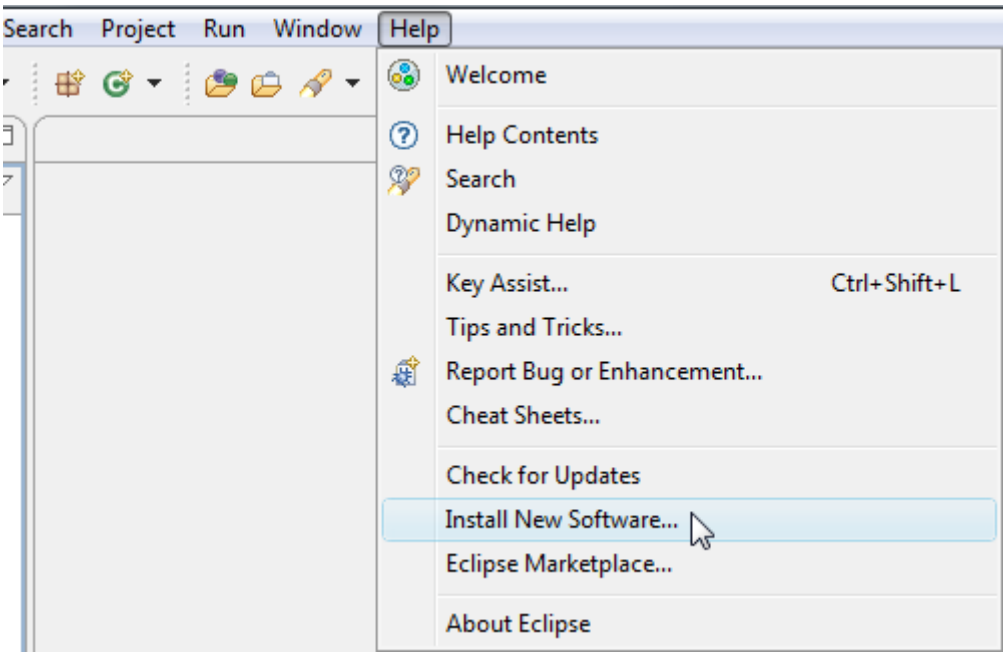
Es öffnet sich ein bereits bekannter Dialog, dem dann mit „Confirm>“ gefolgt wird.


Eclipse Marketplace
Confirm Selected Features
Press Confirm to continue with the installation. Or go back to choose more solutions to
Subclipse 4.3.0 https://dl.bintray.com/subclipse/releases/subclipse/4.3.x/
Subclipse (required)
Subclipse Integration for Mylyn (Optional)
Subversion 1.11 JavaHL
Subversion Revision Graph
SVNKit Client Adapter
< Install More
Confirm
Finish
Cancel

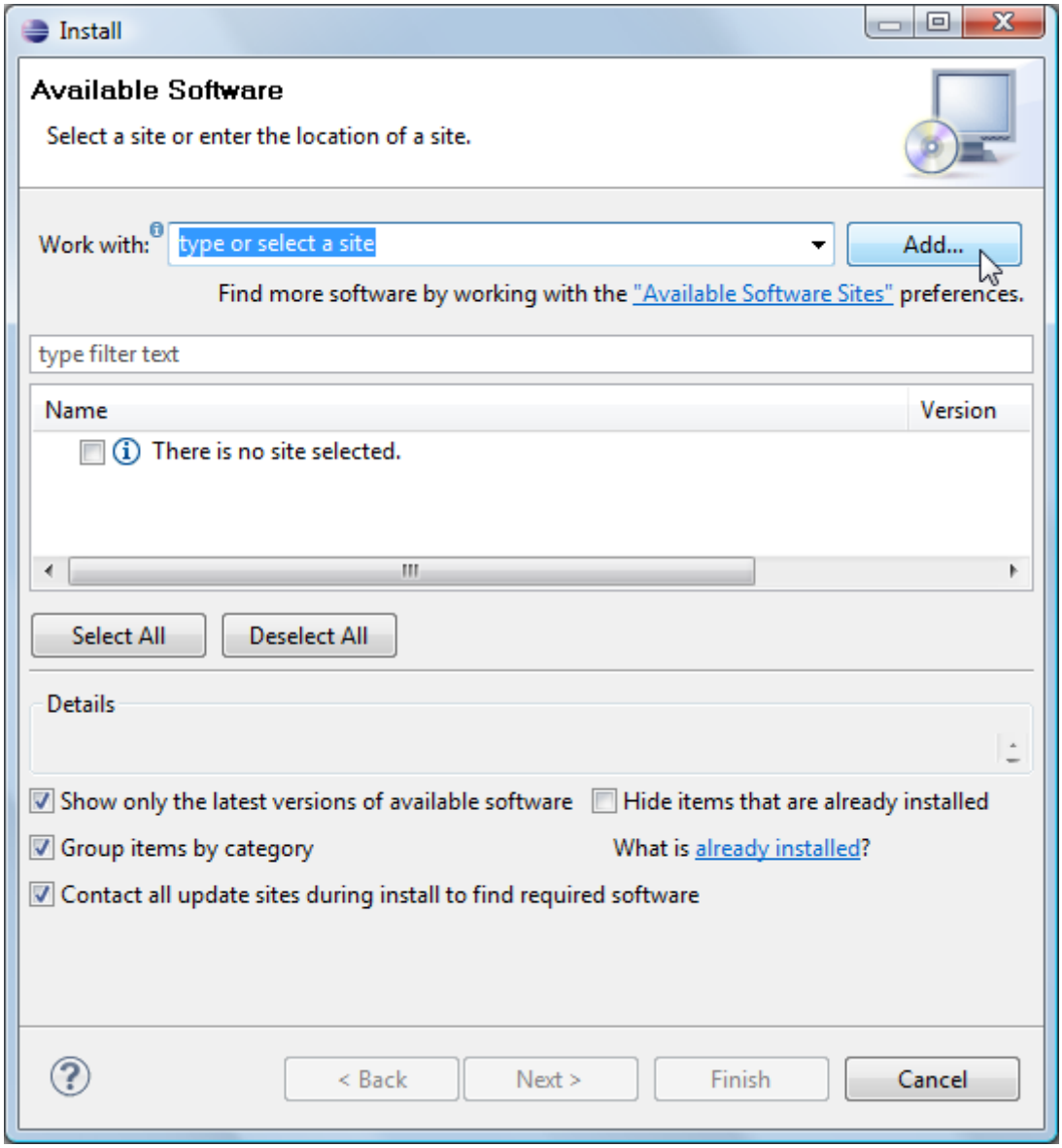
9 Direkte Installation von Plugins (4.11.0) am Beispiel TestNG

Formal gesehen ist Eclipse nur eine Basisplattform, die die einfache Zusammenarbeit verschiedener Werkzeuge 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.

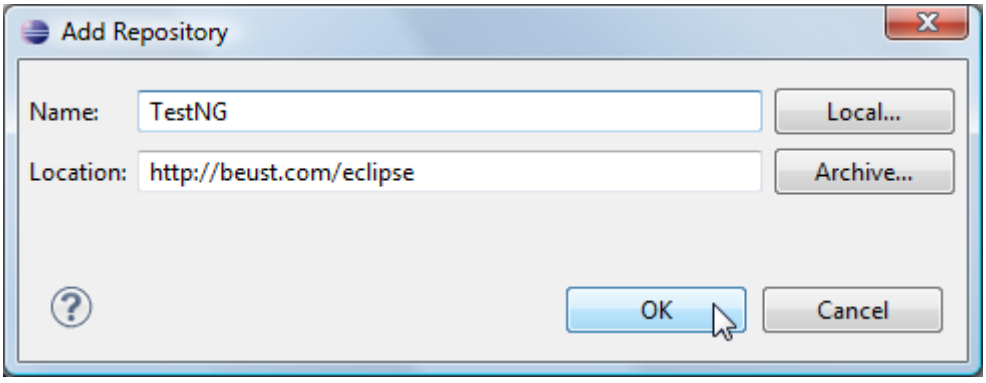
Gestartet wird die Installation über „Help>Install New Software...“.



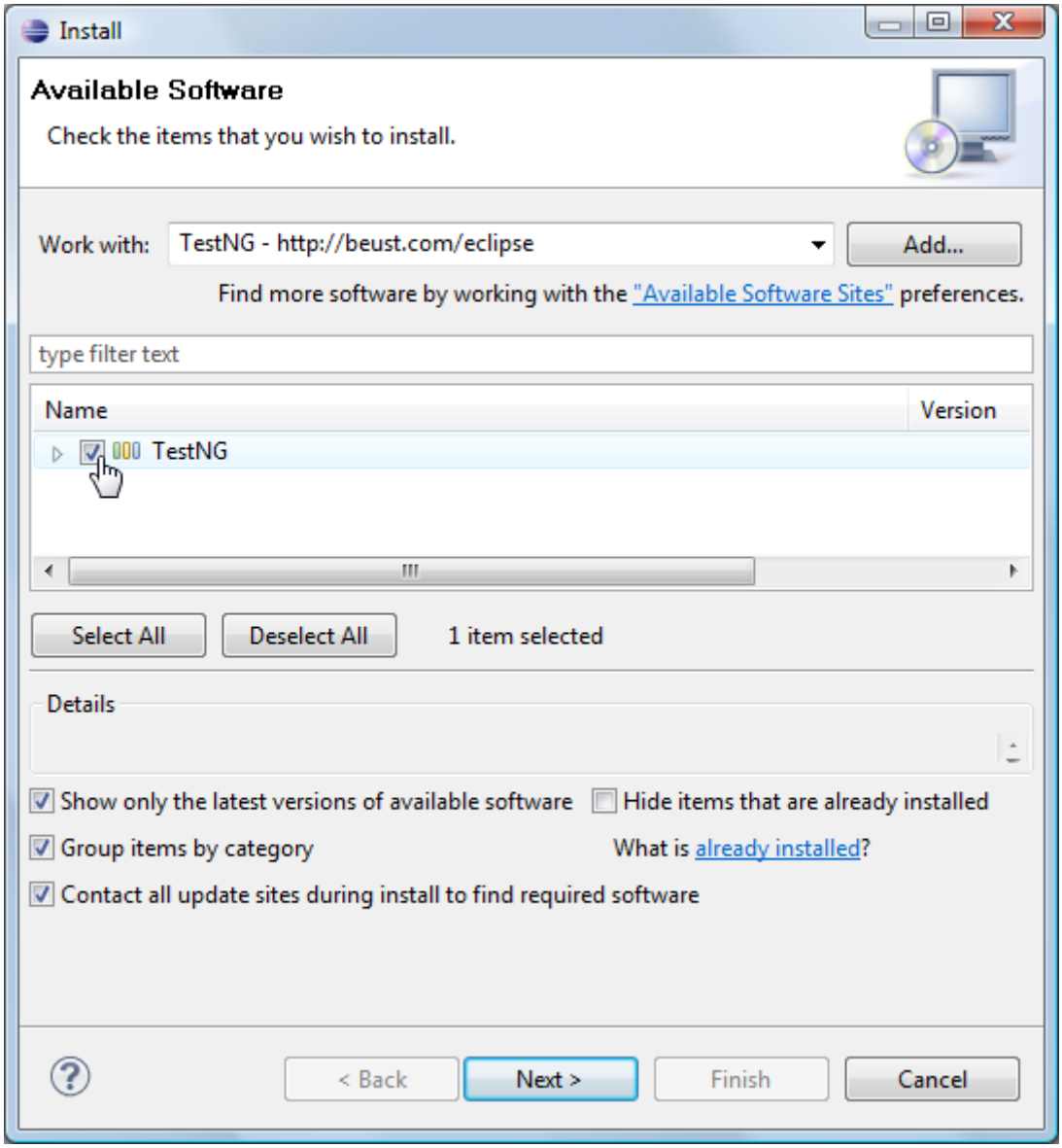
Hier gibt es zwei Möglichkeiten, da bereits einige Web-Seiten existieren, über die Plugins gefunden werden können. Alternativ hat man eine Update-Adresse zur Verfügung. Um eine solche Adresse zu nutzen, wählt man „Add“ am rechten Rand.



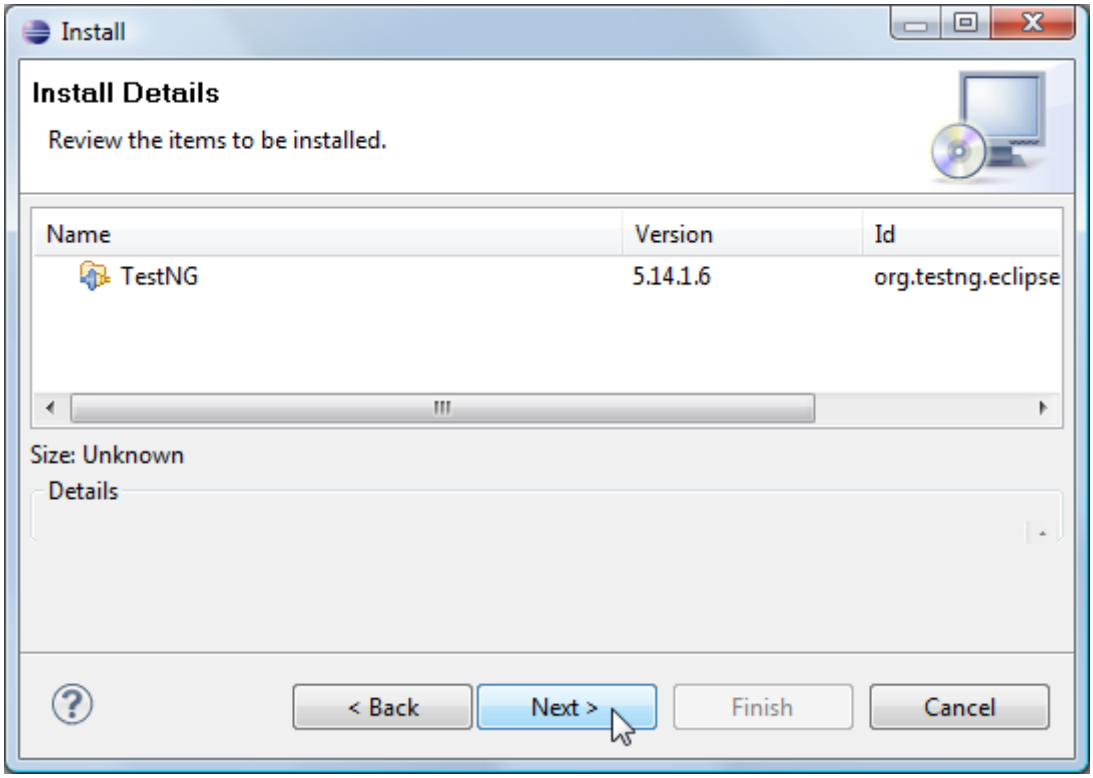
Hier kann man die Web-Adresse mit den Update-Informationen als „Location“, z. B. http://beust.com/eclipse eingeben. der dazu einzugebende Name sollte eindeutig sein und erleichtert die spätere Verwaltung. Der Eintrag wird mit „OK“ abgeschlossen.



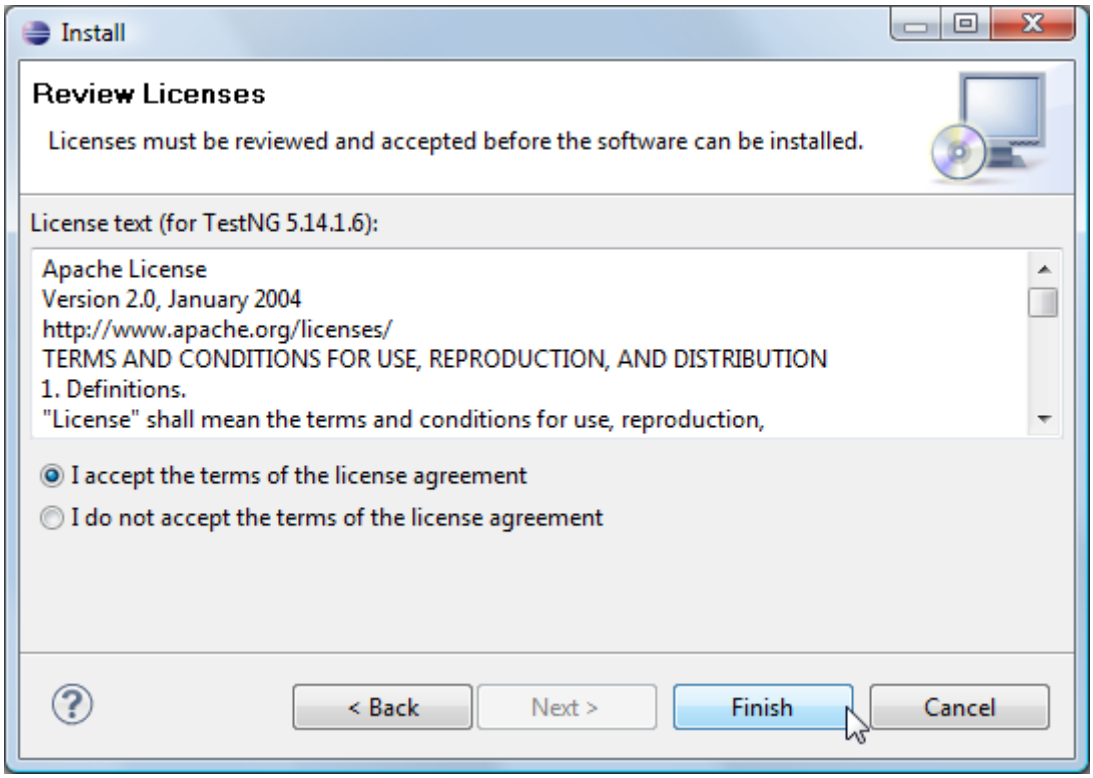
In der Tabelle (Name, Version) werden die gefundenen Erweiterungen angeboten und können mit einem Haken ausgewählt werden. Danach wird „Next>“ gedrückt. Die Schritte danach können bei verschiedenen Plugins etwas variieren, da nach abhängigen Plugins gesucht werden kann, die benötigt werden und Lizenzbedingungen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



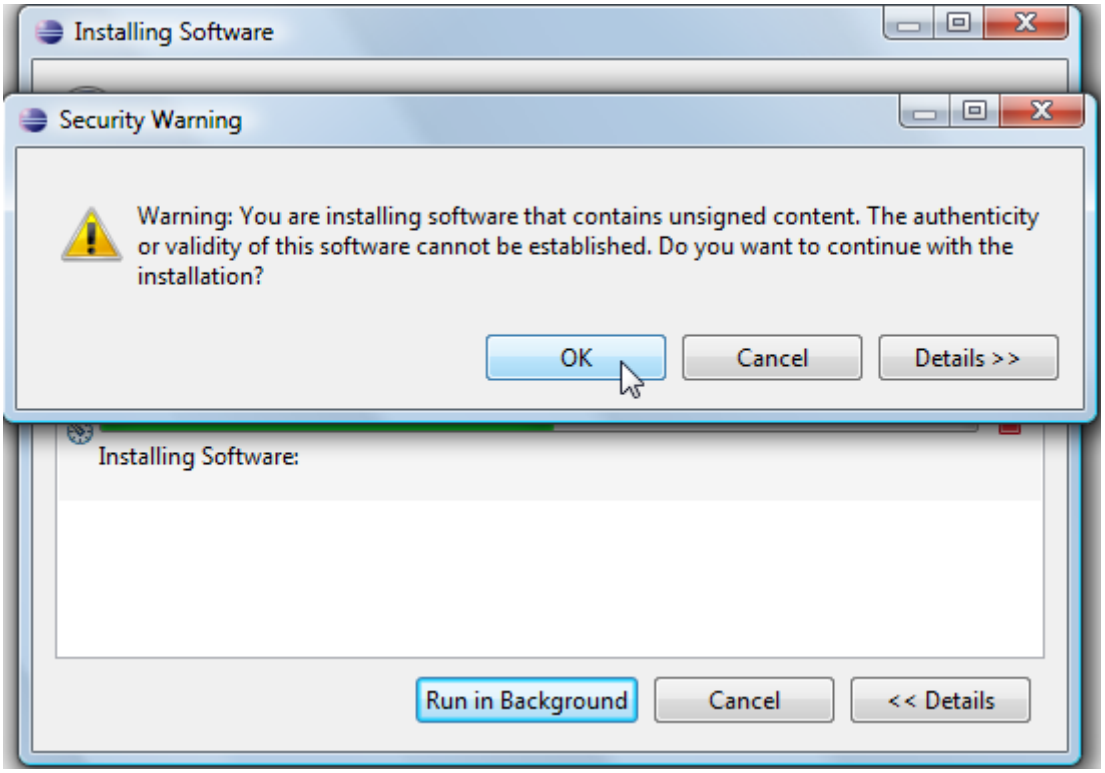
Man erhält eine Übersicht, was installiert werden soll und drückt „Next>“.



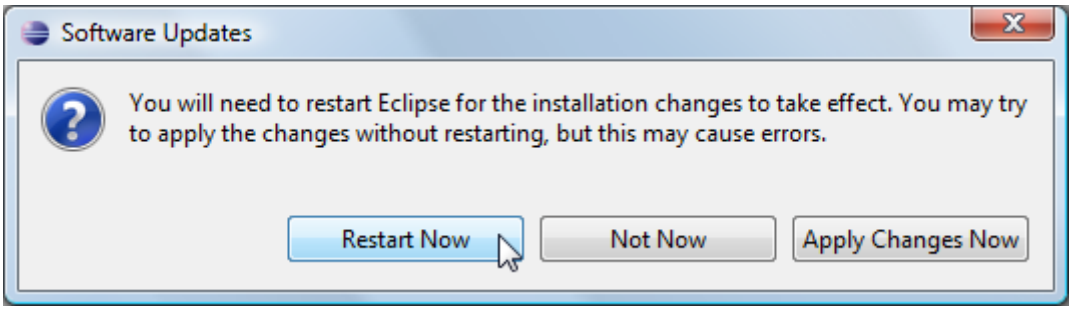
Nun müssen noch die Lizenzbestimmungen gelesen und angenommen und die Installation mit „Finish“ abgeschlossen werden.



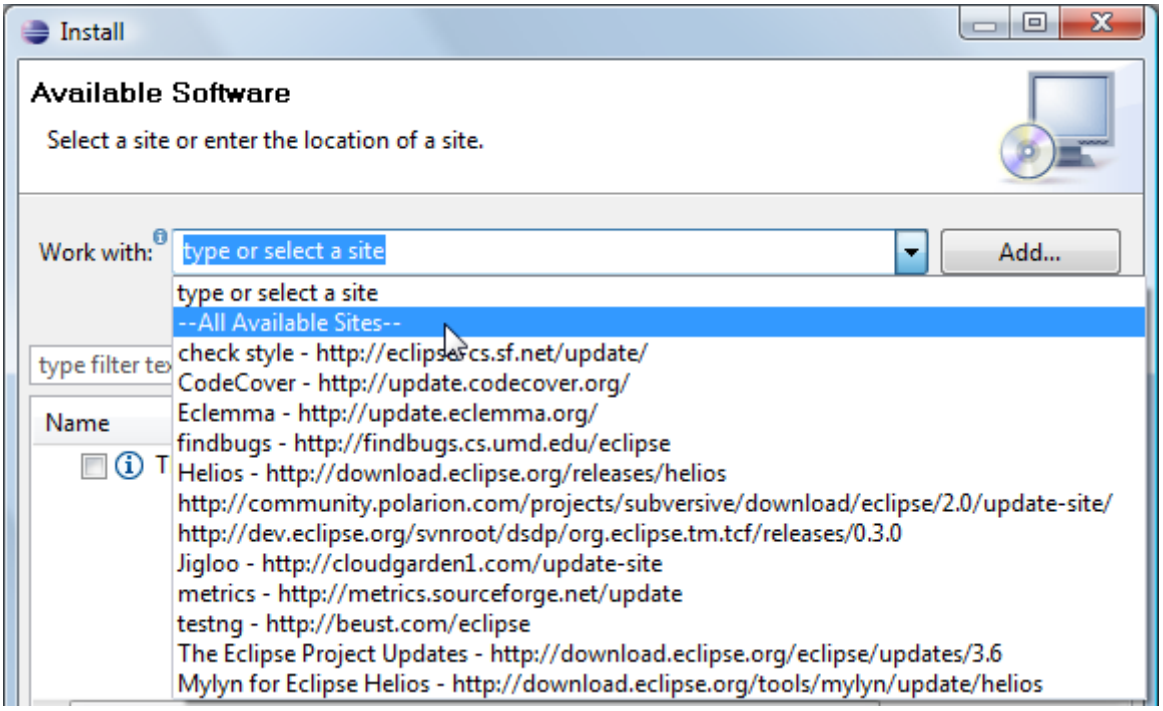
Da eine Signierung relativ teuer ist, wird sie häufiger nicht durchgeführt. Meist sollte man aber die Installation trotzdem mit einem „OK“ fortsetzen.



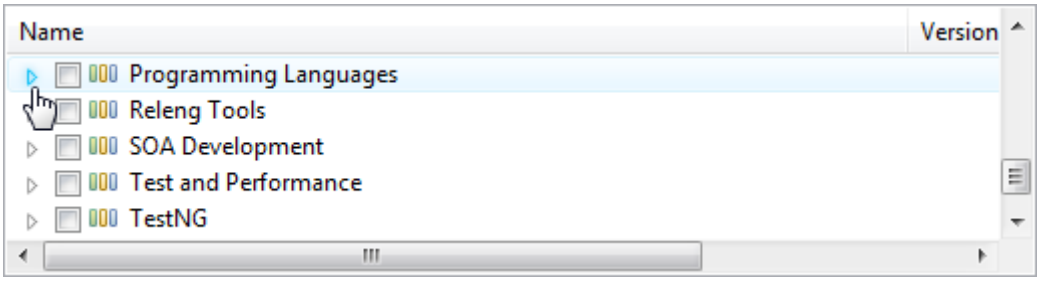
Zum Abschluss der Installation kann man wählen, ob Eclipse neu gestartet werden soll. Generell soll die Installation ohne Neustart möglich sein, trotzdem sollte man einen Neustart durchführen.



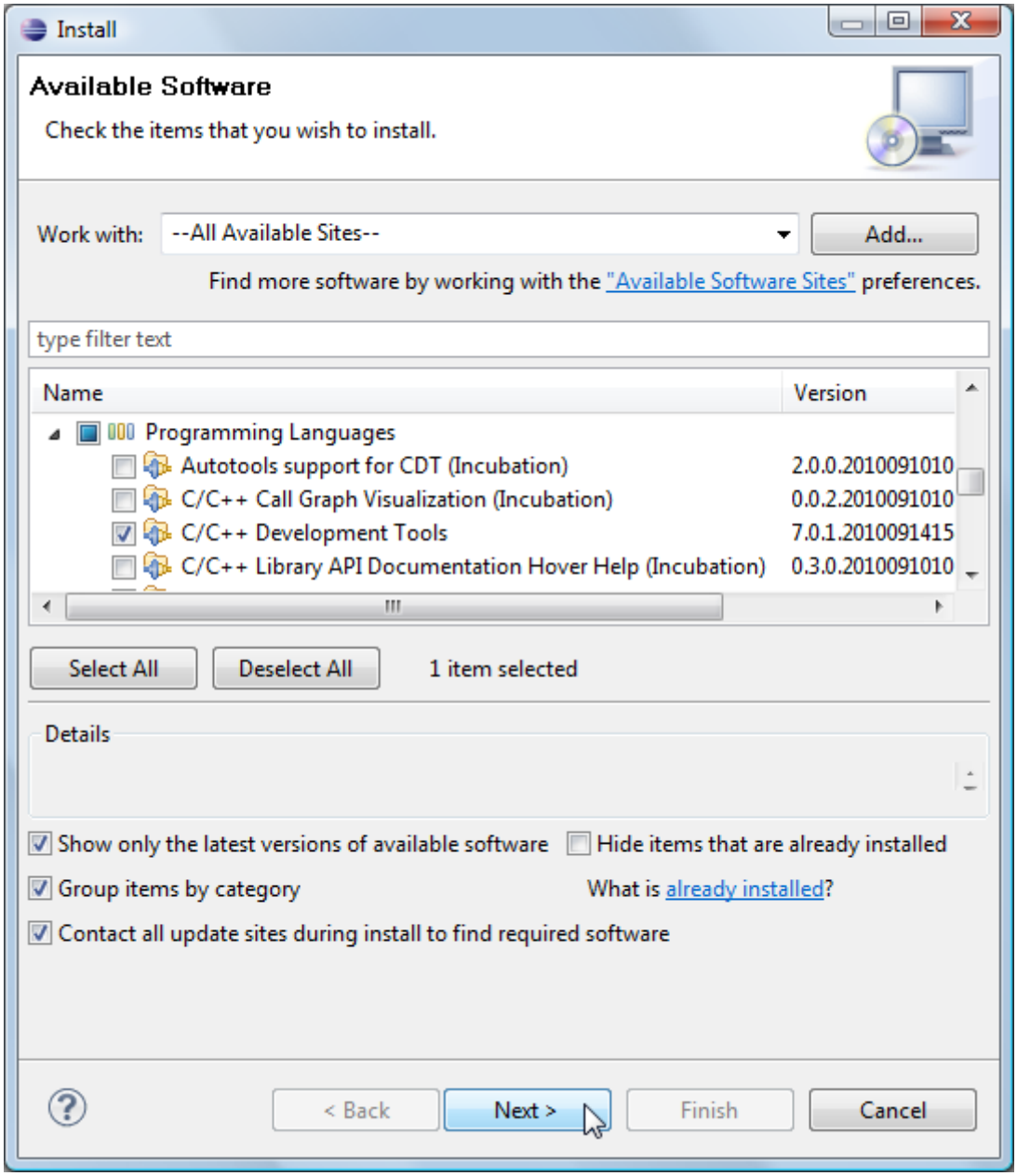
Bereits angedeutet, kann man auch nach Plugins auf bereits vorgegebenen Seiten suchen, was hier ebenfalls gezeigt werden soll. Hierzu wird beim „Install“-Fenster der kleine Pfeil rechts bei der „Work with:“ Box genutzt. Der genaue Inhalt des Fensters hängt von der Eclipse-Version und den bereits vorher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



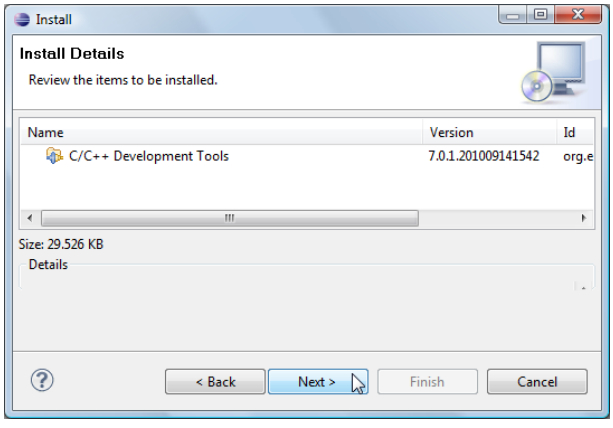
Jetzt ist die Übersicht mit den möglichen Plugins reich gefüllt und kann mit Haken die gewünschten Plugins auswählen. Man beachte, dass man direkt einen Haken setzen oder die Elemente, wie unten angedeutet, weiter aufklappen kann.



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



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

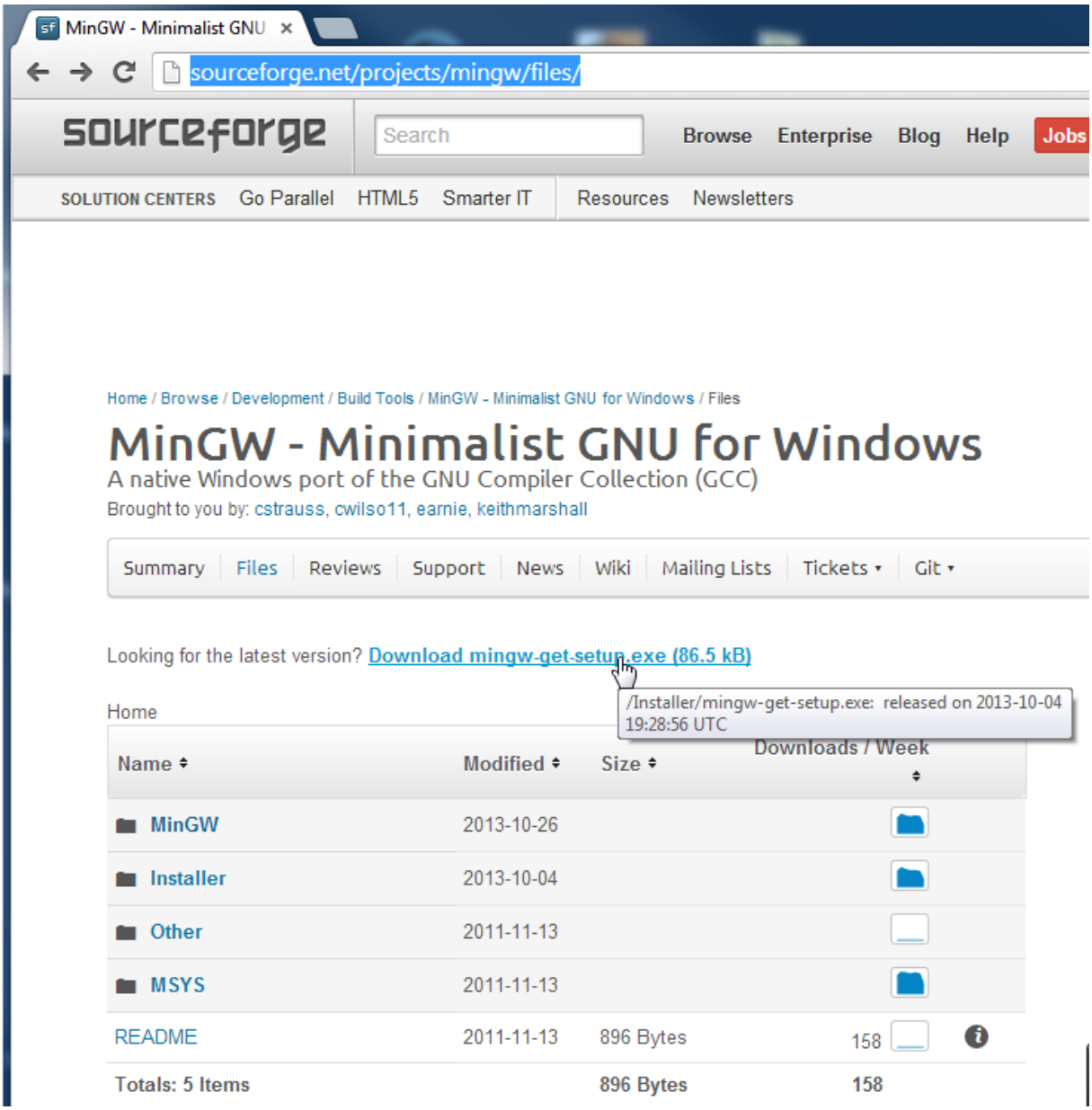


10 Einblick in die C und C++-Entwicklung (4.11.0)

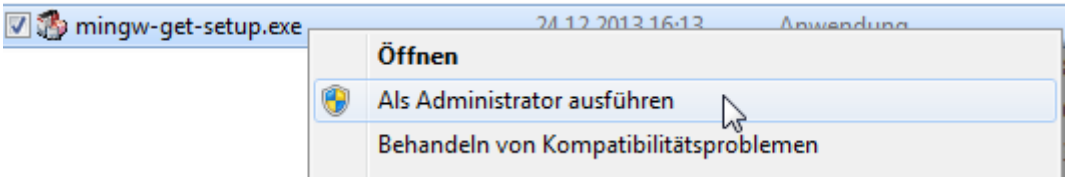
Gerade ab der Version 3.3 hat Eclipse einen wesentlichen Sprung zur vollwertigen C++-Entwicklungsumgebung gemacht, wobei allerdings der Ansatz älterer Versionen ziemlich stark geändert wurde. Zur Nutzung der Umgebung muss ein C++-Compiler installiert sein, dabei wird unter Windows der Gnu-Compiler, der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

10.1 Installation von MinGW

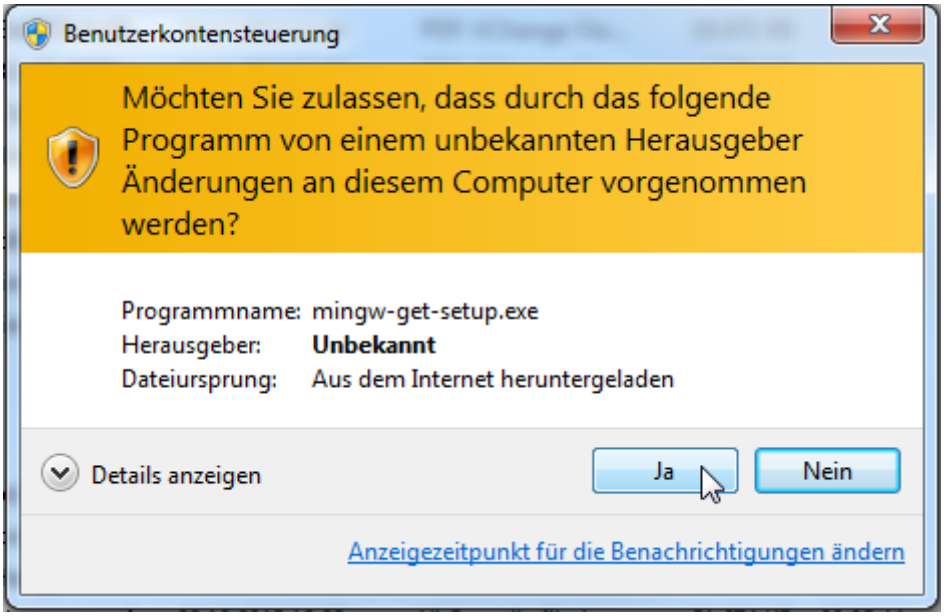
Hier wird ein Setup-Programm genutzt, es gibt alternative Installationsmöglichkeiten. Der Download findet von der Seite http://sourceforge.net/projects/mingw/files/ satt, hier wird der Download-Link in der oberen Zeile genutzt, alternativ muss das Programm im Ordner Installer gesucht werden.



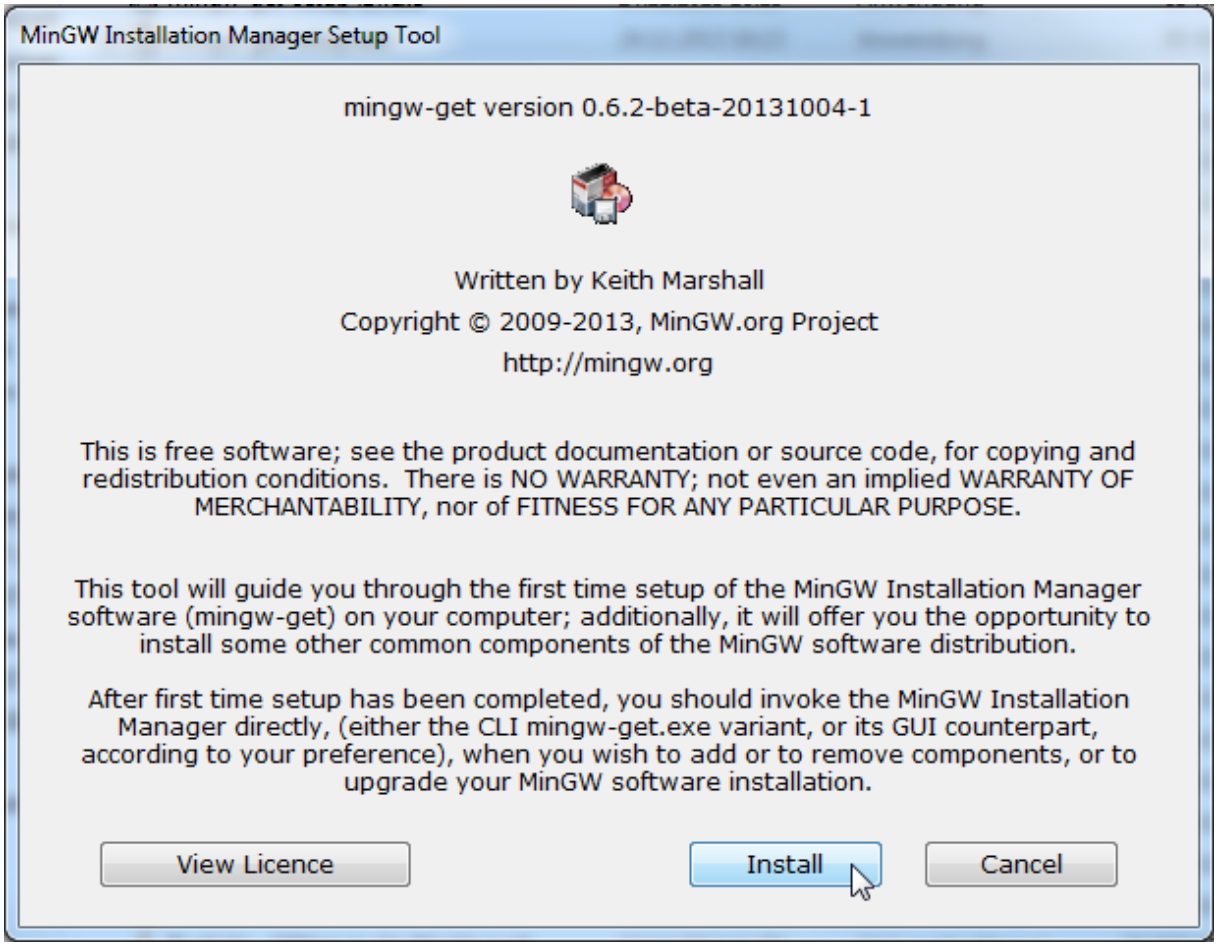
Die Installation wird hier über einen Rechtsklick und der Auswahl „Als Administrator ausführen“ gestartet. Wenn nicht möglich, ist auch eine Installation ohne Admin-Rechte möglich.



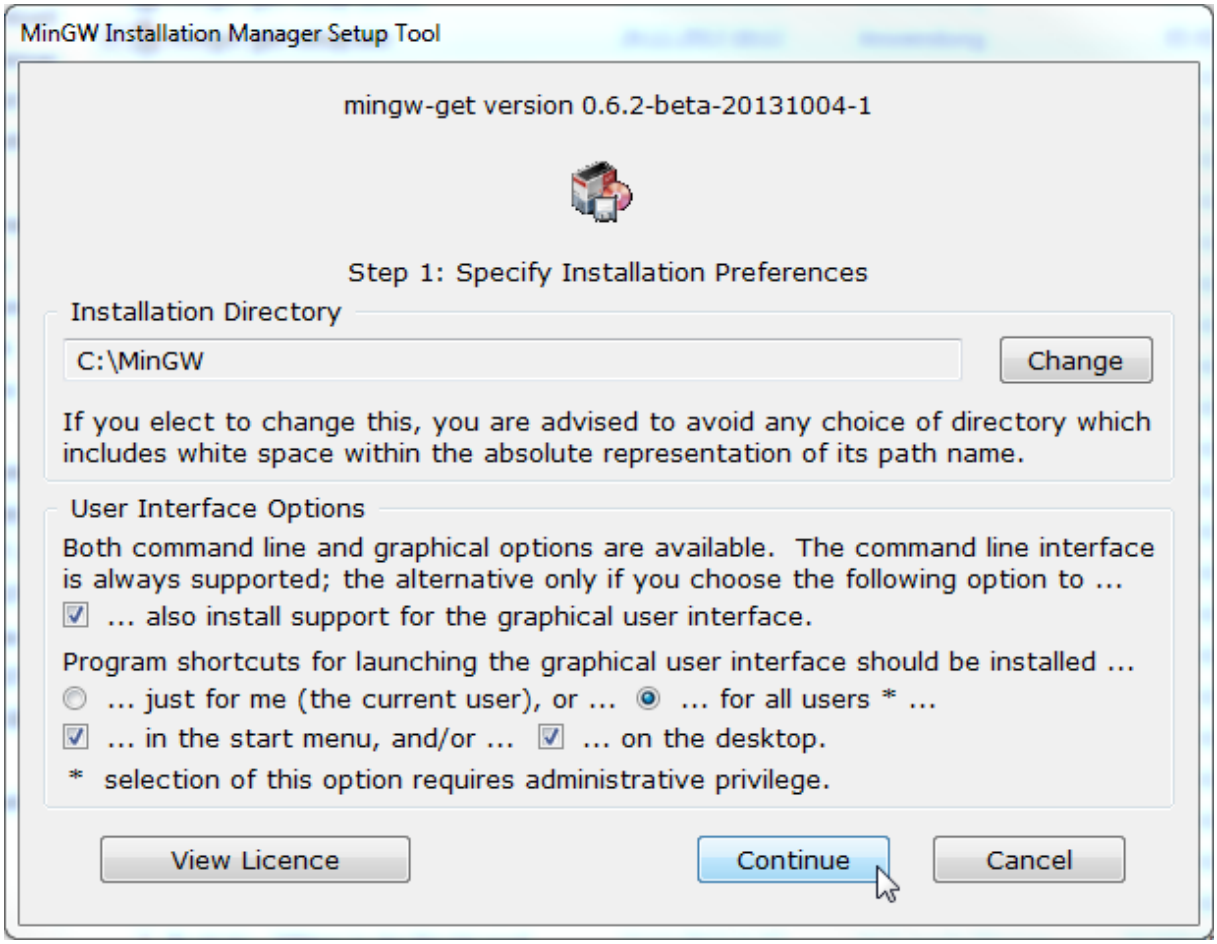
Abhängig von Sicherheitseinstellungen und Sicherheitsprogrammen müssen einige Rechte eingeräumt werden.



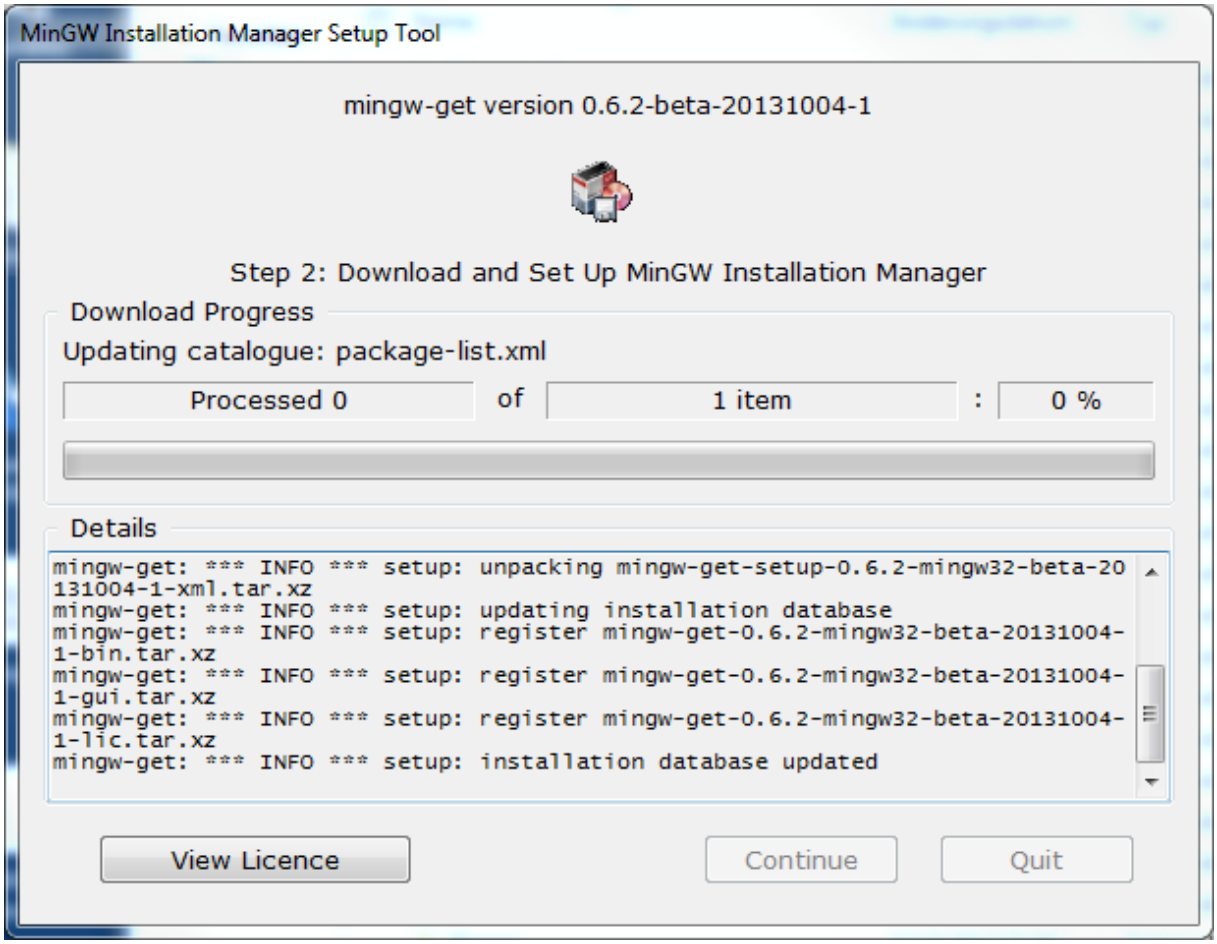
Die Lizenz wird gelesen und akzeptiert.



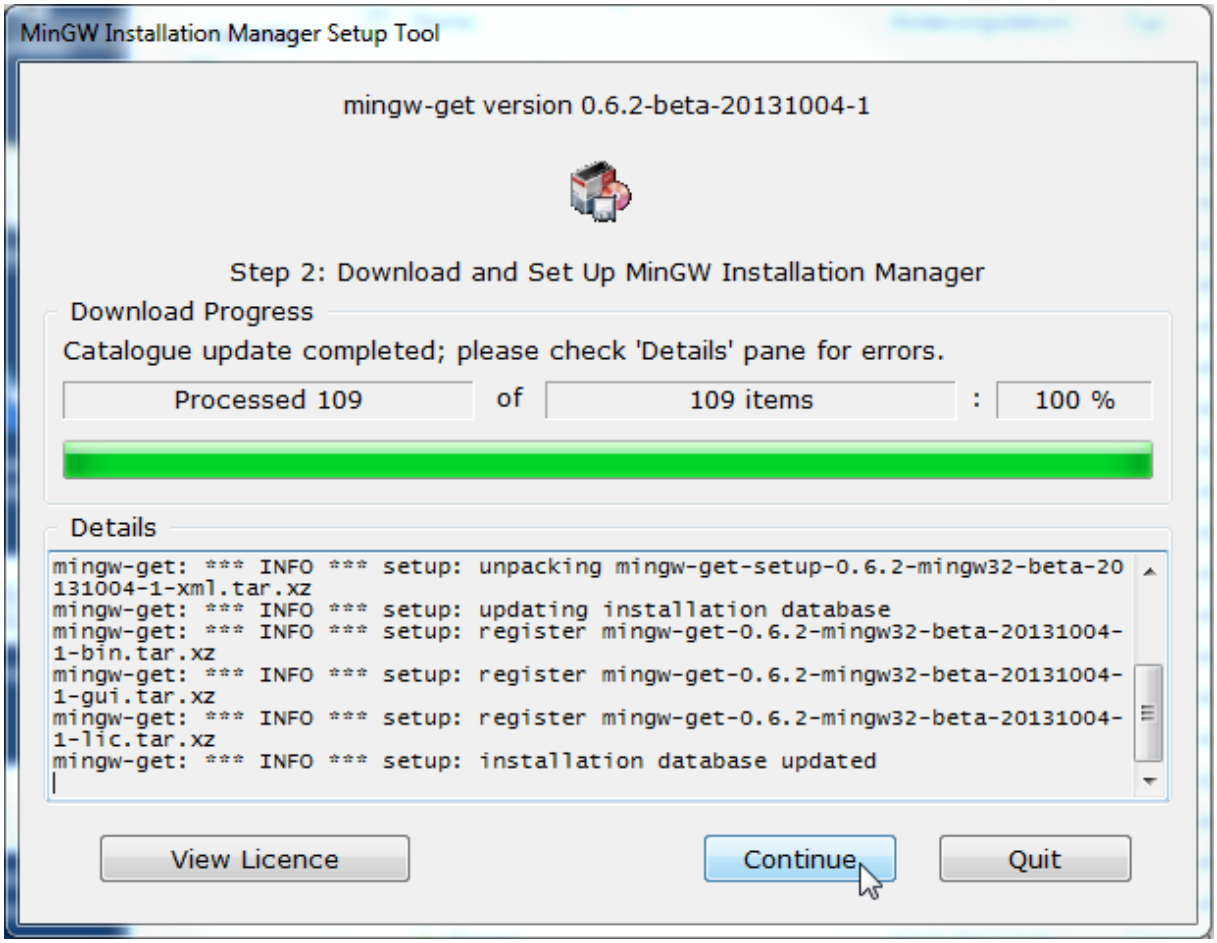
Der Installationspfad sollte möglichst so gelassen werden. Wichtig ist allerdings nur, dass keine Leerzeichen im Pfad enthalten sind. Die Einstellungen bleiben unverändert, es wird „Continue“ geklickt.



Es werden einige Informationen aus dem Internet geladen.



Zum Abschluss wird wieder „Continue“ geklickt.



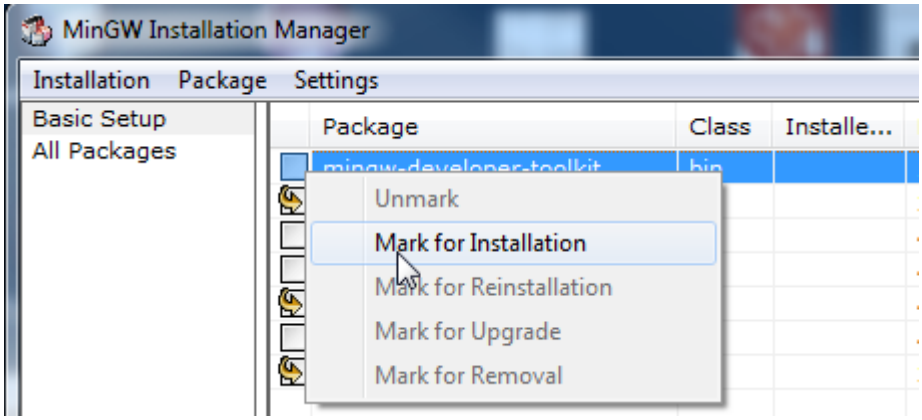
Es öffnet sich eine graphische Oberfläche, in der die zu installierenden Komponenten ausgewählt werden. Im konkreten Fall wird eine minimale Installation angestrebt, weshalb vereinfachend links auf „Basic Setup“ und rechts auf die gezeigten Pakete geklickt wird. Bei jedem Paket wird dann mit einem Rechtsklick „Mark for Installation“ gewählt. Einige der Markierungen werden automatisch gesetzt, so dass die Auswahl von

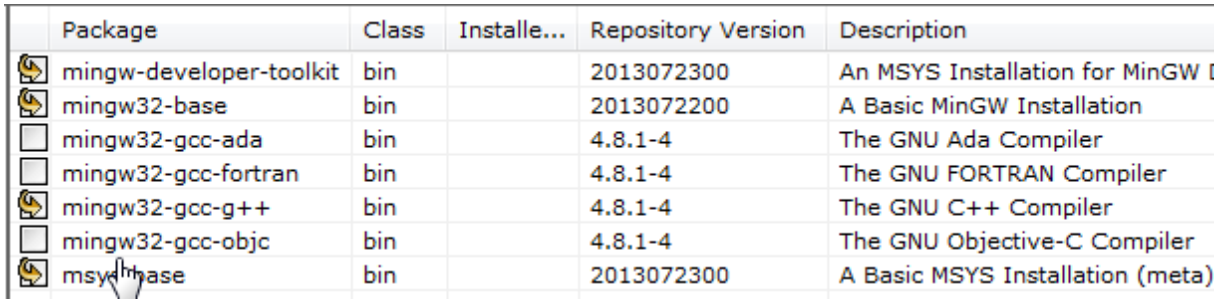
mingw-developer-toolkit

mingw32-base

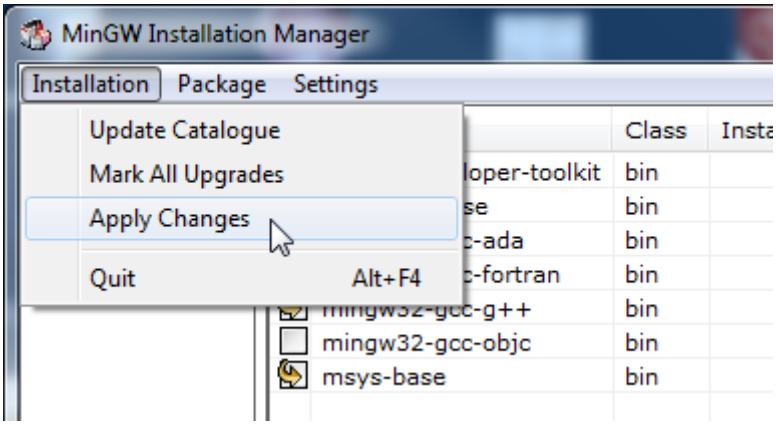
mingw32-gcc-g++

ausreicht. Die zusätzlich angebotene Software, enthält weitere Bibliotheken und aus UNIX bekannte Befehle.

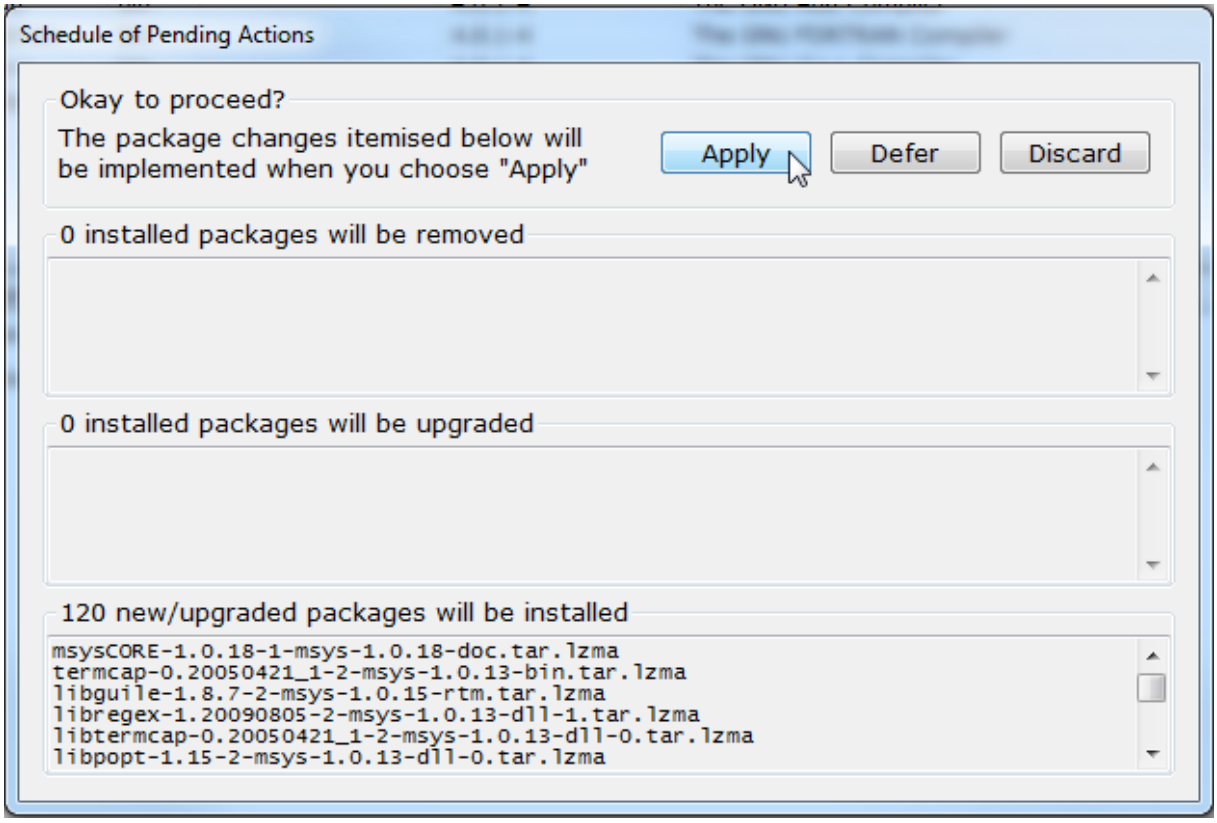




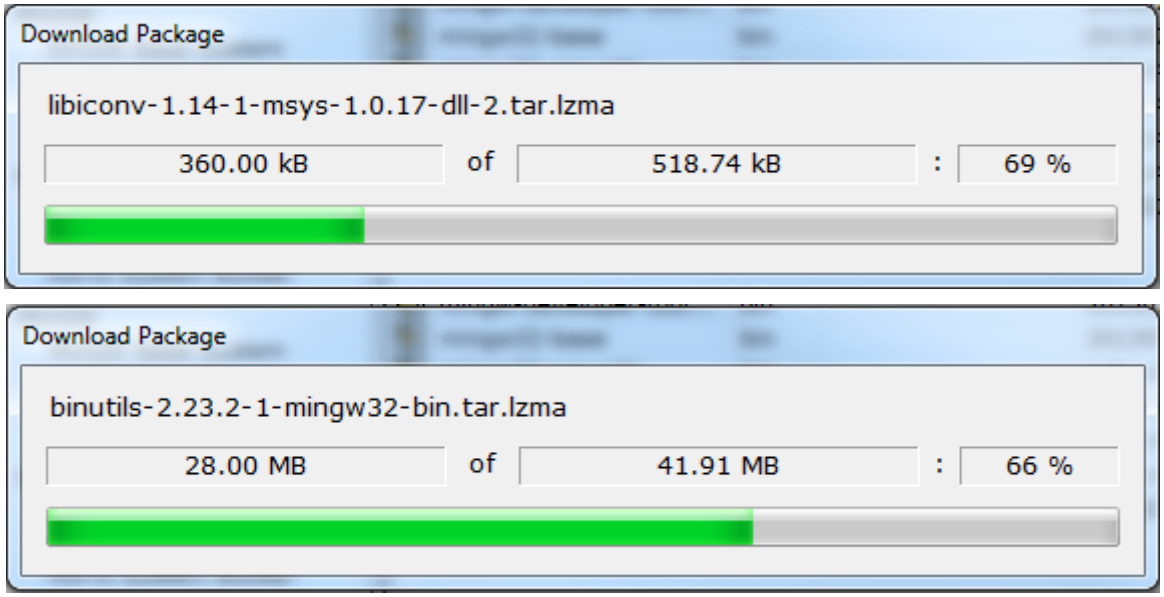
Das Programm kann später zur Veränderung der Installation genutzt werden. Zum Start der Installation wird „Installation > Apply Changes“ ausgewählt.

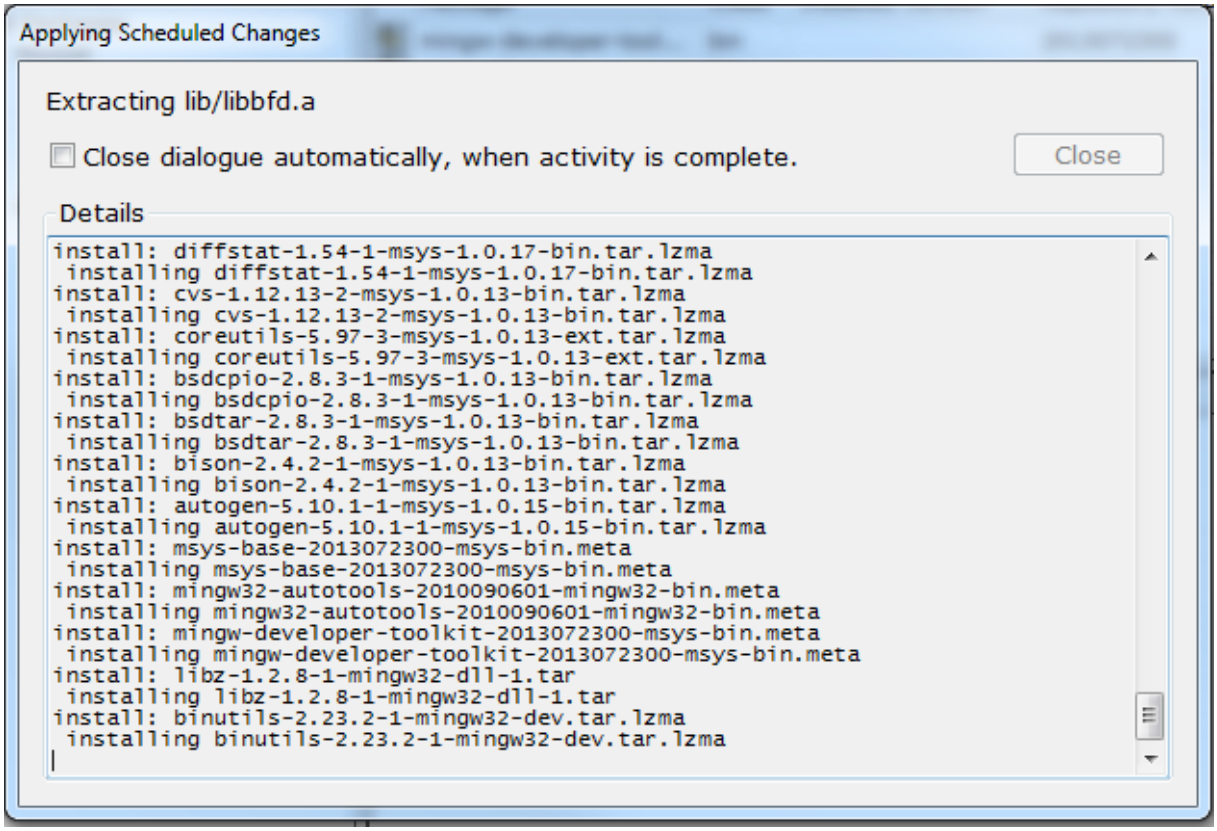


Es wird auf „Apply“ geklickt.

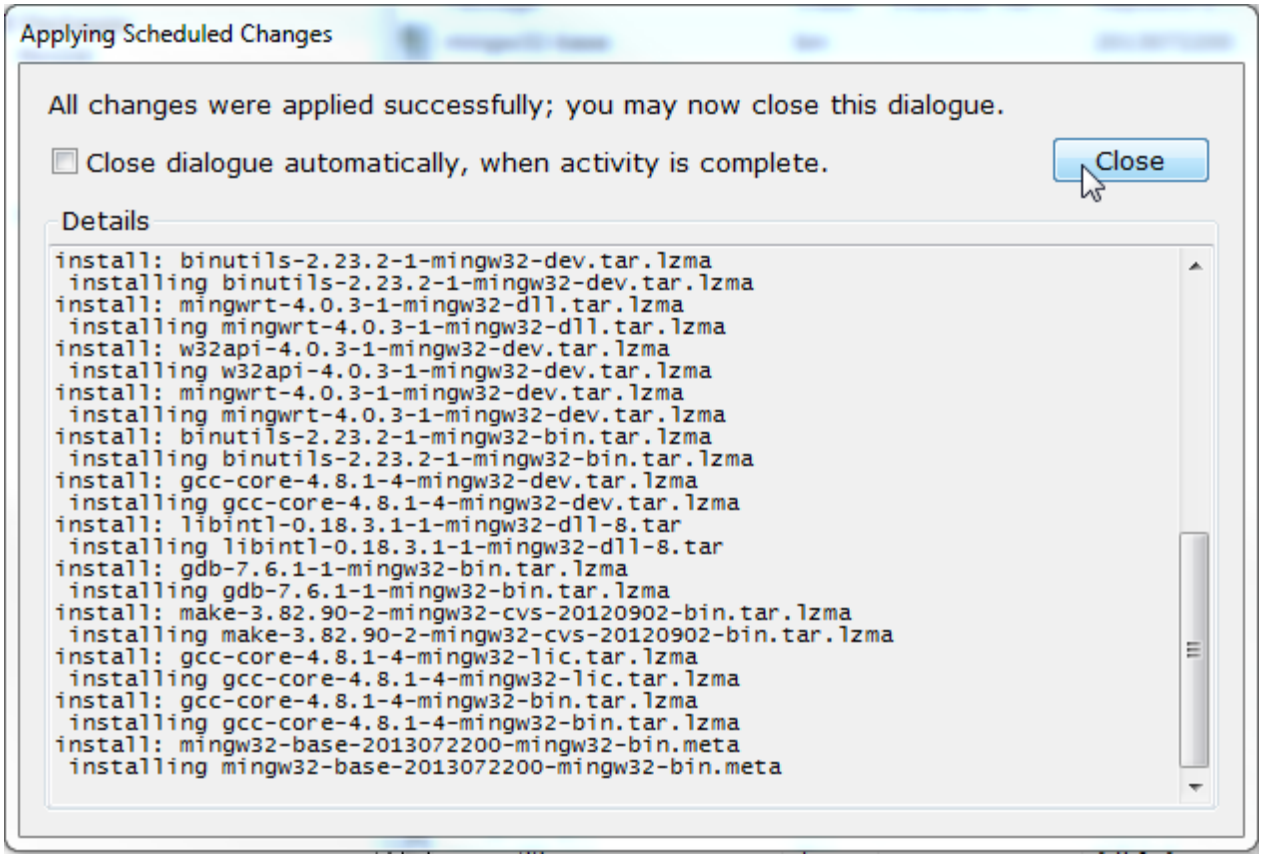


Die Pakete werden heruntergeladen und installiert, was einige Zeit dauert.

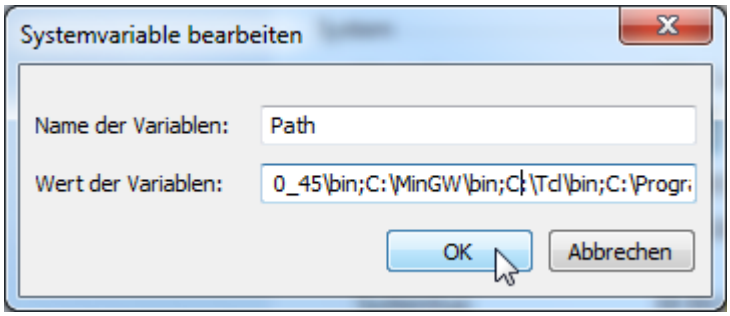




Am Ende wird „Close“ geklickt.

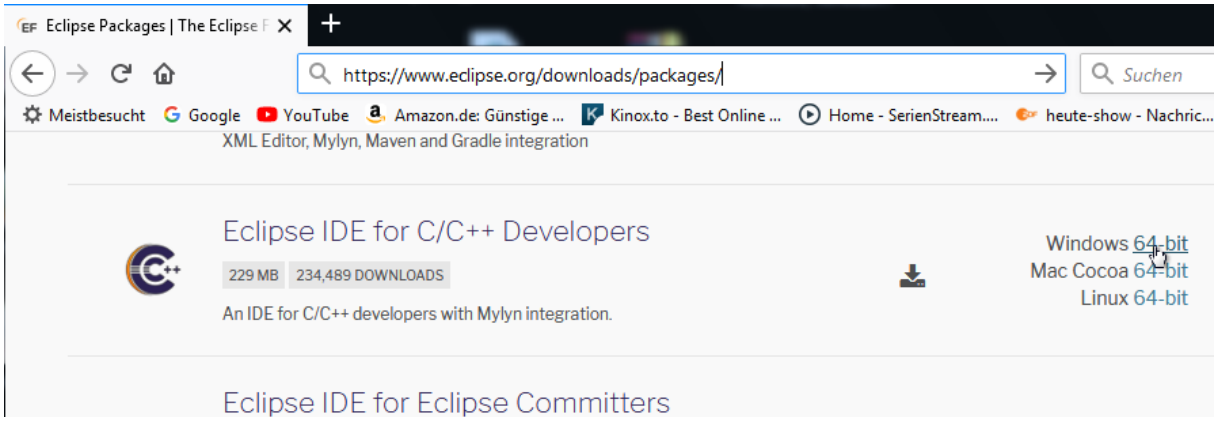


C:\MinGW\bin und C:\MinGW\MSYS\1.0\bin sind getrennt mit einem Semikolon in die PATH-Variable des Systems einzutragen. Die Arbeitsschritte sind für Java ab der Seite **Fehler! Textmarke nicht definiert.** beschrieben.



10.2 Installation von Eclipse für C++

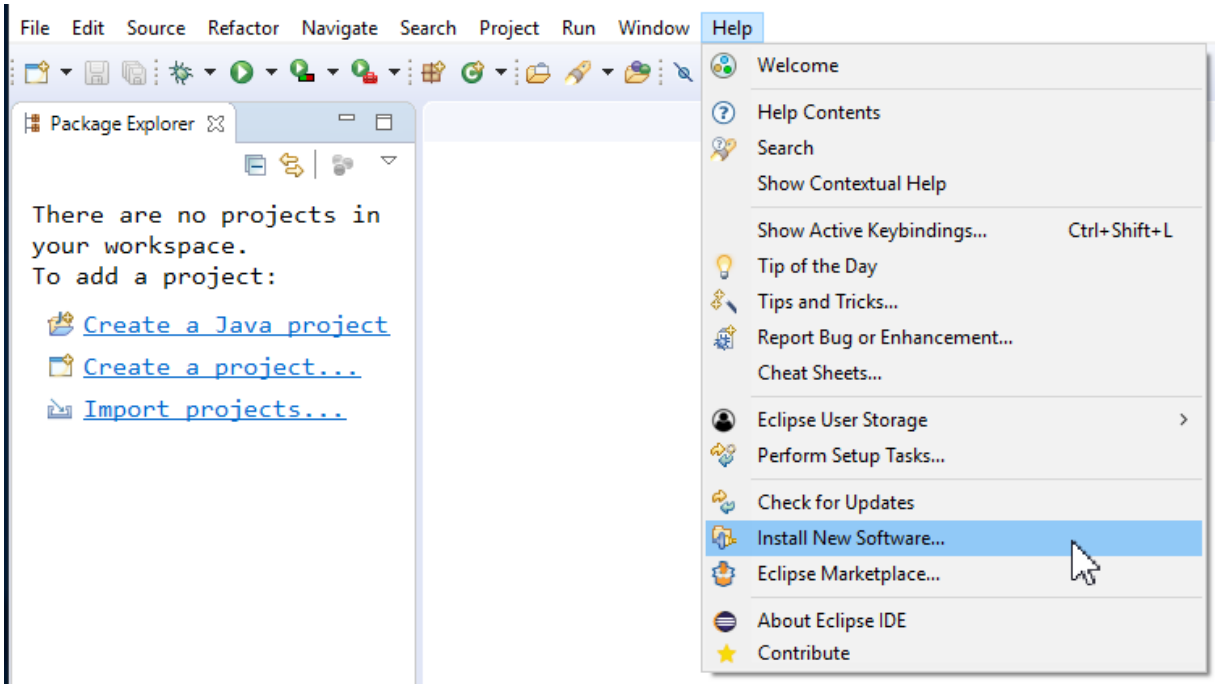
Da man nicht zu viele Eclipse-Plugins zusammen installieren soll, um die Schnelligkeit und Stabilität zu garantieren, sollte für eine professionelle Entwicklung ein Eclipse für C/C++ installiert (https://www.eclipse.org/downloads/packages/), also ausgepackt vorliegen. Alternativ kann die CDT-Erweiterung auch direkt für Eclipse installiert werden, wie es im folgenden Unterkapitel beschrieben wird.



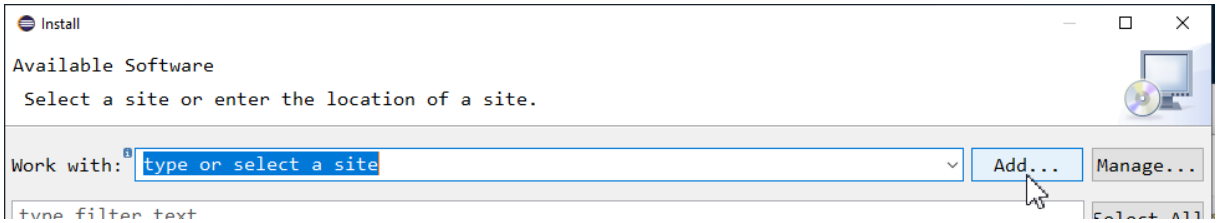
10.3 Installation der CDT-Erweiterung in Eclipse für C++

Generell können mehre Eclipse-Varianten zur Entwicklung mit verschiedenen Programmiersprachen in einem Eclipse existieren. Es ist aber zu beachten, dass Eclipse dadurch nicht nur beim Start langsamer und auch instabiler werden kann.

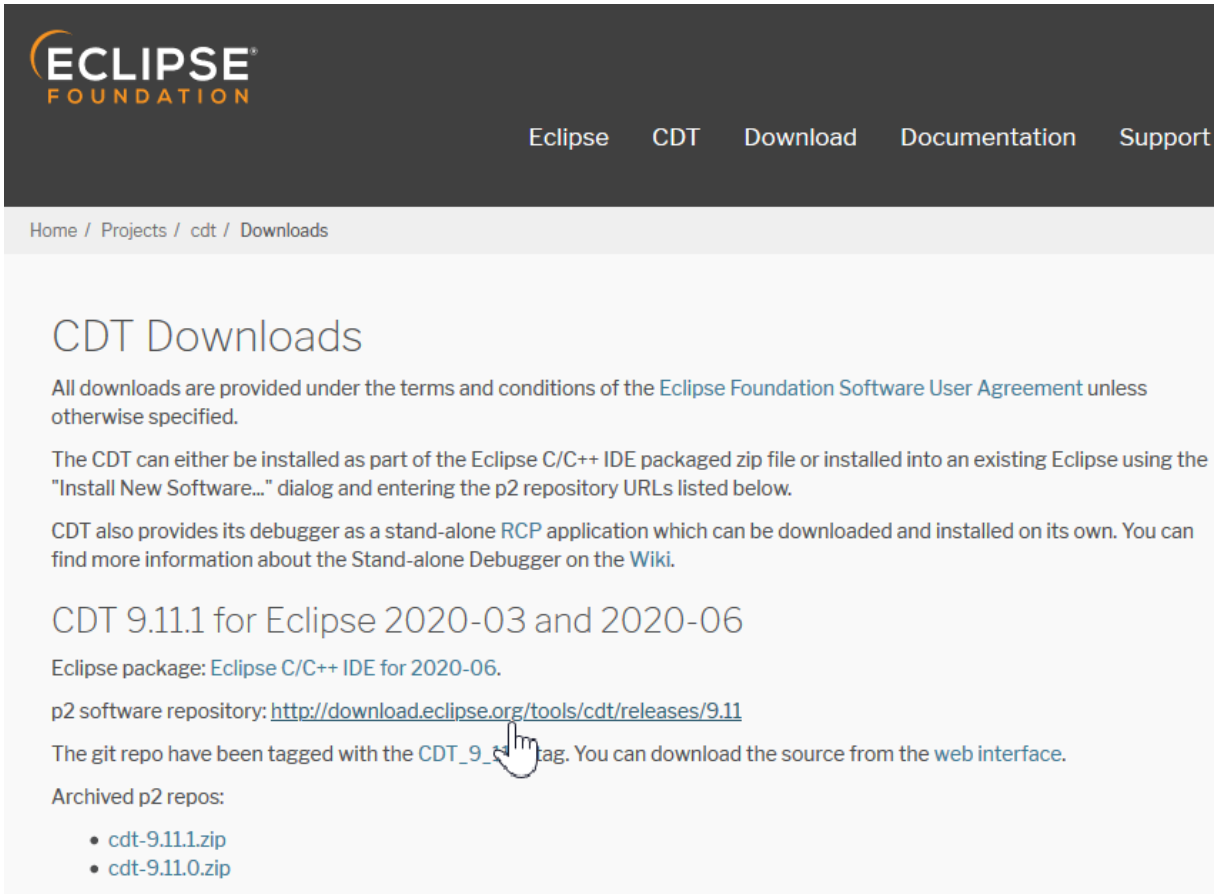
Es wird „Help > Install New Software…“ angewählt.



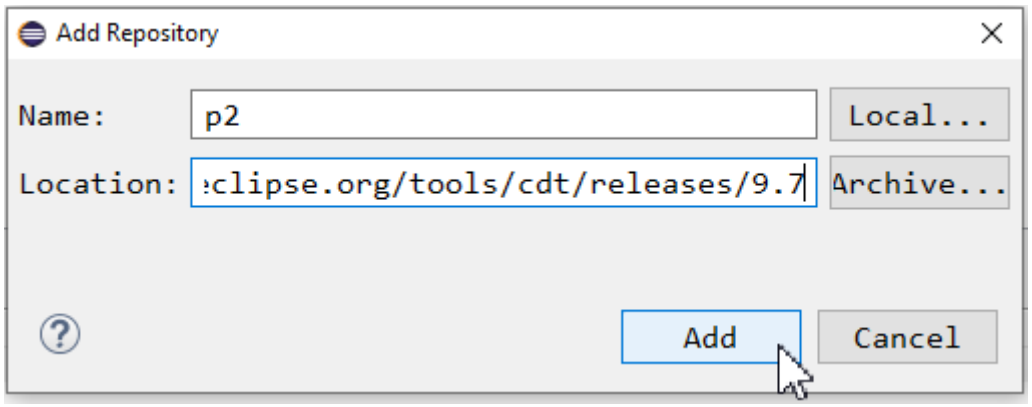
Rechts-oben wird auf „Add…“ geklickt.



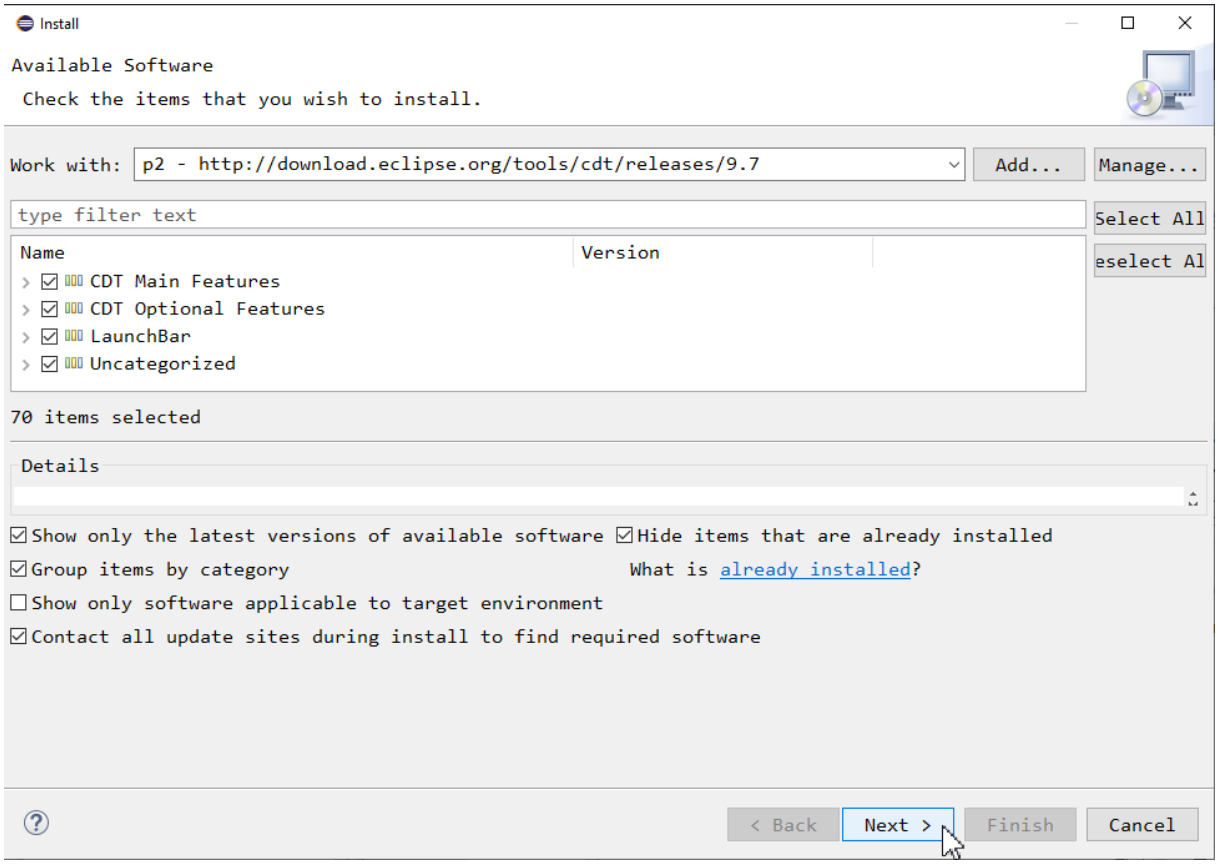
Der Name ist generell beliebig, für die C/C++-Erweiterung ist aber „p2“ üblich. Nun muss die passende „Location gefunden werden, die typischerweise auf der Webseite https://www.eclipse.org/cdt/downloads.php steht. Die zu wählende “Location“ steht hinter „p2 software repository“.



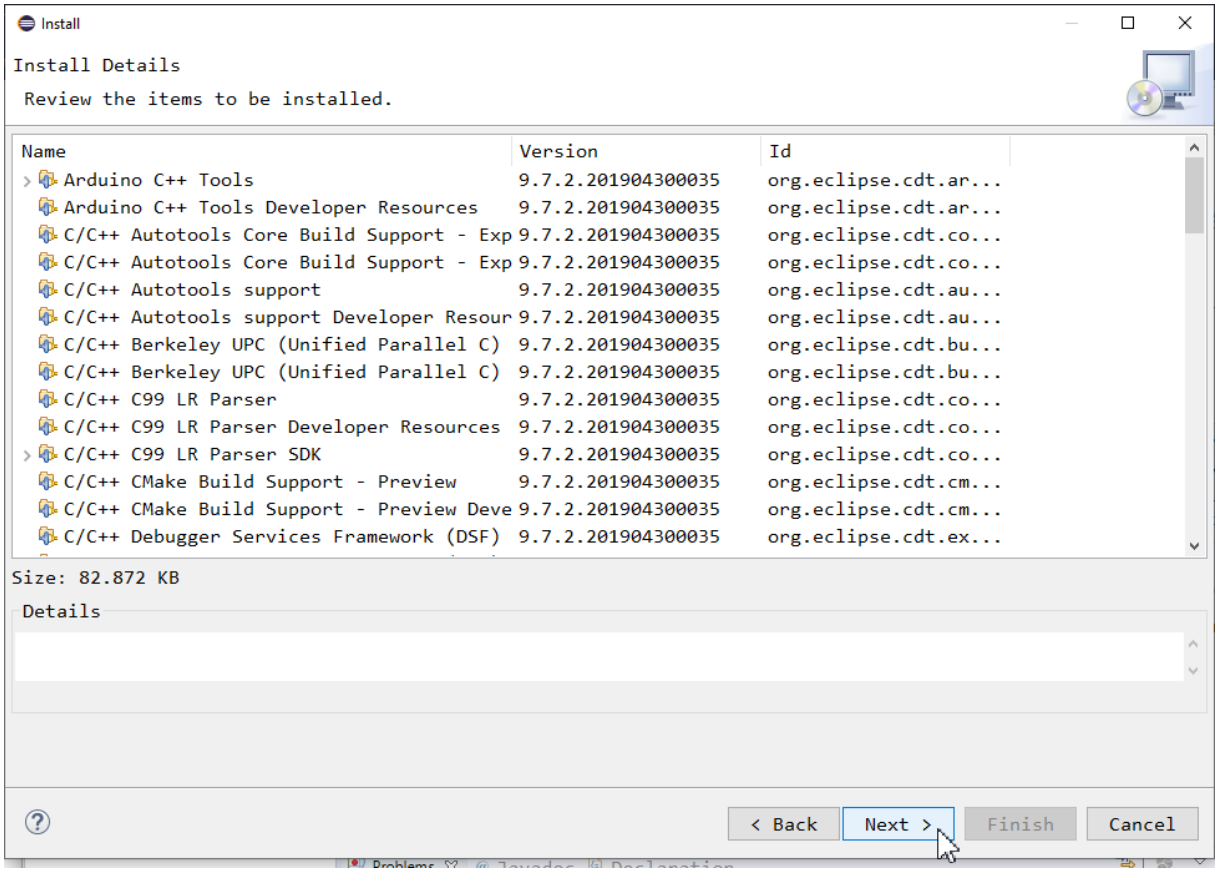
Als Location wird http://download.eclipse.org/tools/cdt/releases/9.11 angegeben. Die Aktion wird mit „Add“ abgeschlossen.



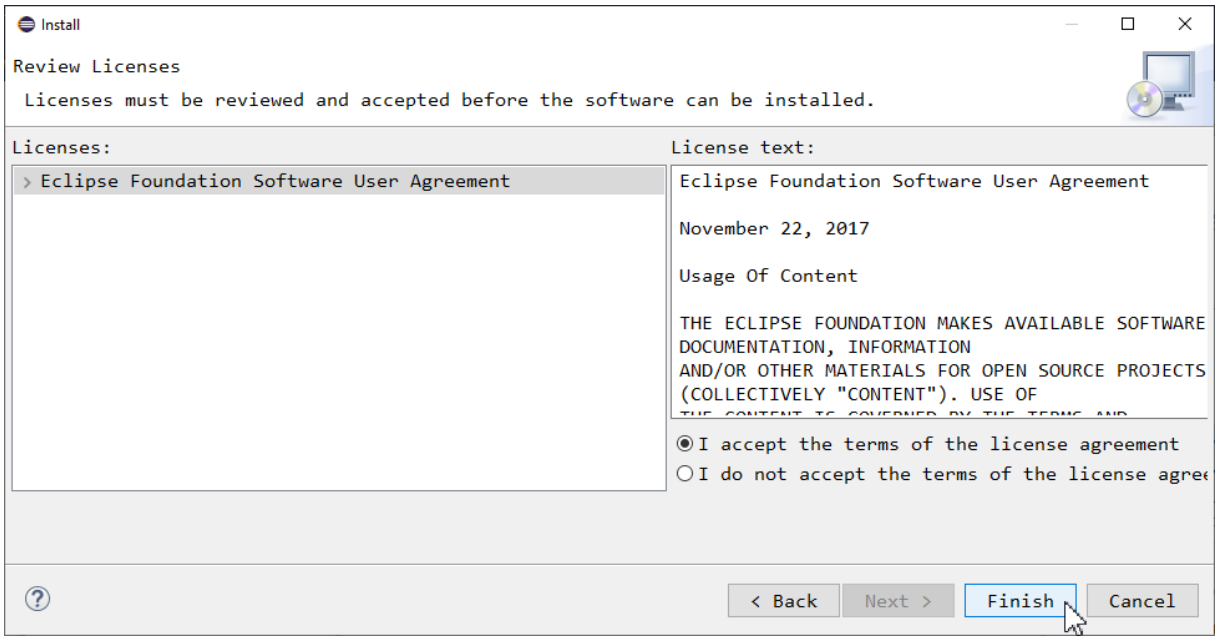
Es dauert etwas bis alle Software-Pakete gefunden werden. Vereinfachend werden alle Pakete mit „Select All“ ausgewählt und auf „Next >“ geklickt.



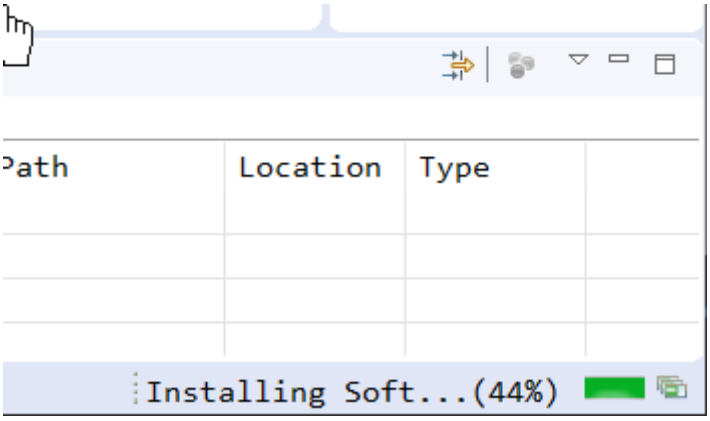
Es wird wieder „Next >“ geklickt.



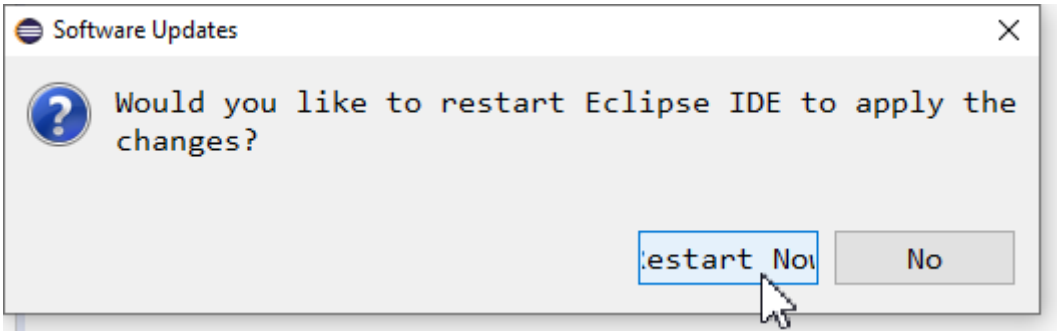
Die Lizenz-Informationen werden gelesen und mit einem Klick bei „I accept…“ angenommen. Dann wird „Finish“ geklickt.



Die Installation startet, was rechts-unten zu erkennen ist. Während der Installation darf nichts in Eclipse gemacht werden. Die Installation dauert.

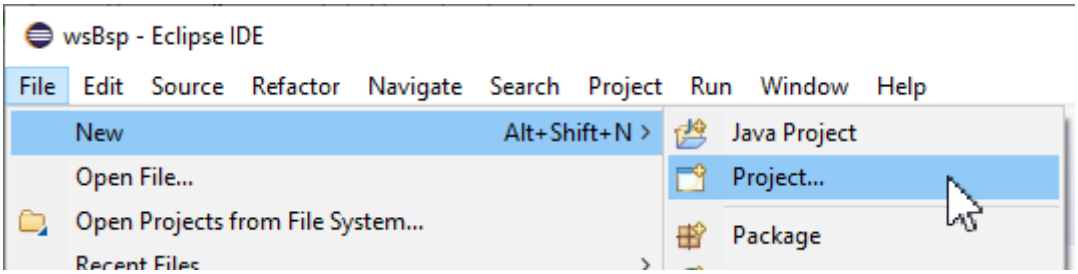


Zum Abschluss erfolgt eine Aufforderung zum Neustart, was dann ausgeführt wird.

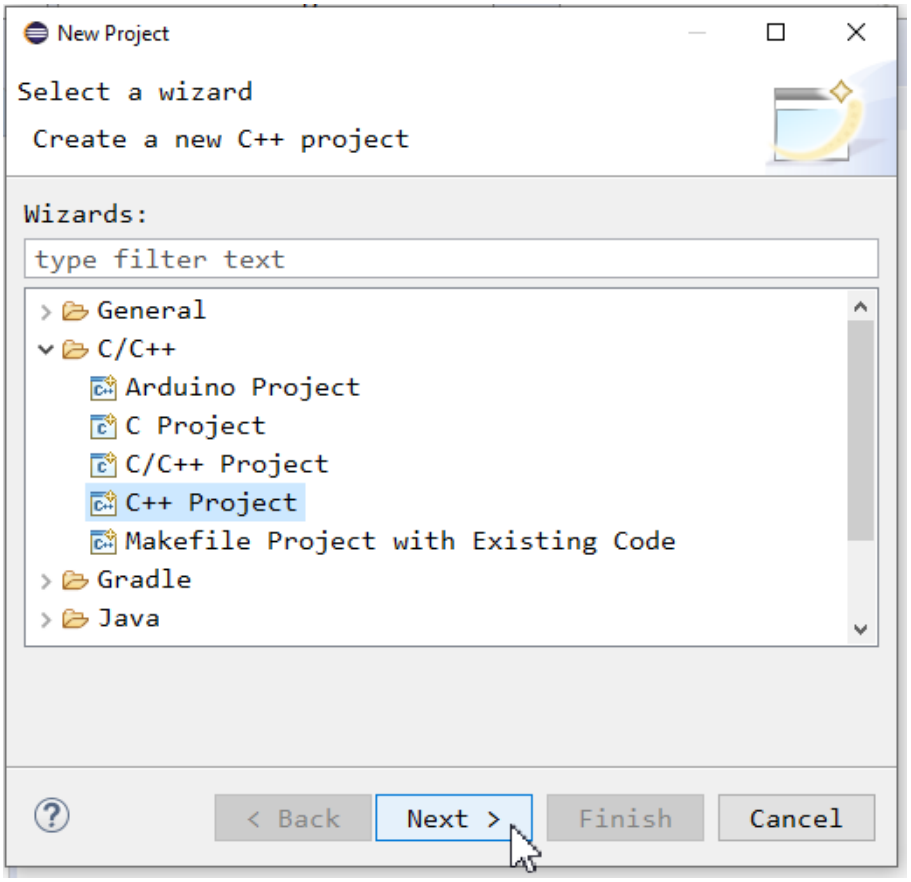


10.4 Erstellung eines C++-Projekts

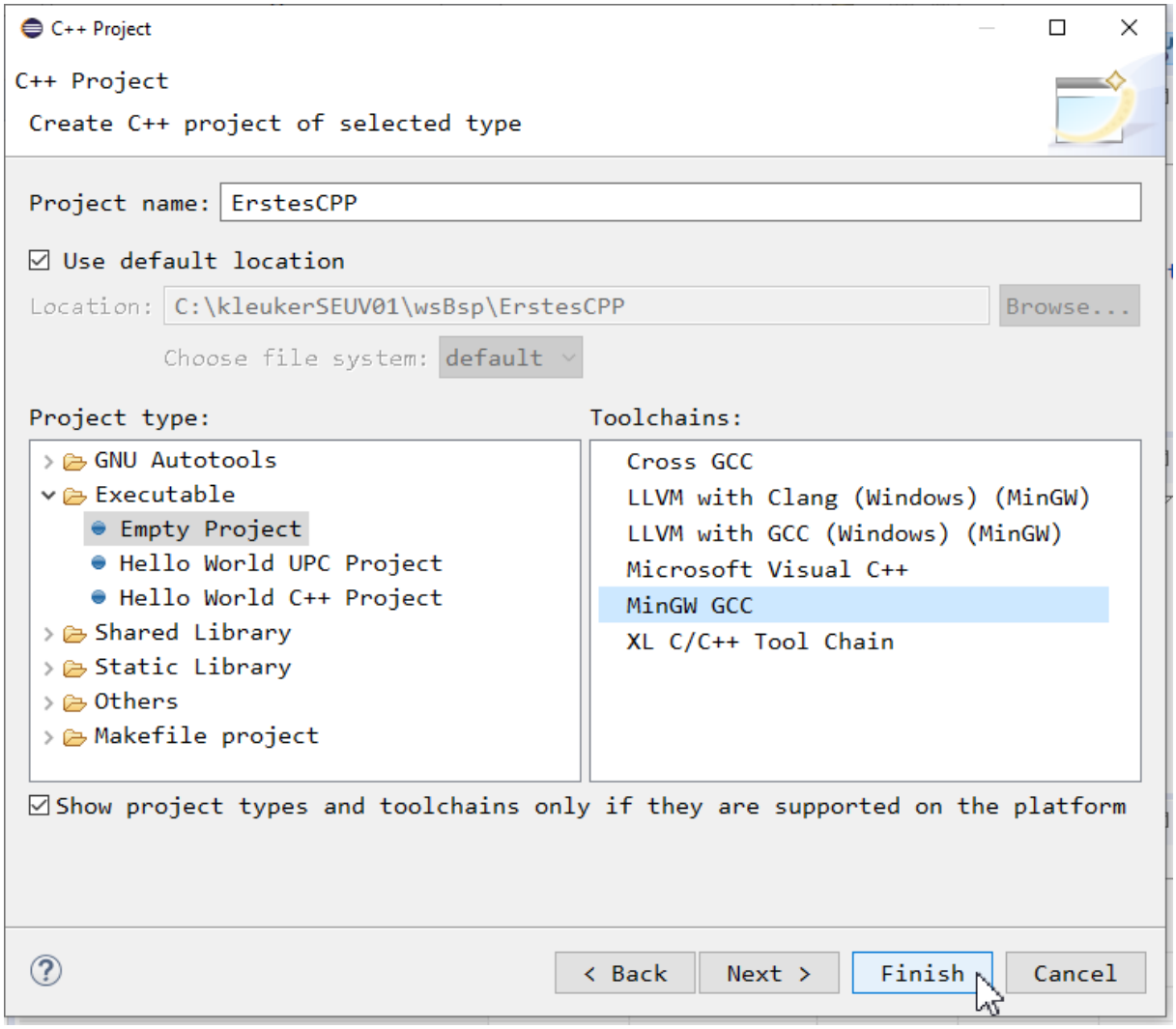
Zunächst wird ein neues Projekt über „File > Project…“ angelegt.



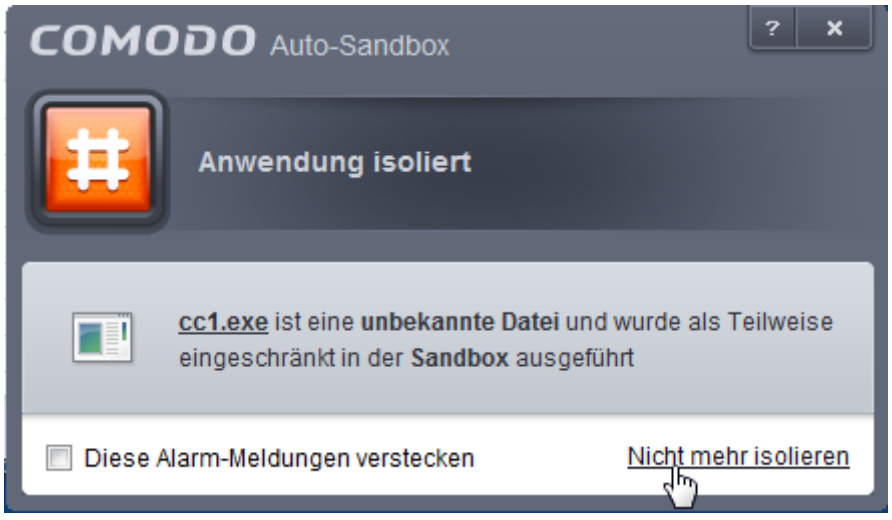
Der Eintrag C++ Project wird gesucht und „Next>“ gedrückt.

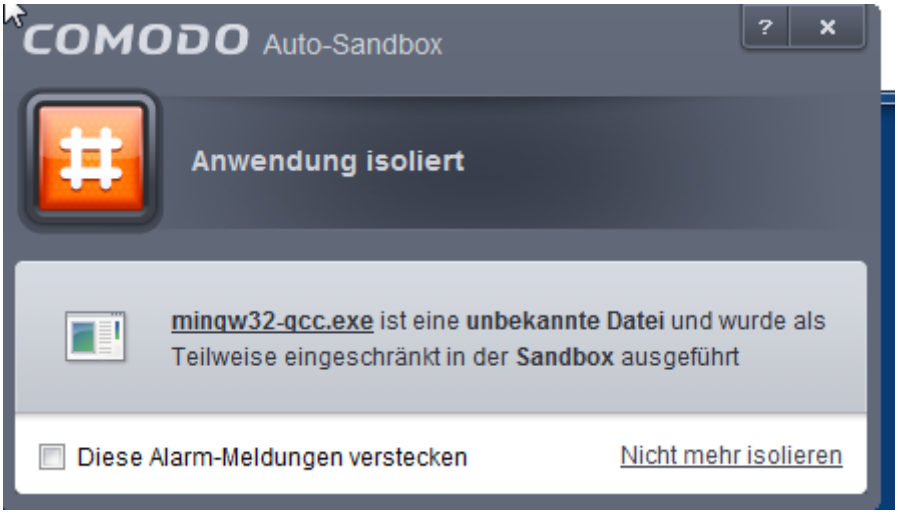


Neben dem Projektnamen wird links „Empty Project“ und rechts „MinGW GCC“ ausgewählt und „Finish“ geklickt.

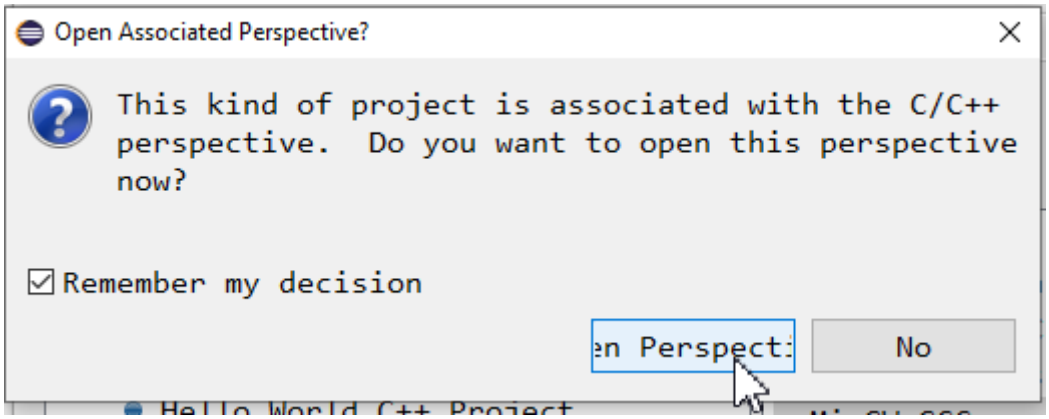


Eventuell meldet sich ein Sicherheitsprogramm, bei dem Rechte eingeräumt werden müssen.

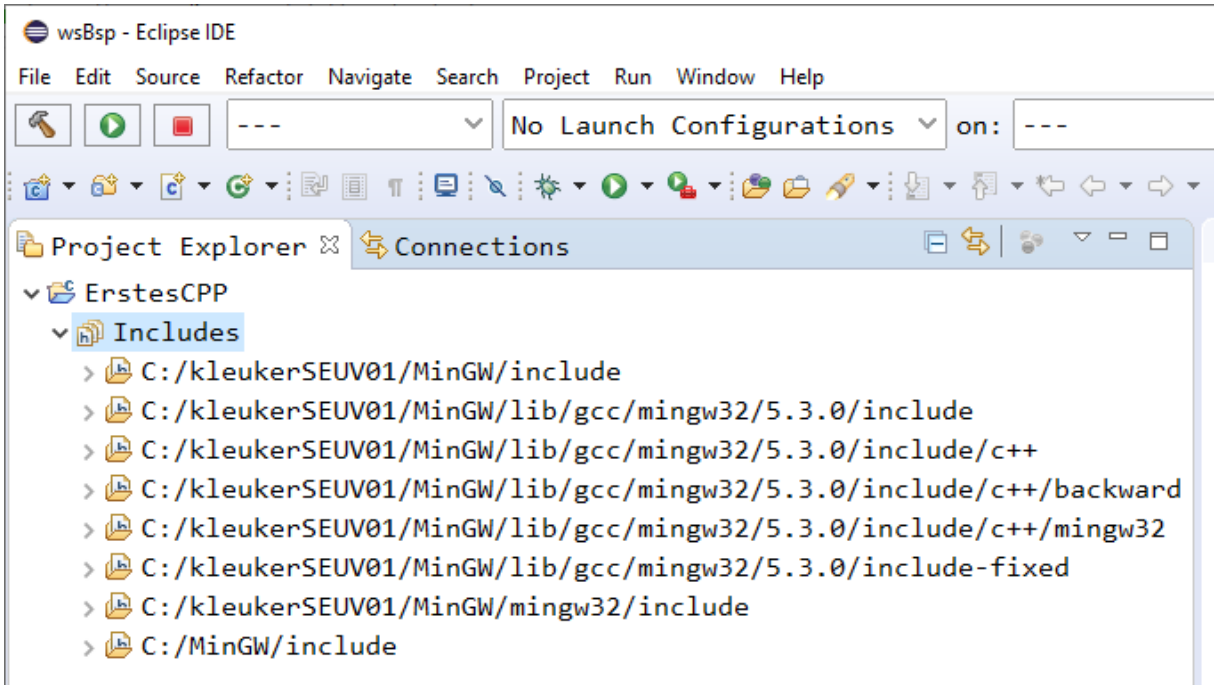




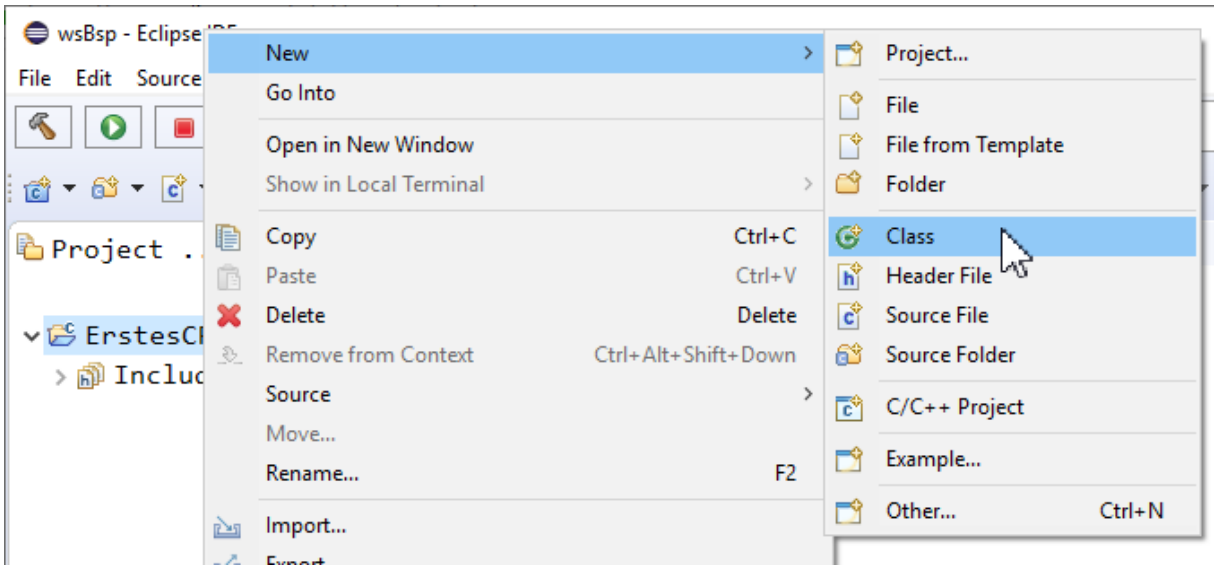
Falls die folgende Meldung kommt, ist diese zu bestätigen und der Haken links unten kann gesetzt werden.



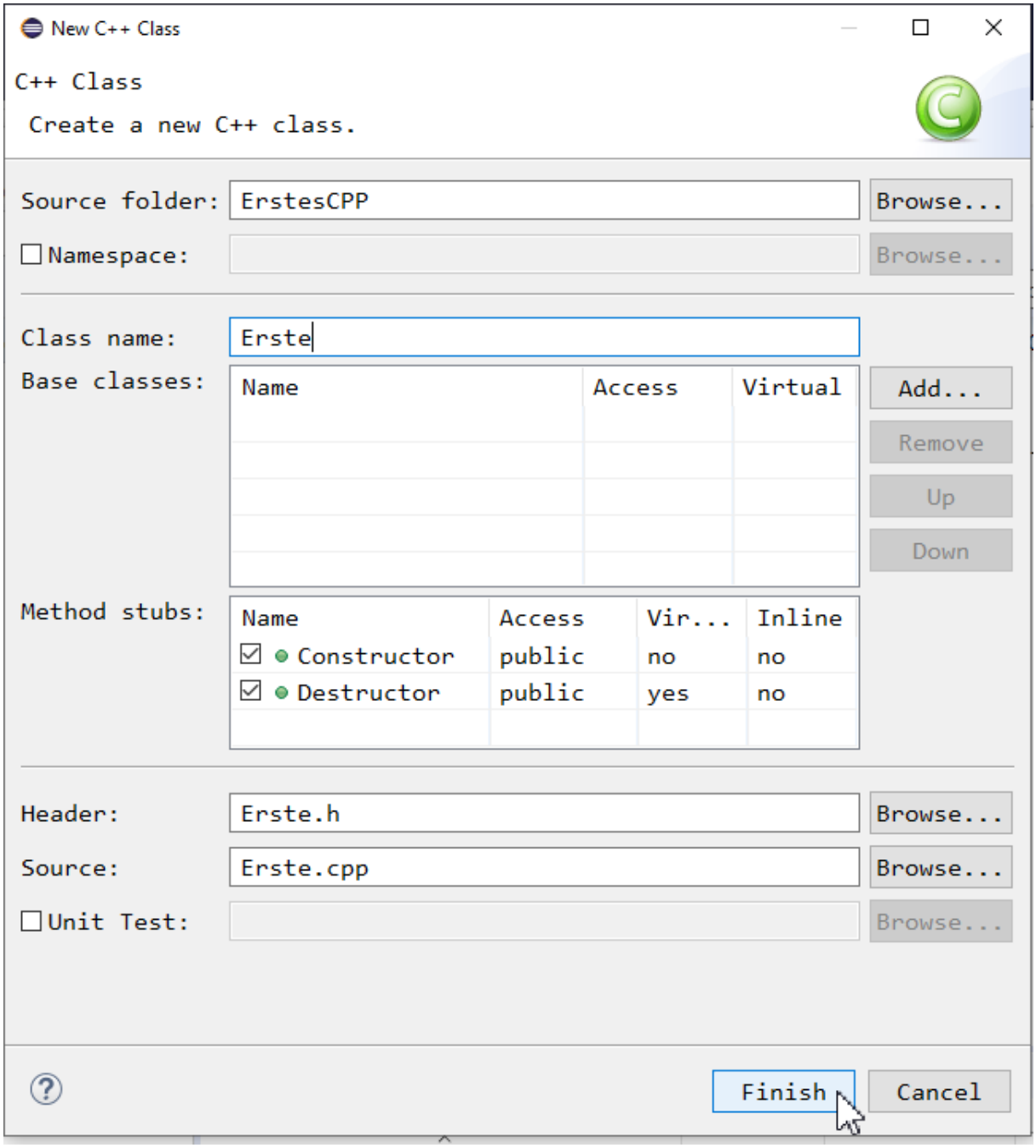
Man erkennt, dass ein Projekt angelegt wird, das einen speziellen Ordner includes enthält. Wird dieser geöffnet, sieht man, welche Libraries zur Verfügung stehen.



Eine Klasse kann z. B. mit einem Rechtsklick auf den Projektnamen über „New > Class“ 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). Es ist sinnvoll zuerst die Header-Datei zu speichern, da dann in der Realisierungsdatei erkannt wird, welche Kon- sowie Destruktoren und Methoden umzusetzen sind.

```
#ifndef ERSTE_H_
#define ERSTE_H_

class Erste {
```

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

#endif /* ERSTE_H_ */
```

und

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

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

Erste::~Erste() {}

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

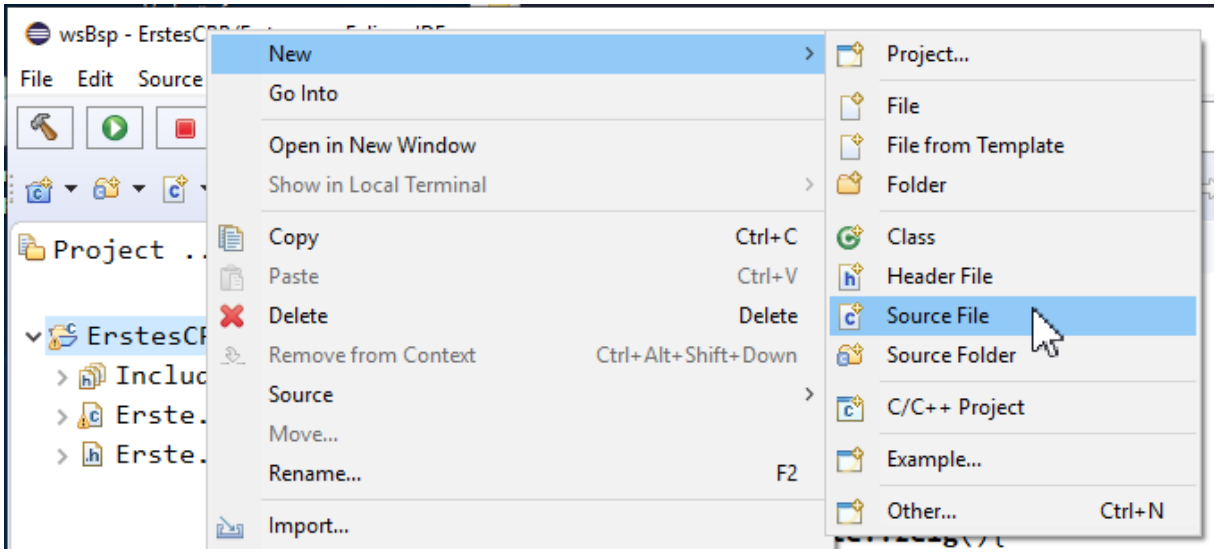
Erste.cpp | Erste.h

```
1  #ifndef ERSTE_H_
2  #define ERSTE_H_
3
4  class Erste {
5
6  private:
7      int x;
8  public:
9      Erste(int x);
10     virtual ~Erste();
11     void zeig();
12 };
13
14 #endif /* ERSTE_H_ */
15
```

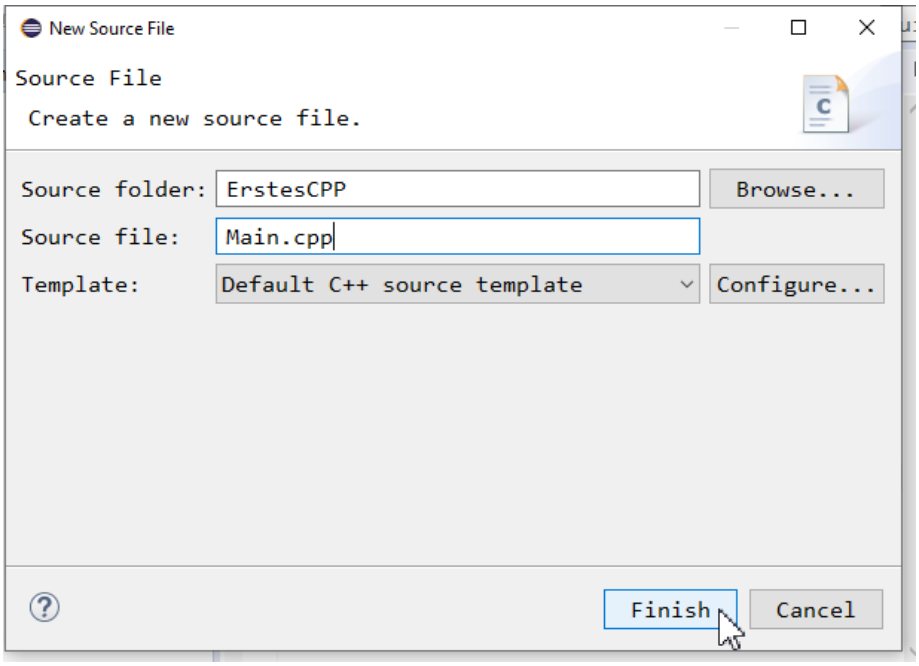
Erste.cpp | Erste.h

```
1  #include "Erste.h"
2  #include <iostream>
3
4  Erste::Erste(int x): x(x) {}
5
6  Erste::~Erste() {}
7
8  void Erste::zeig(){
9      std::cout<< "Sinn=" << x << "\n";
10 }
11
```

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



Es muss nur der vollständige Dateiname angegeben werden.



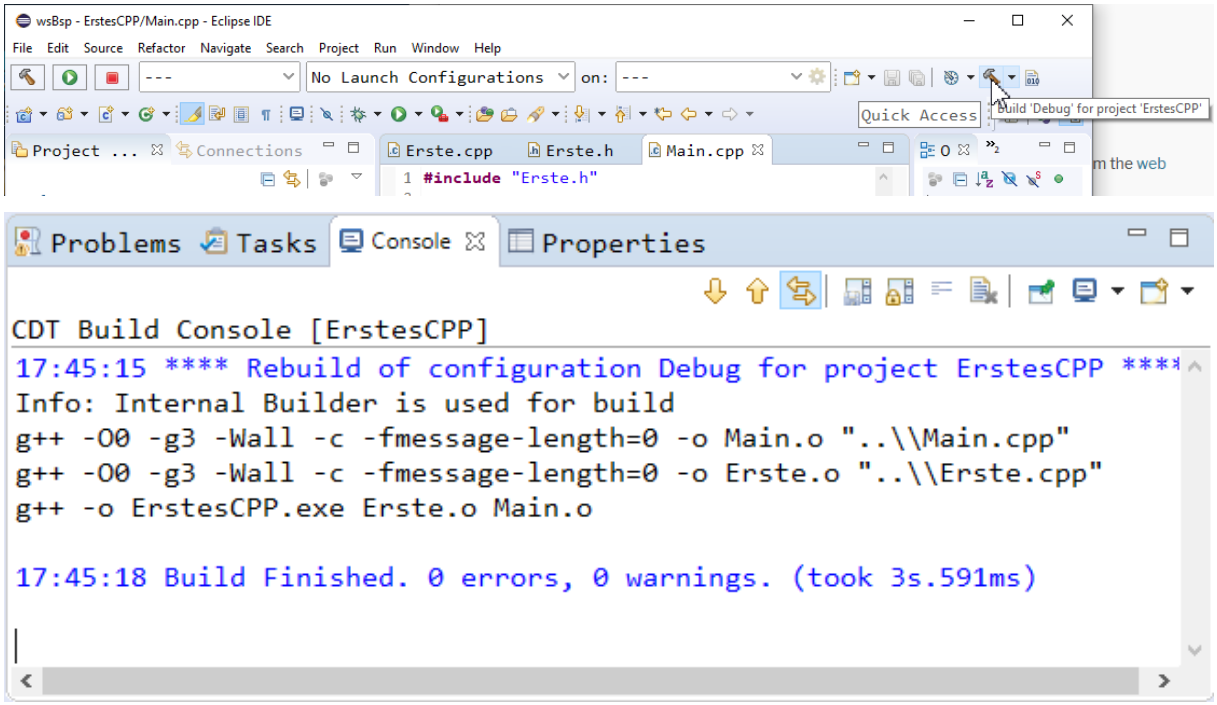
Die Beispieldatei sieht wie folgt aus. Man beachte, dass am Ende der Datei nicht mehr nachzuvollziehenden Gründen immer eine Leerzeile stehen muss.

```
#include "Erste.h"

int main(){
	Erste e(42);
	e.zeig();
	return 0;

}
```

Erste.cpp | Erste.h | Main.cpp

```
1  #include "Erste.h"
2
3  int main(){
4      Erste e(42);
5      e.zeig();
6      return 0;
7  }
8
```

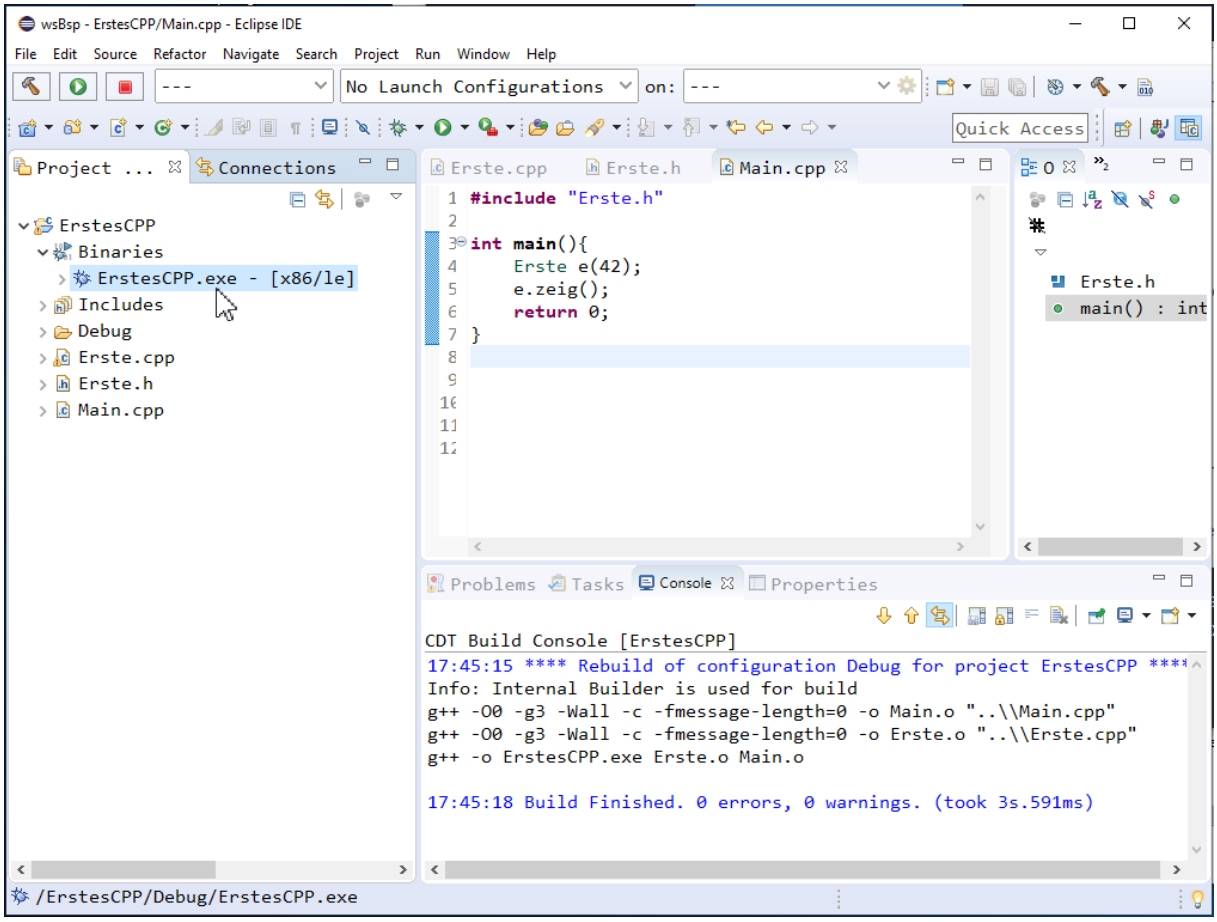
Bis jetzt wurden die Dateien nur angelegt und nicht kompiliert, was theoretisch auch eher möglich gewesen wäre. Zum ersten Kompilieren kann z. B. der mit einem Hammer markierte Button genutzt werden. Wichtig ist, dass alle Dateien *vorher gespeichert wurden*! Die Compilermeldungen werden dann unten ausgegeben.



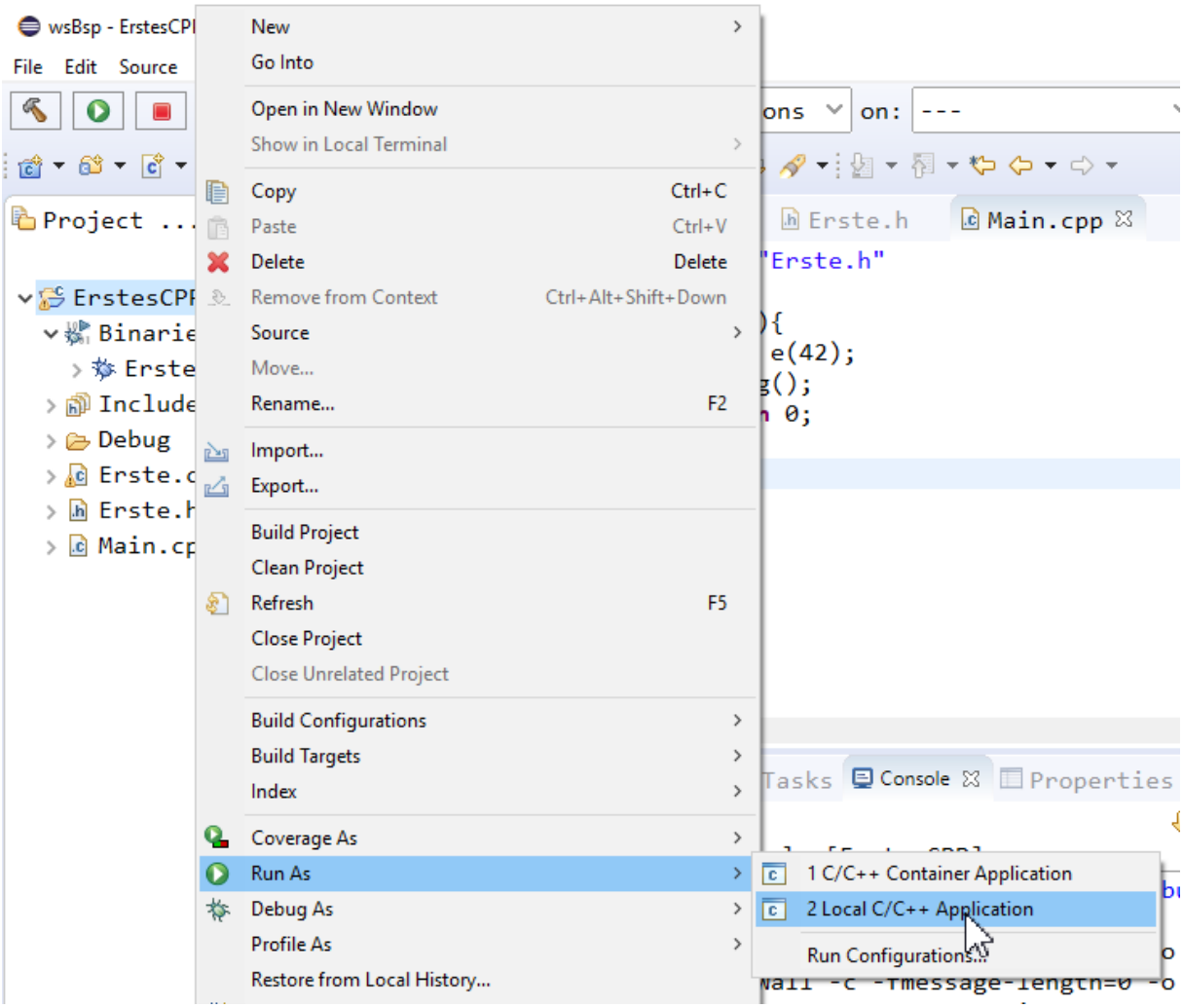

```
CDT Build Console [ErstesCPP]
18:46:09 **** Incremental Build of configuration Debug for project ErstesCPP ****
Info: Internal Builder is used for build
g++ -O0 -g3 -Wall -c -fmessage-length=0 -o Erste.o "..\\Erste.cpp"
g++ -o ErstesCPP.exe Main.o Erste.o

18:46:11 Build Finished (took 2s.122ms)
```

Eventuell müssen wieder Sicherheitseinstellungen beachtet werden.

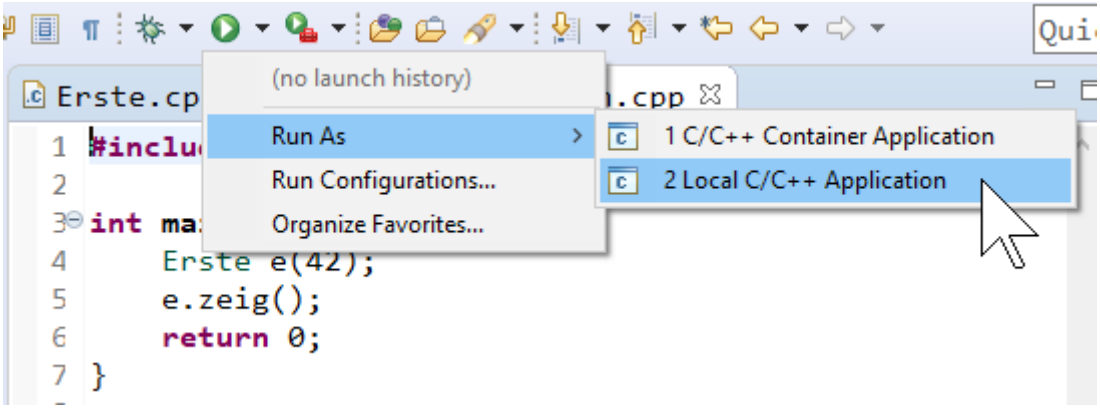
Nach der erfolgreichen Kompilierung ist ein Ordner Binaries entstanden.



Zur Programmausführung gibt es die Möglichkeit über einen Rechtsklick auf dem Projekt „Run As > Local C/C++ Application“ auszuwählen.

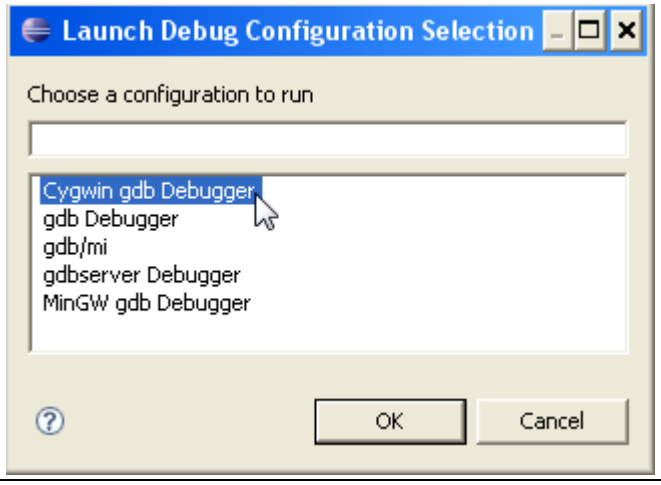


Alternativ ist ein Start über einen Klick auf den kleinen Pfeil neben dem grünen Pfeil und der gleichen Auswahl möglich.

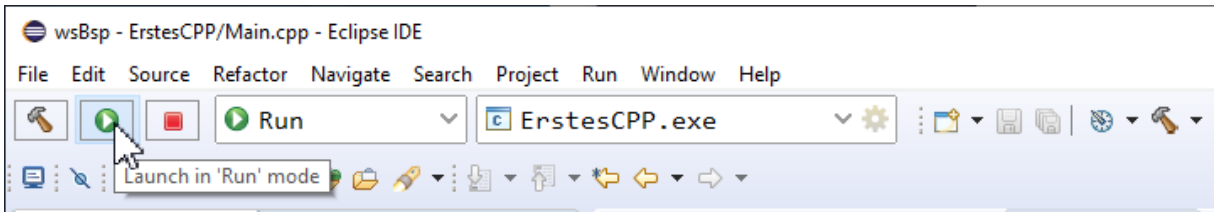


nur für ältere Eclipse-Varianten:

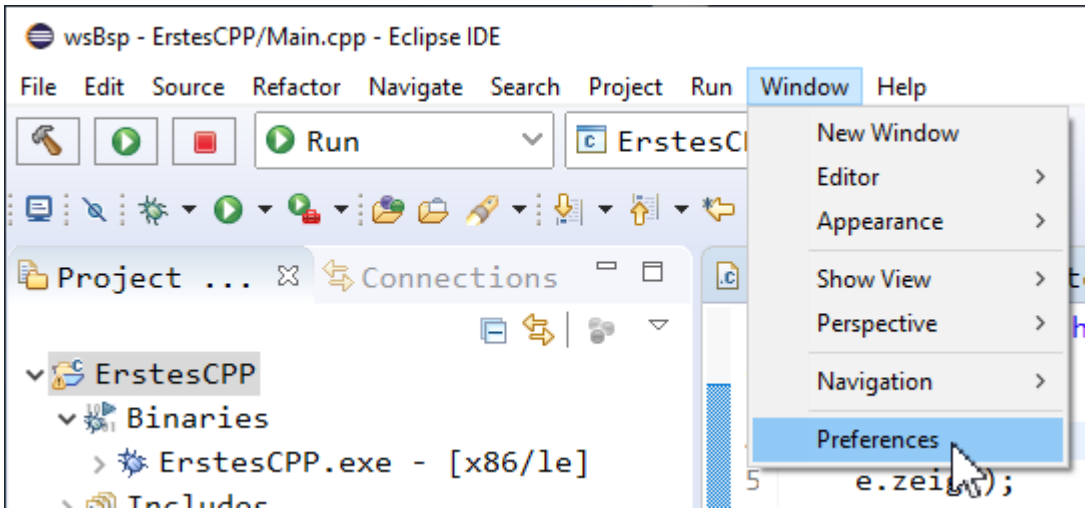
Da Projekte standardmäßig im Debug-Modus laufen, muss jetzt einmal ein Debugger für das Projekt gewählt werden. Da Cygwin genutzt wird, ist natürlich auch dieser Debugger zu nutzen.

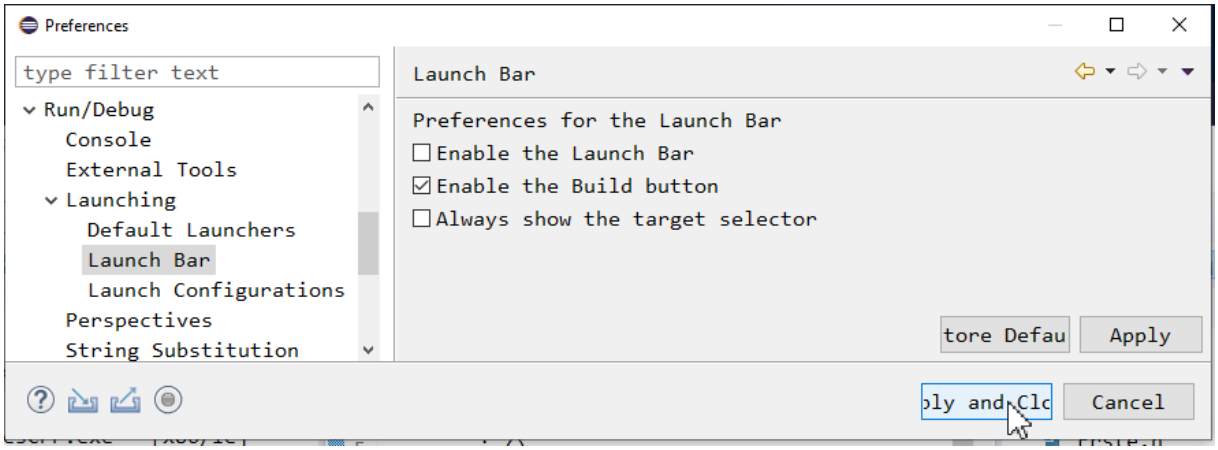


In neueren Eclipse-Versionen steht ein Launcher in der oberen Zeile zur Verfügung, über den ein Build und eine Ausführung möglich ist. Dies wird aber erst angeboten, wenn das Programm einmal über eine der vorher angegebenen Möglichkeiten gestartet wurde.



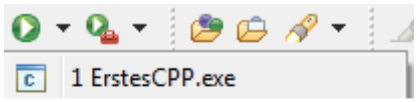
Falls der Launcher wieder entfernt werden soll, wird „Window > Preferences > Run/Debug > Launching > Launch Bar“ ausgewählt.

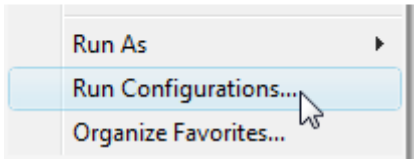




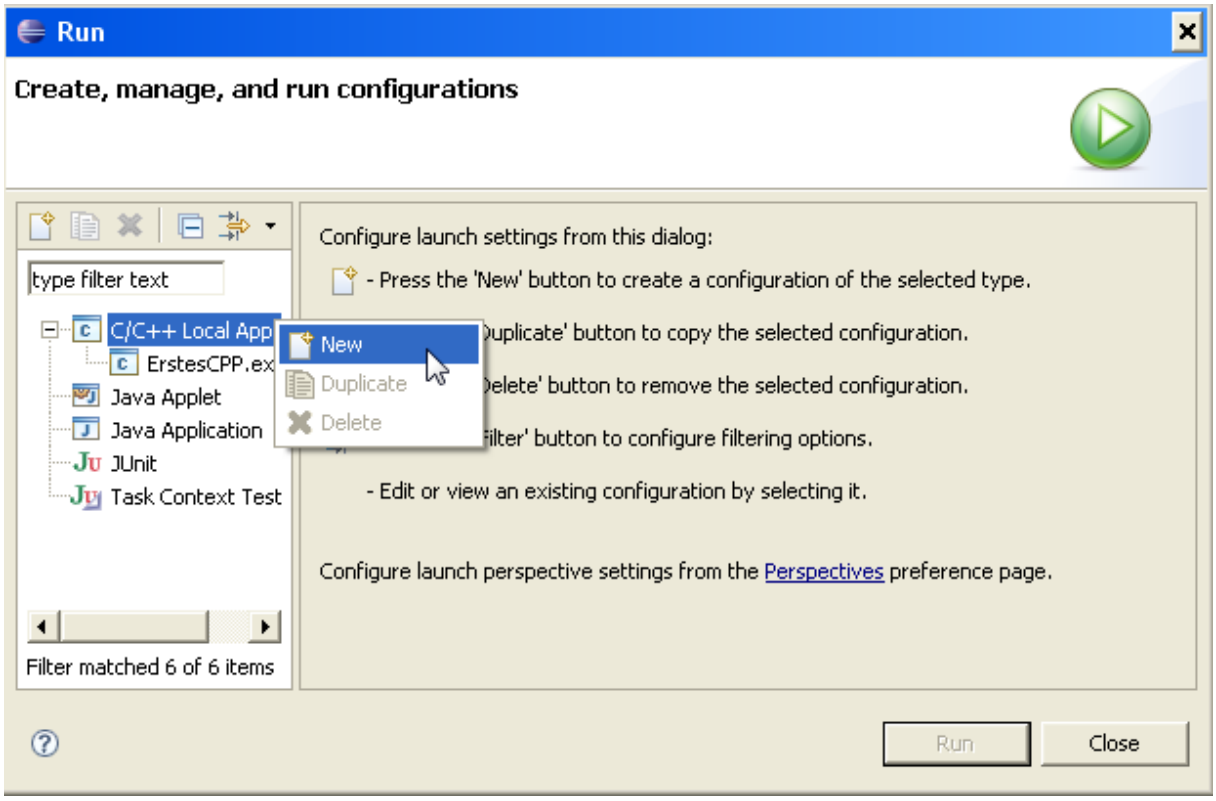
Zur erneuten Ausführung reicht es dann, direkt auf den Ausführungspfeil zu drücken.

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

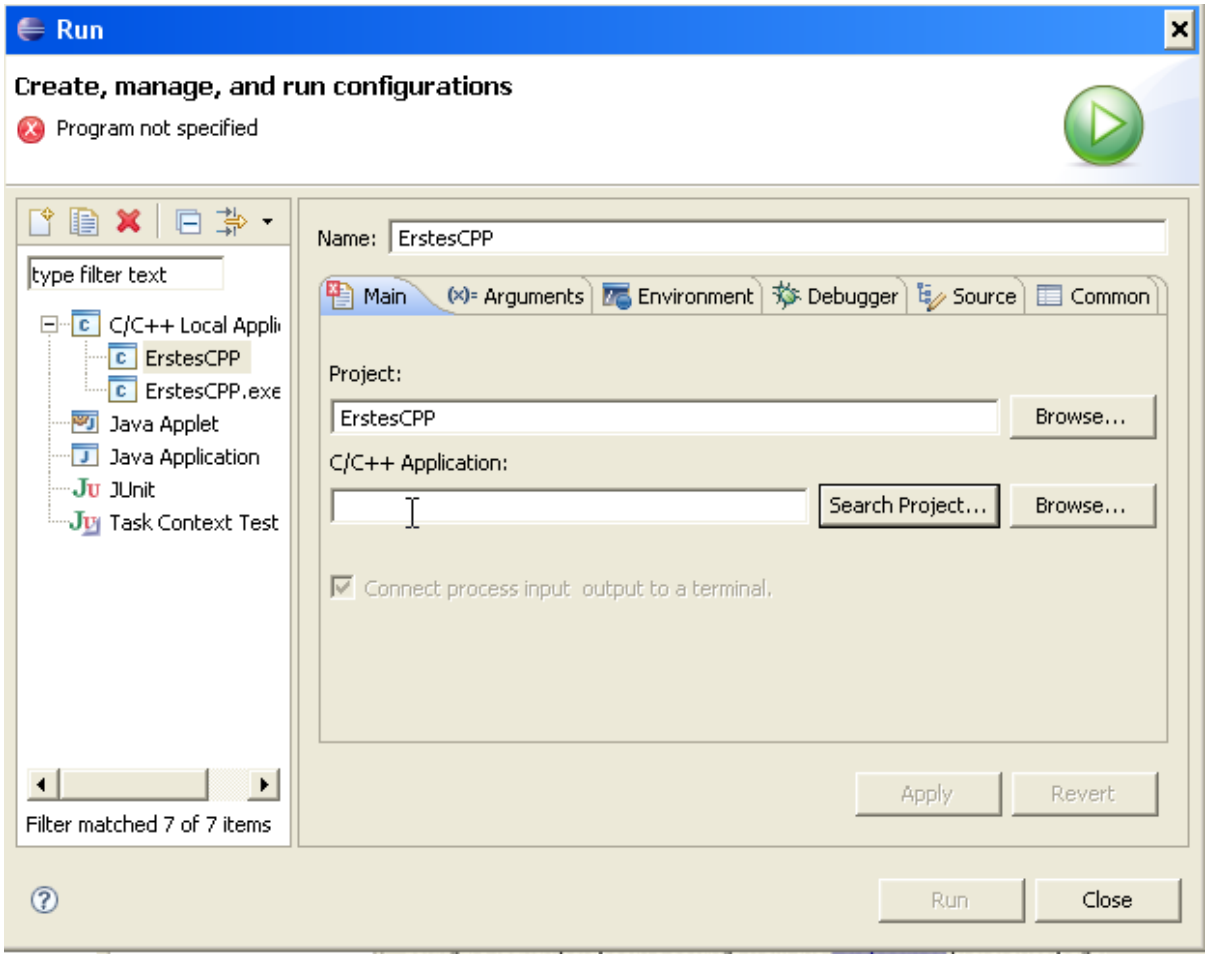




Eine neue Konfiguration wird dann durch einen Rechtsklick auf „C/C++ Local Application“ und „New“ erzeugt.

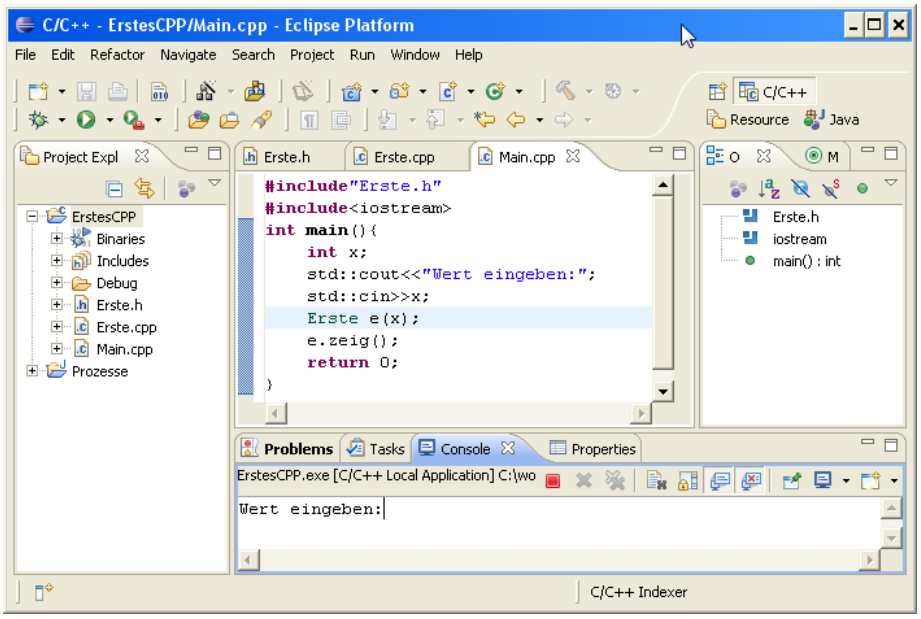


Hier kann u. a. der Name der Konfiguration angegeben werden. Weiterhin muss unter „C/C++ Application“ eine ausführbare Datei gesucht werden. Der erweiterte Modus ist gerade dann sinnvoll, wenn man Werte eingeben möchte, die eigentlich über die Kommandozeile eingegeben werden. Hierfür steht der Reiter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

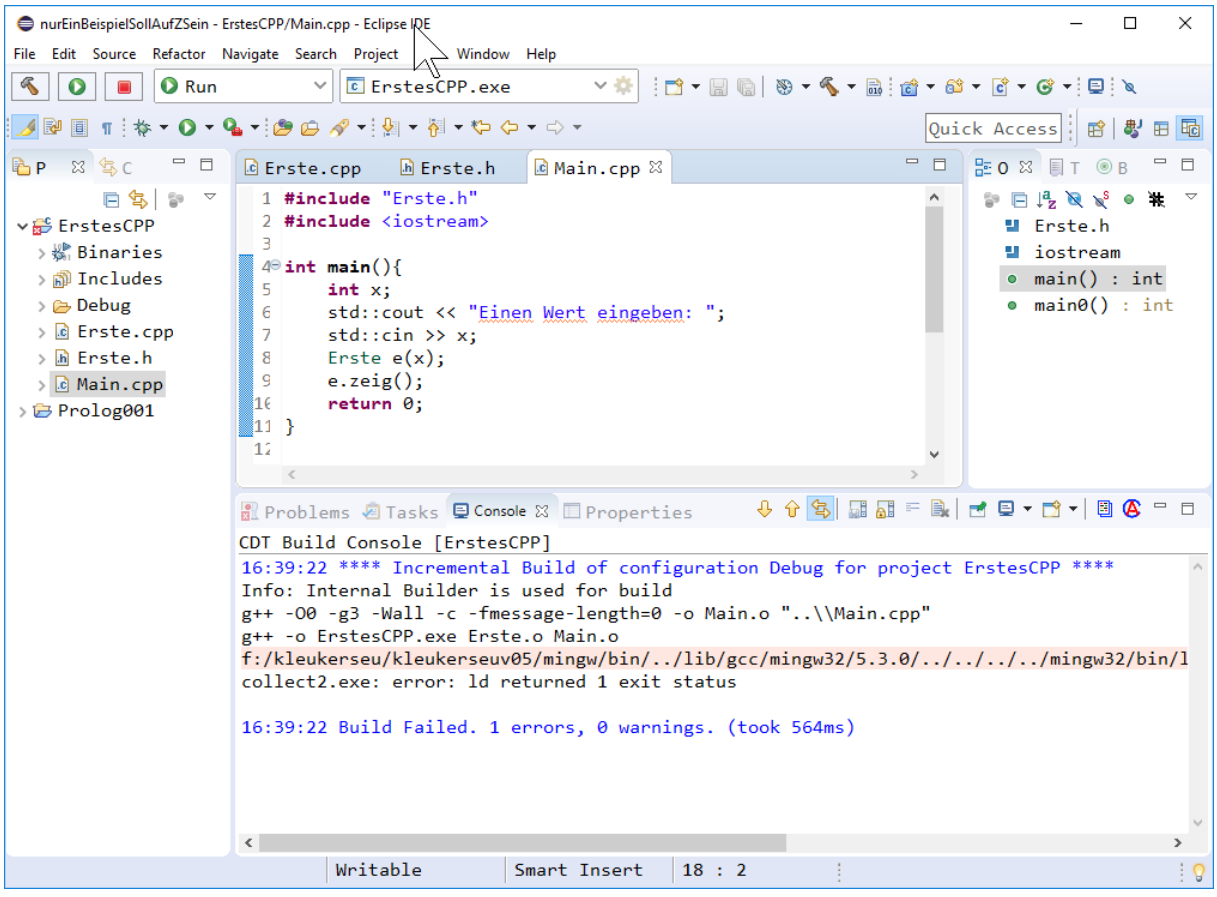


Anders als in Java kann es Probleme geben, wenn man vergisst, ein Programm zu terminieren und dieses erneut kompiliert. Als Beispiel wird folgende Variante der Main-Funktion genutzt, die zunächst eine Eingabe erwartet. Man erkennt, dass unten beim Programm kein <terminated> steht. Eingaben können in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

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

int main(){
	int x;
	std::cout << "Wert eingeben: ";
	std::cin >> x;
	Erste e(x);
	e.zeig();
	return 0;

}
```

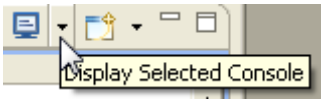


Jetzt kann man während das Programm läuft eine Datei ändern. Drückt man dann erneut auf den Build-Button, erhält man folgendes Ergebnis.



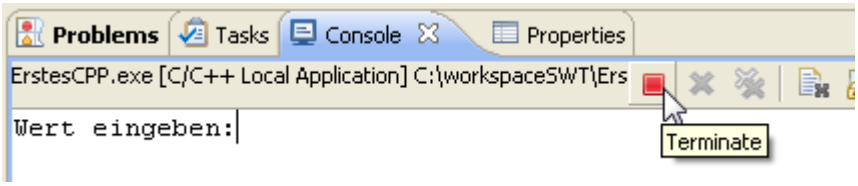
Die Ausgabe bedeutet, dass ErstesCPP.exe nicht erstellt werden kann, da eben dieses Programm noch läuft. Um sich über noch laufende Programme zu informieren und zu diesen zu wechseln,

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.

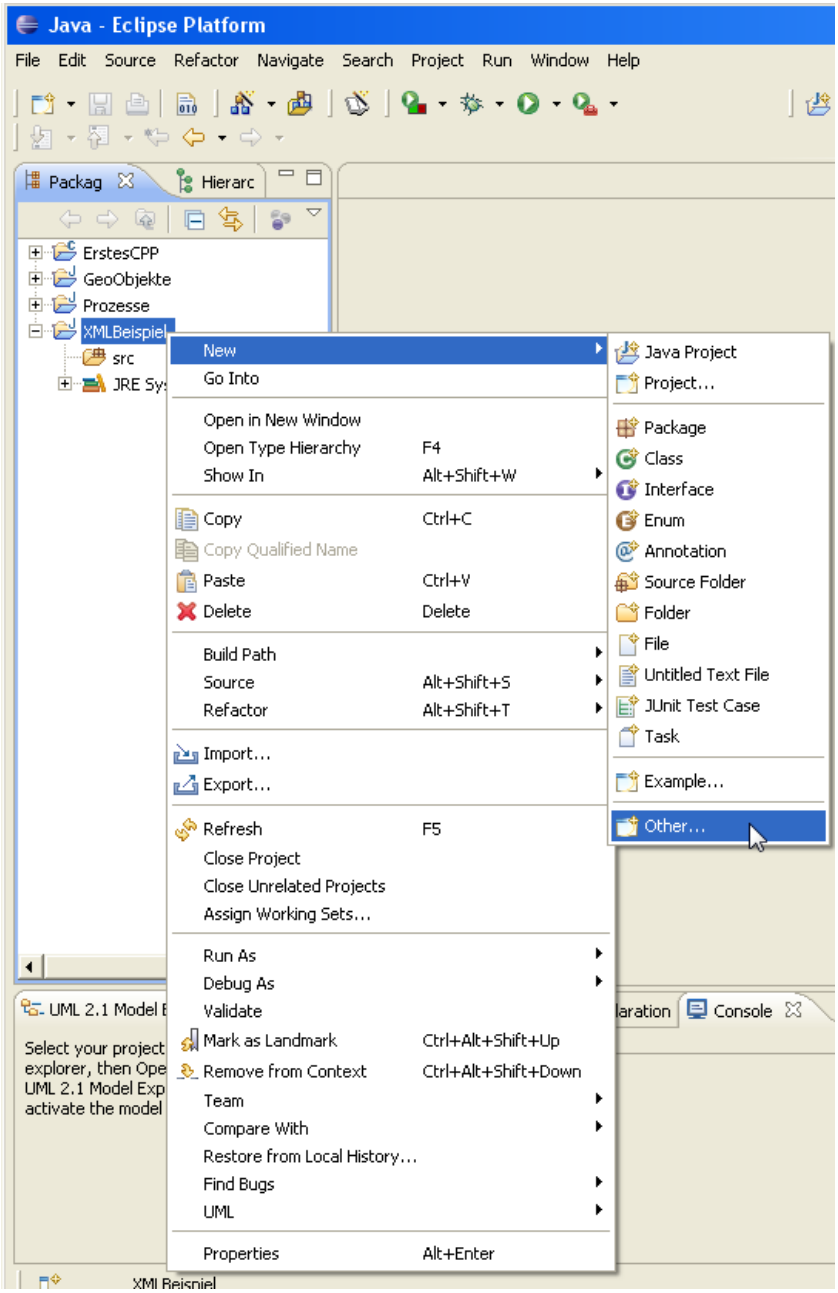


Danach kann erneut kompiliert werden.

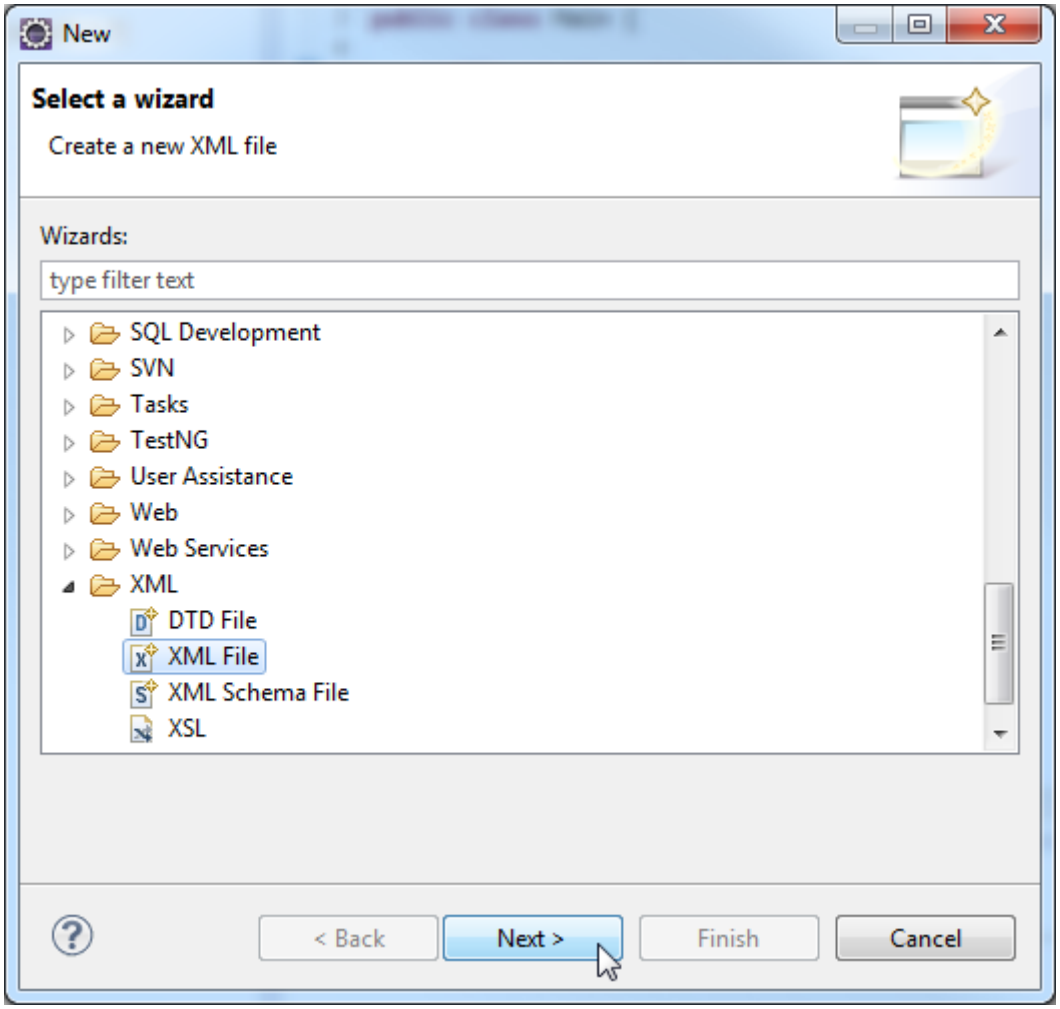
11 XML in Eclipse (4.3.1)

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.

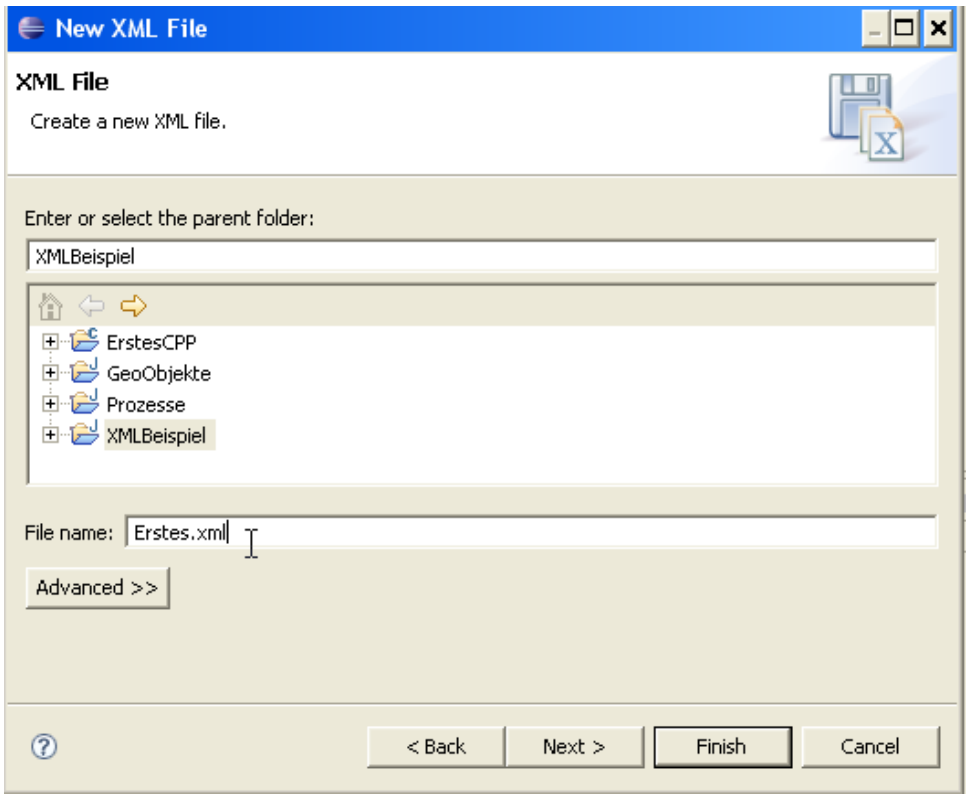
Um mit XML zu arbeiten, muss zunächst ein Java-Projekt angelegt werden (wie es im zugehörigen Kapitel beschrieben ist). In diesem Projekt können dann an beliebigen Stellen XML-Dokumente erstellt werden. Dazu führt man in dem Verzeichnis, das das neue Dokument enthalten soll, z. B. einen Rechtsklick aus,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



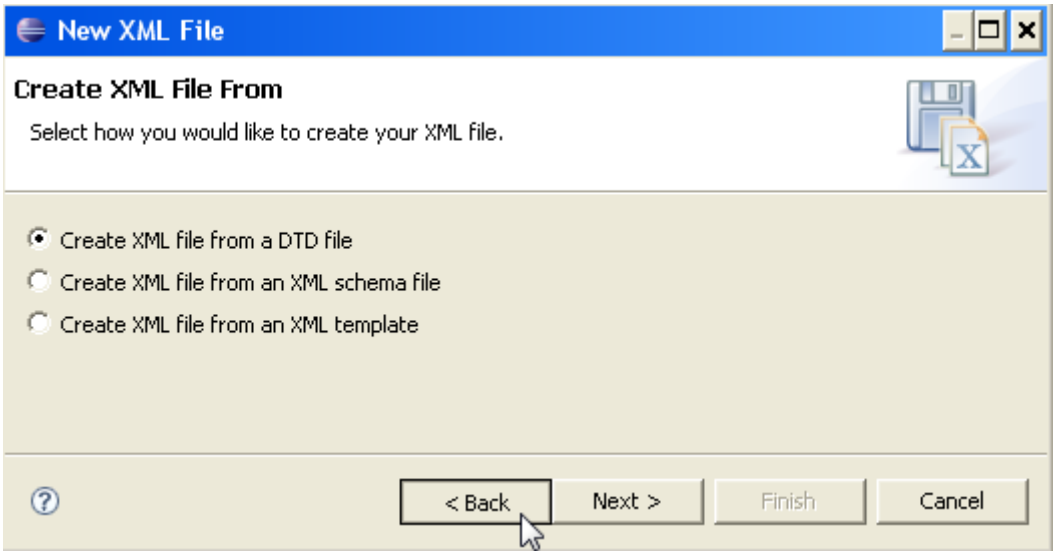
Im folgenden Menü, was in seiner Darstellung stark von den vorhandenen Plugins abhängig ist, wird der Eintrag „XML-File“ gesucht und „Next>“ gedrückt.



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.

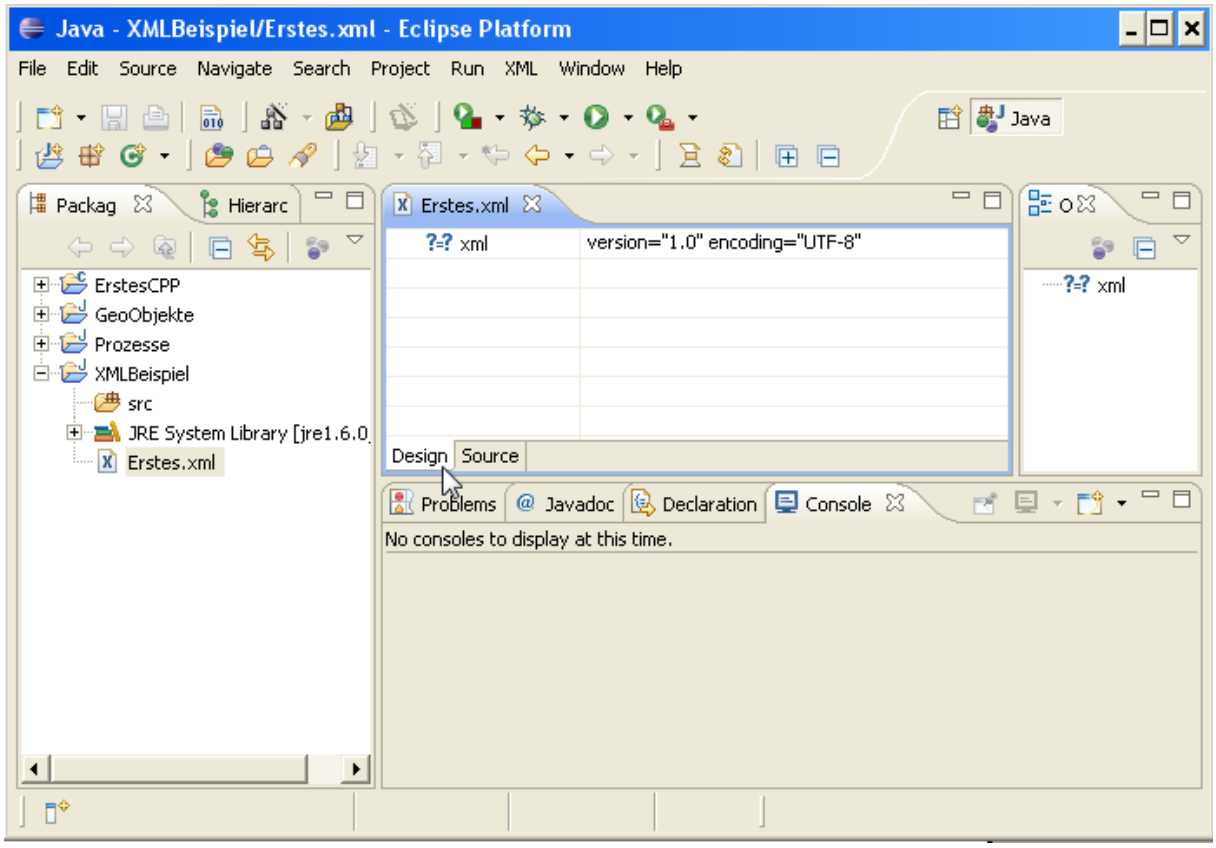


Man hat jetzt die Möglichkeit, festzulegen, auf welcher Basis das XML-Dokument entwickelt werden soll. Nachdem man diese festgelegt hat, kommt man mit „Next>“ zu einem individuellen Dialog, mit dem man die zu nutzende Datei angeben kann. Dieser Weg wird später betrachtet, im Beispiel kehren wir mit „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

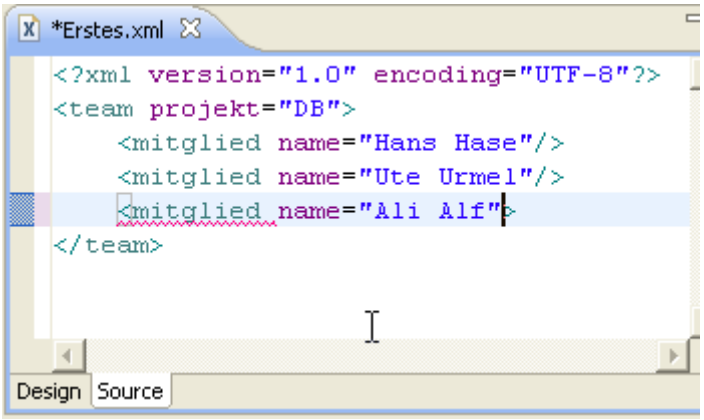


Der XML-Editor besteht aus zwei Varianten. Im Design-Modus wird die Struktur des XML-Dokuments angezeigt. Diese baumartige Anzeige kann u. a. mit Rechtsklicks an der richtigen

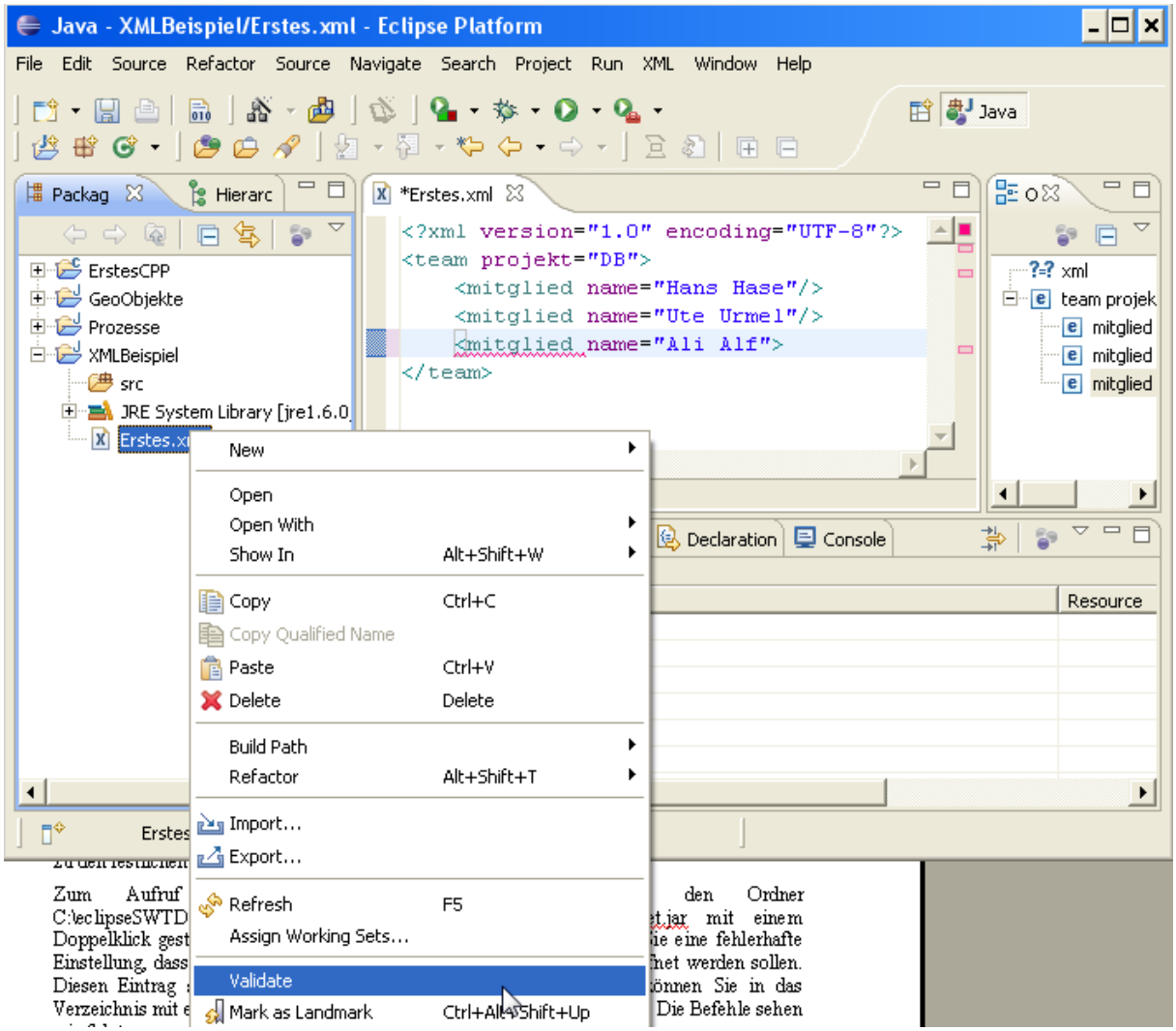
Stelle bearbeitet werden. Gerade für Anfänger wird aber die einfache Source-Ansicht empfohlen, die hier als Einzige genauer betrachtet wird.



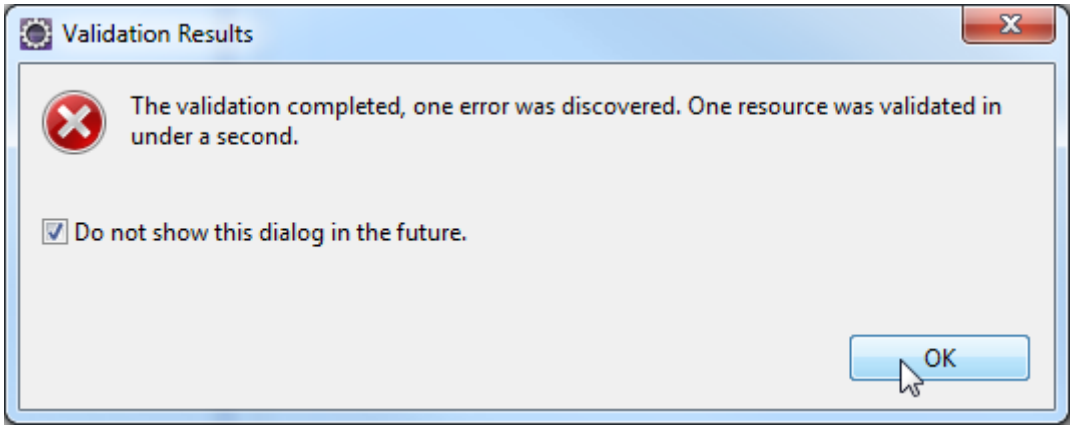
Im Source-Fenster kann dann ein XML-Dokument eingegeben werden. Die Wohlgeformtheit wird bei der Eingabe geprüft, kritische Bereiche rot unterschlängelt, was im folgenden Beispiel für das fehlende End-Tag gilt.



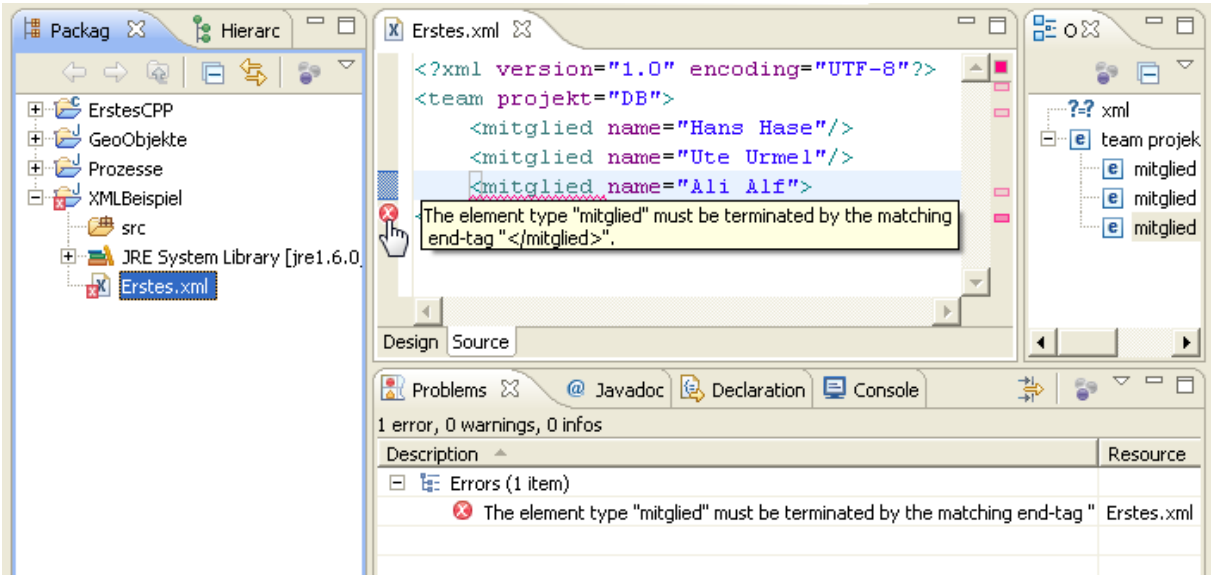
Achtung, nach dem Abspeichern werden Fehler nicht sofort klar gekennzeichnet. Dies passiert erst, wenn z. B. durch einen Rechtsklick auf das Dokument der Punkt „Validate“ ausgewählt wird. Eine Validierung ist auch für Ordner und ganze Projekte möglich.



Die folgende Information kann mit einem Haken links für später abgeschaltet werden, da der Fehler auch Editor angezeigt wird.

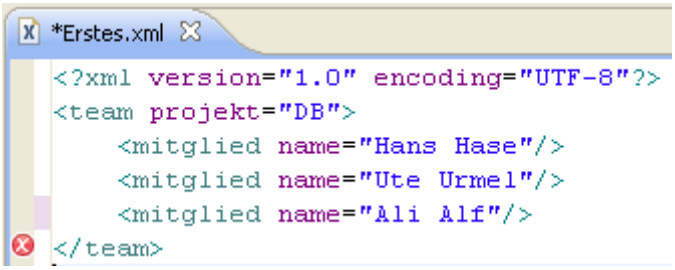


Man erhält dann eine Fehleranzeige, schiebt man die Maus auf das weiße Kreuz im roten Kreis, wird der Fehlertext angezeigt, der auch unten im Reiter „Problems“ sichtbar ist.



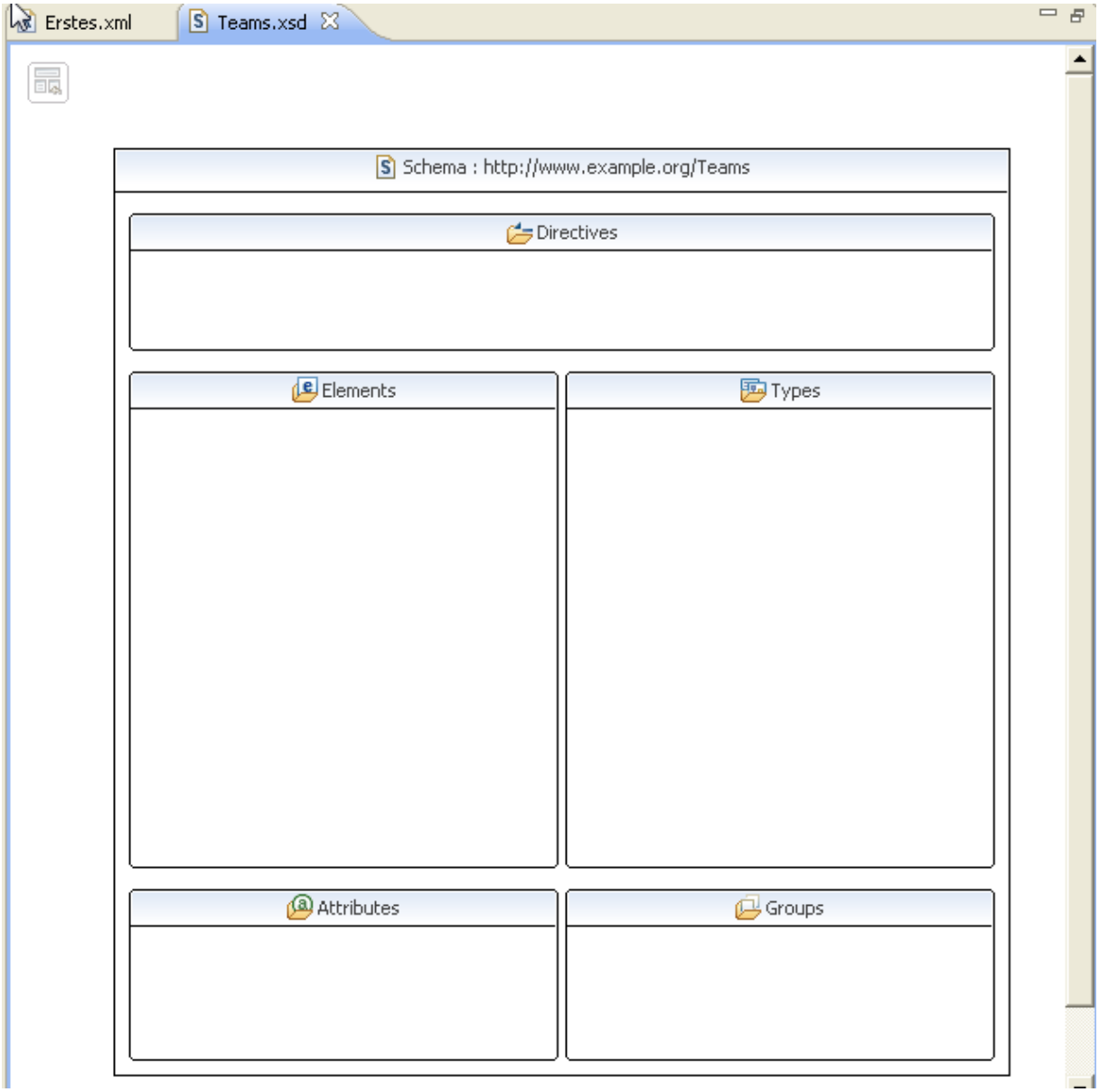
Wird der Fehler korrigiert, ändert sich die Anzeige am Rand nicht. Die Prüfung erfolgt erst, wenn erneut ein Validate ausgeführt wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<team projekt="DB">
  <mitglied name = "Hans Hase"/>
  <mitglied name = "Ute Urmel"/>
  <mitglied name = "Ali Alf"/>
</team>
```



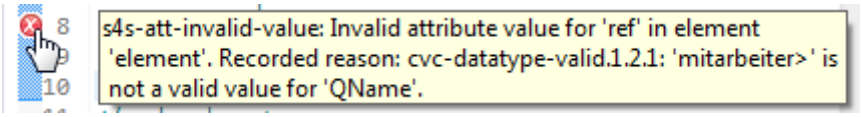
Ähnlich wie XML-Dokumente können auch XML-Schemata erstellt werden. Bei XML-Schemata wird aber durch das Validate geprüft, ob die Anforderungen an ein XML-Schema erfüllt sind, d.h. ob es gültig bezüglich der XML-Schema-Definition ist.

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. Das ursprüngliche schema-Start-Tag wurde in `<xsd:schema xmlns:xsd=` abgeändert.

Die zugehörige leicht längliche Fehlermeldung lautet wie folgt.



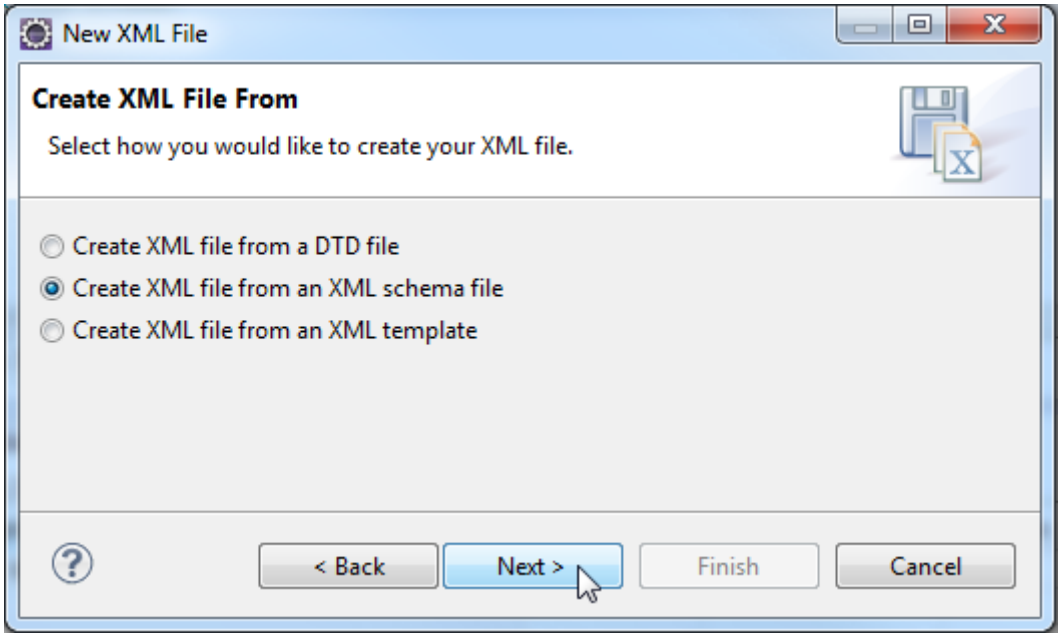
Das Schema wird jetzt wie folgt ergänzt und xmlns:tns als Attribut von schema auf xmlns geändert (erleichtert etwas die Dokumentenerstellung).

```
<?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>
```

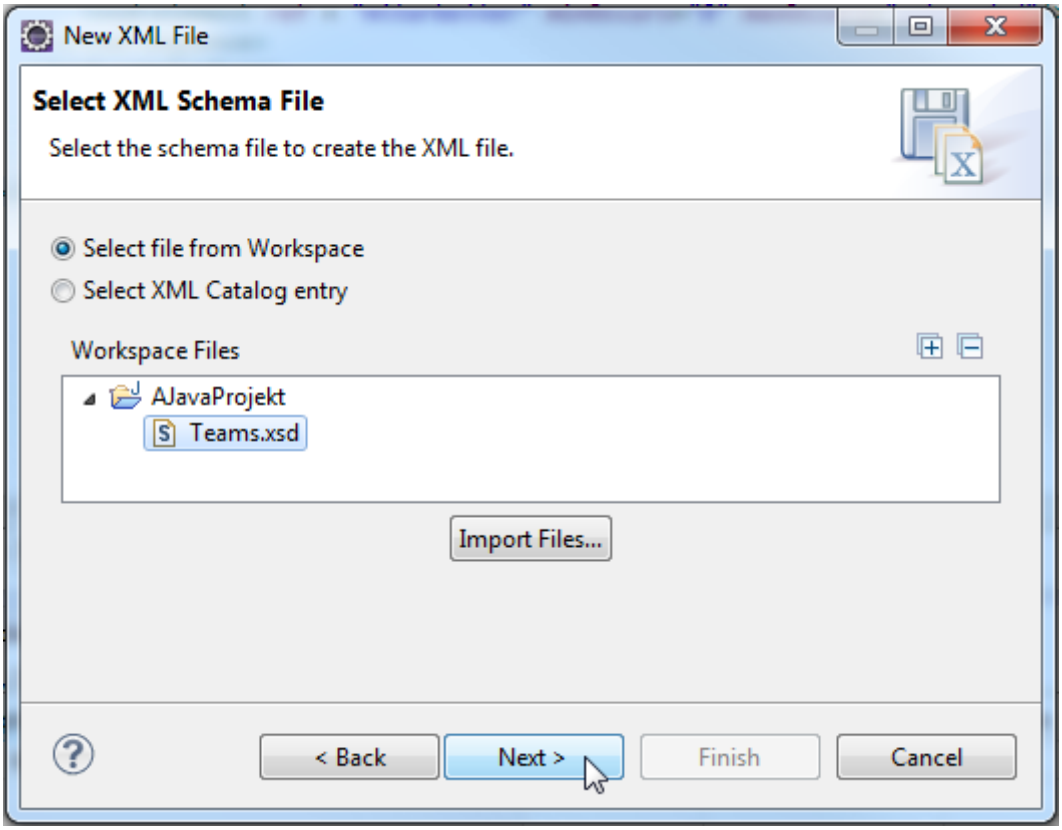
Erstes.xml | Teams.xsd

```
<?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>
```

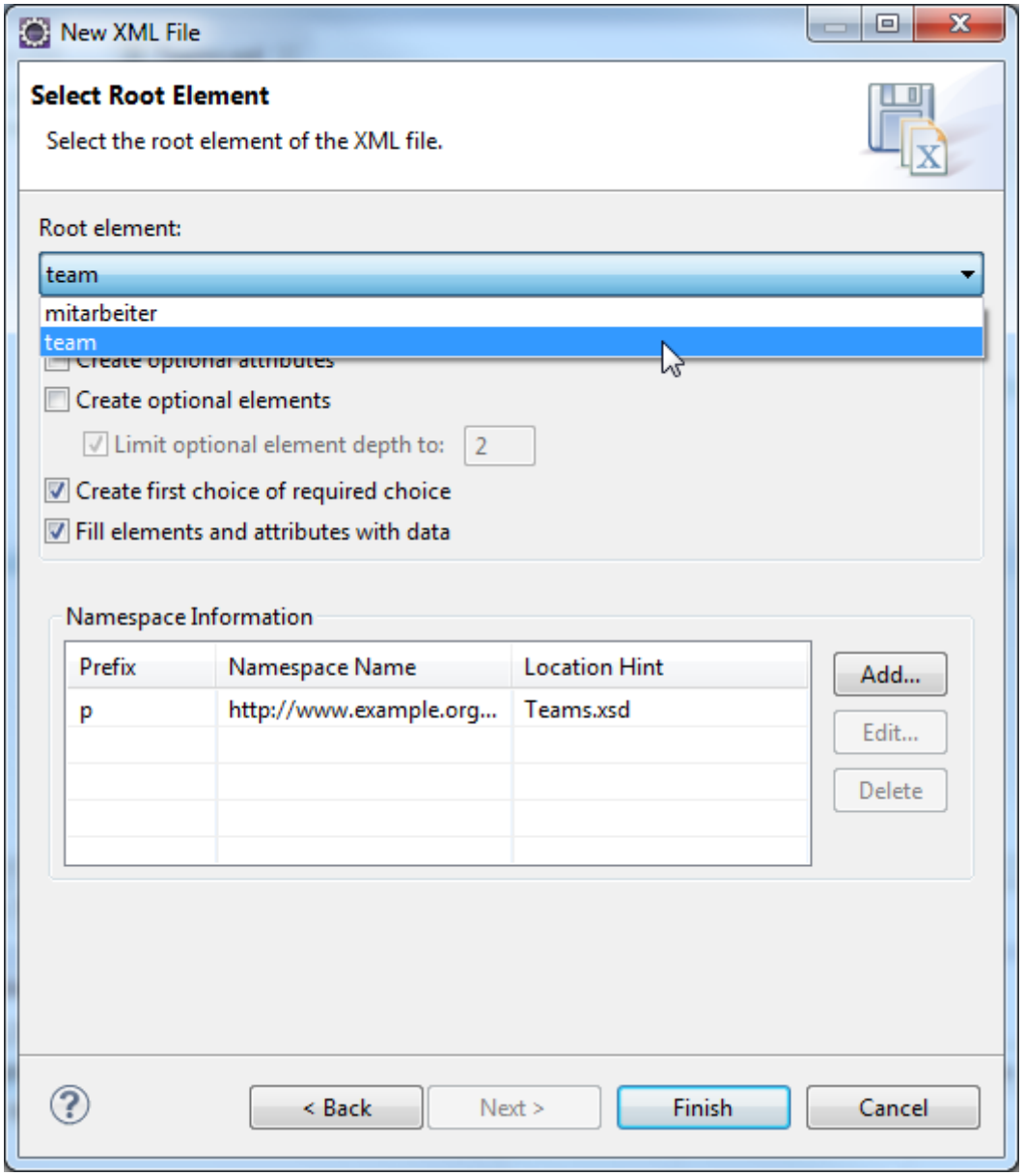
Es soll jetzt ein zum Schema passendes XML-Dokument angelegt werden. Dazu wird wie vorher beschrieben eine neue Datei angelegt und angegeben, dass diese zu einem existierenden XML-Schema passen soll.



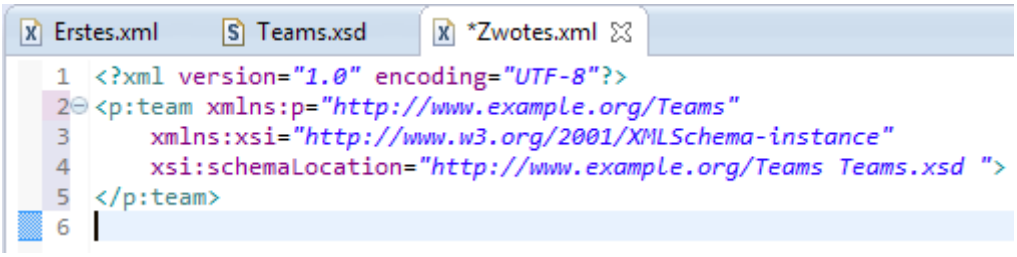
Danach kann das passende XML-Schema gesucht werden.



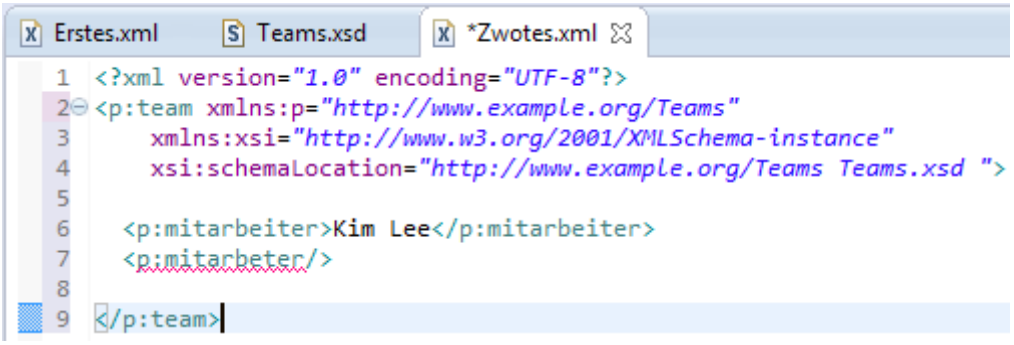
Danach wird das Wurzel-Element bestimmt. Weitere Einstellungen können passend ausgesucht werden. Die Erstellung wird mit „Finish“ abgeschlossen.



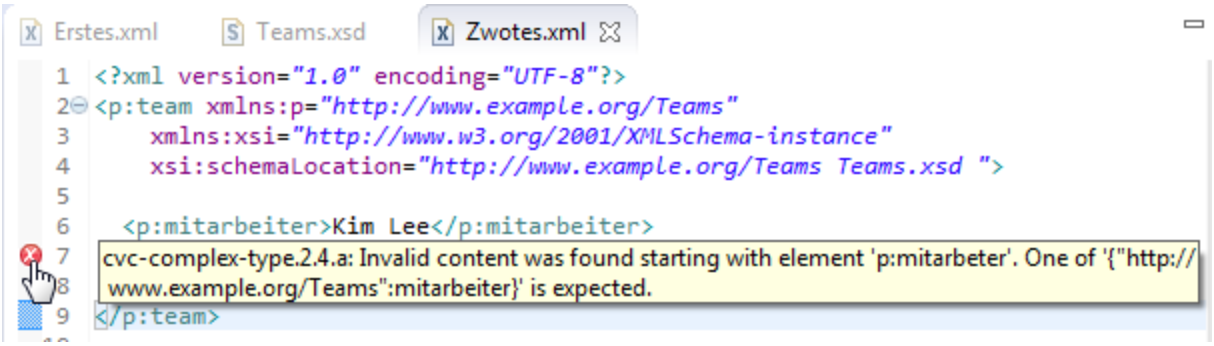
Es wird folgende Datei angelegt, wobei die Namensraumvergabe mit „p:“ wenn gewünscht, von Hand entfernt werden muss. (Von Hand bedeutet, dass es unter „Edit“ den passenden Menü-Punkt „Find/Replace...“ gibt.)



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

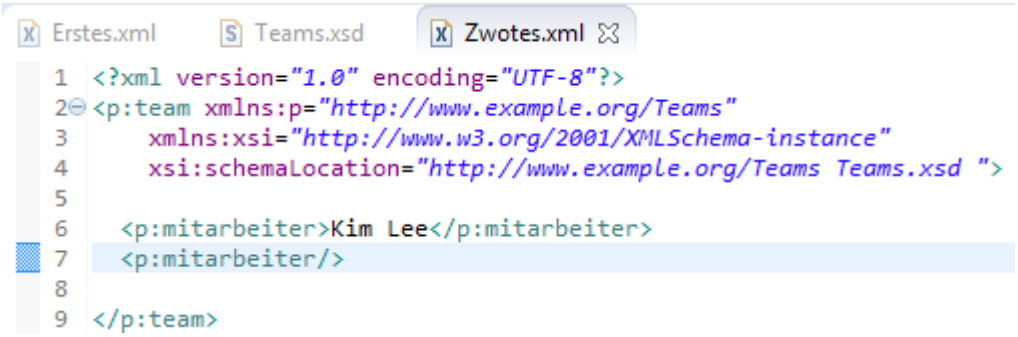


Eine Prüfung auf Gültigkeit findet durch „Validate“ statt, so dass der Fehler konkret angezeigt wird. Die Platzierung der Fehlermeldung kann ab und zu irreführend sein, wenn der Fehler erst mit dem letzten schließenden Tag erkannt wird und so hier die Fehlermeldung z. B. bei der Prüfung von Abhängigkeiten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



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

```
<?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>Kim Lee</p:mitarbeiter>
  <p:mitarbeiter/>

</p:team>
```



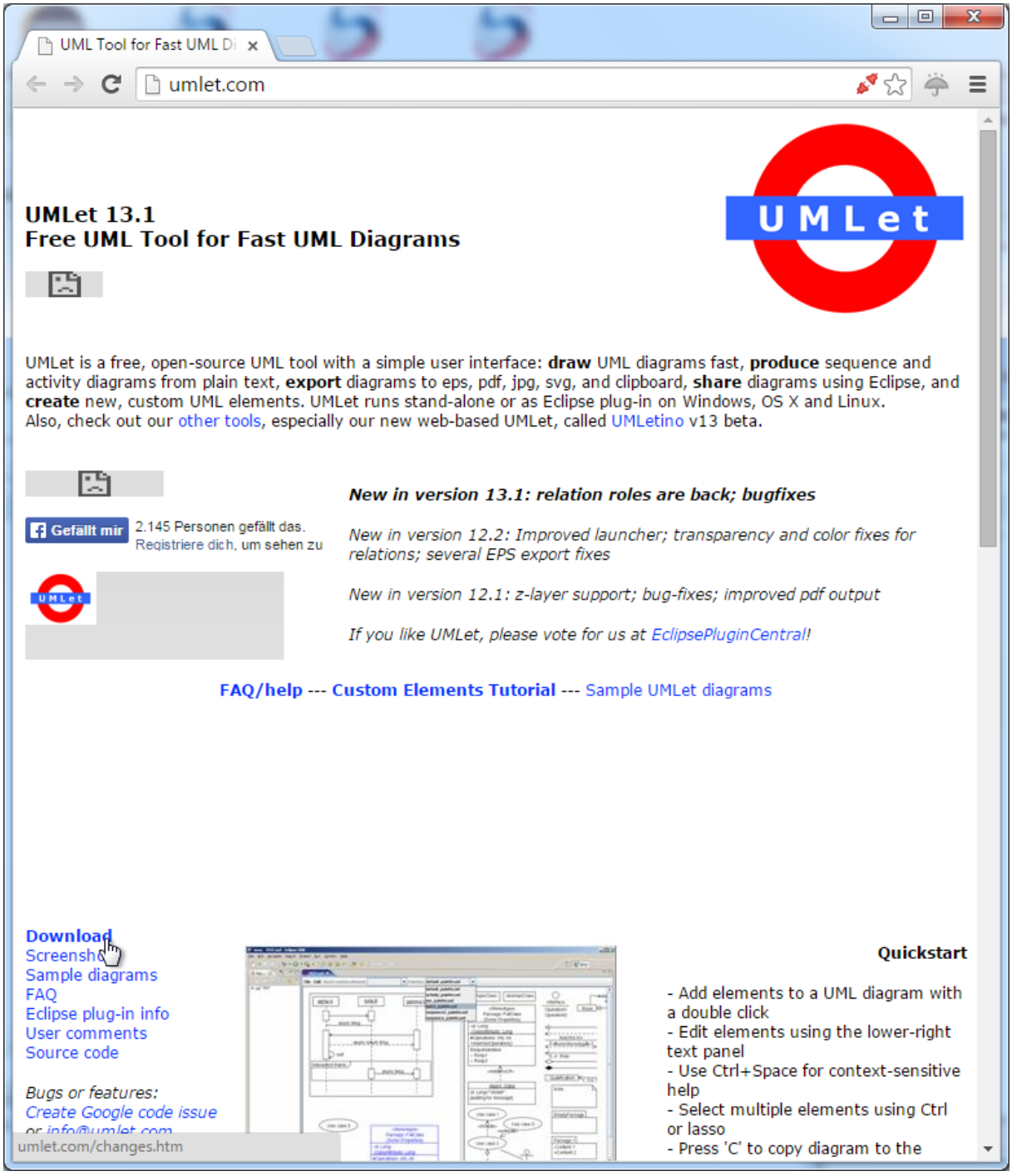
12 Erstellung von Analysemodellen mit UMLet (4.11.0)

Hinweis: Im Text wird die Version 13.1 referenziert, zumindest bis 14.3.0 ist diese Anleitung problemlos nutzbar.

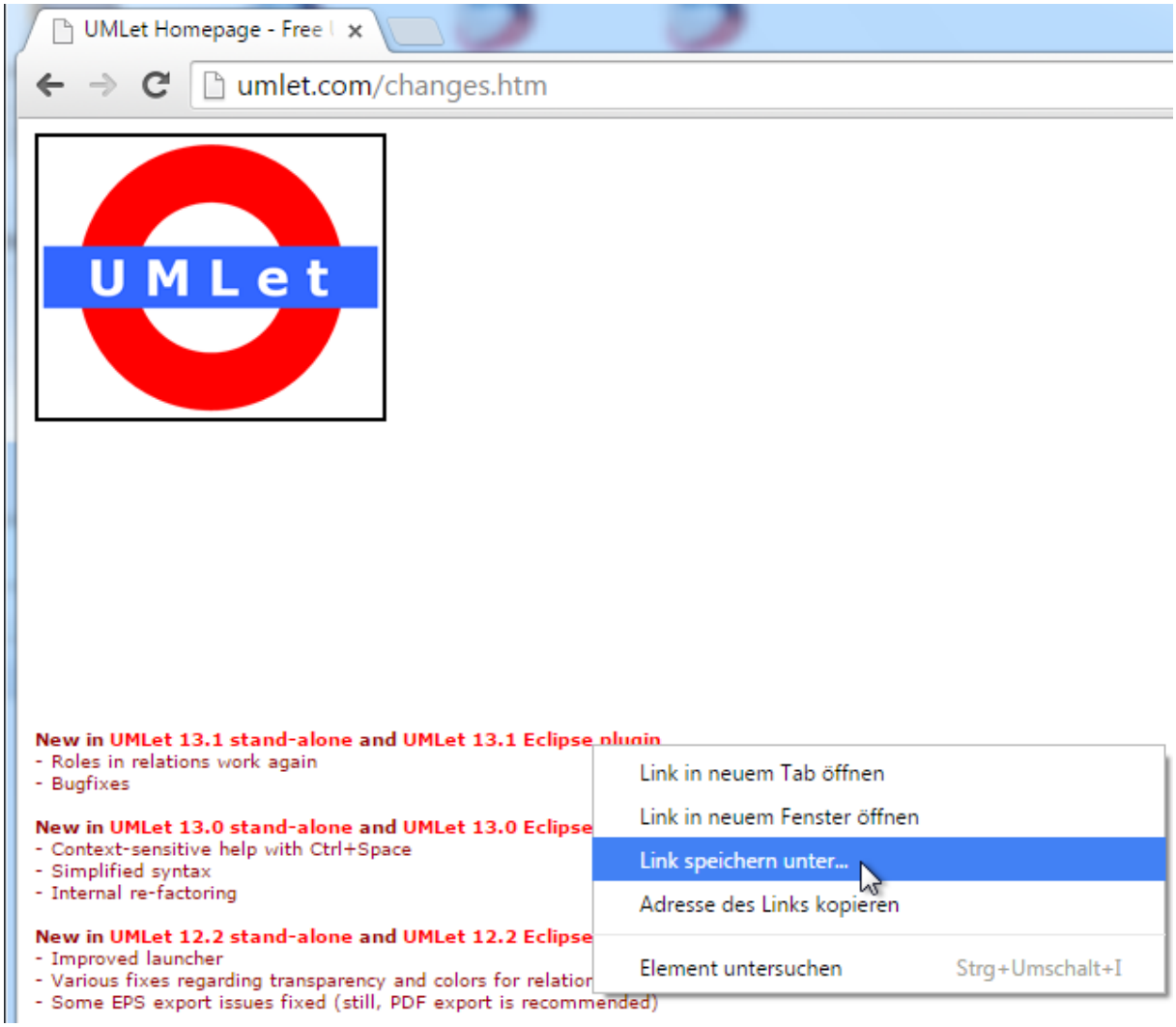
In der Vorlesung wird der pragmatische Ansatz verfolgt, dass nicht alle Dokumente eines Projekts bei jeder Änderung aktualisiert werden müssen. Analysemodelle spielen am Anfang die zentrale Rolle, da sie zum Verständnis der Aufgabenstellung dienen und den Übergang zur Realisierung ermöglichen. Nach einer 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 Eclipse-Teilen genutzt werden. Erfahrungen zeigen, dass die Standalone-Version etwas stabiler läuft.

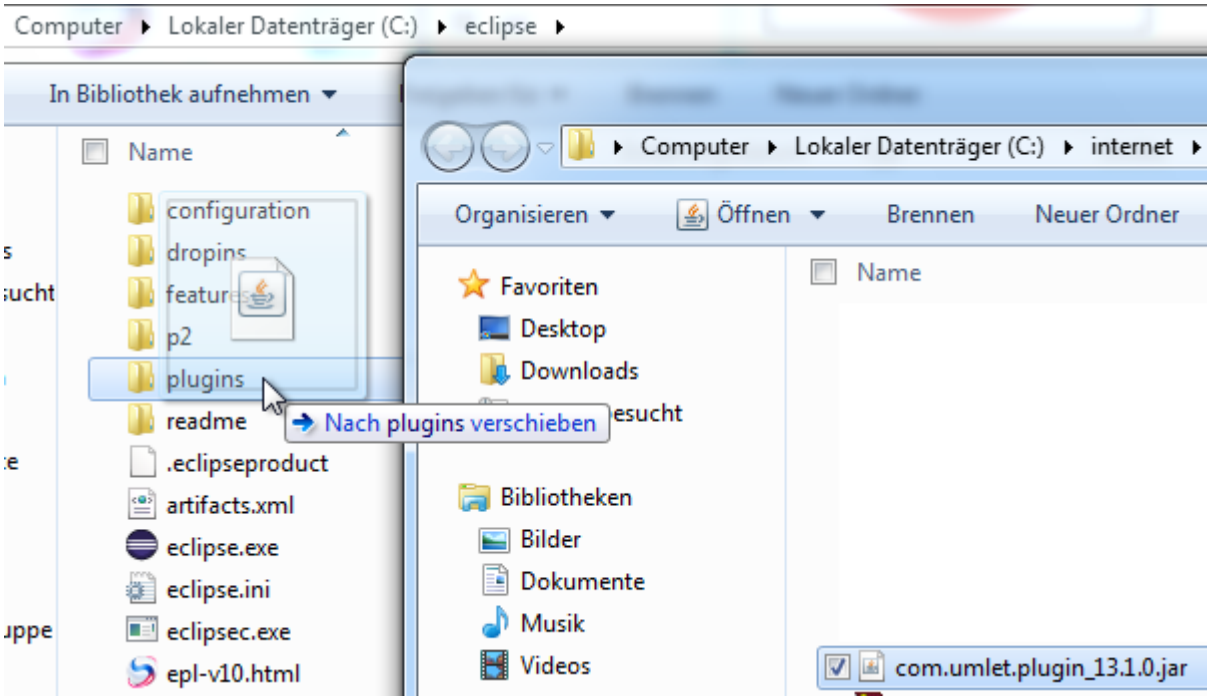
Das Eclipse-Plugin kann von der Web-Seite http://www.umlet.com/ geladen werden. Sollte es größere Probleme geben, ist die Stand-alone-Version zu nutzen. weiterhin befindet sich ein Web-basiertes Werkzeug UMLetino http://www.umlet.com/umletino mit verwandter Funktionalität in Entwicklung. Bei der Nutzung ist darauf zu achten, dass erstellte Dateien wirklich als Dateien über den Browser abgespeichert werden, da der HTML5-Speicher nicht beliebig lang existieren muss.



Hier wird das aktuelle Eclipse-Plugin, hier 13.1, mit einem Rechtsklick heruntergeladen.

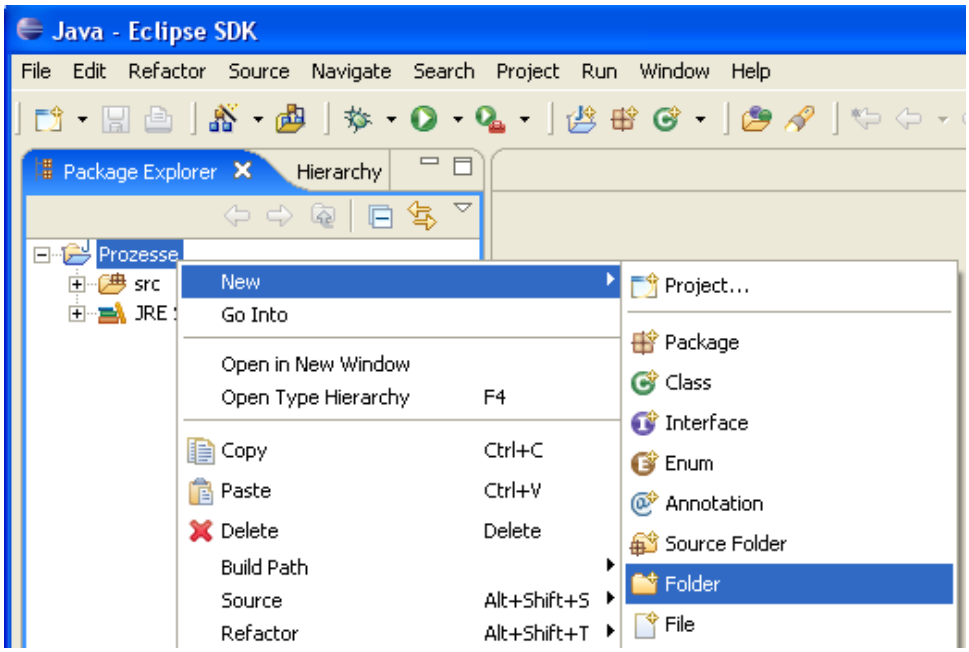


Die jar-Datei com.umlet.plugin_13.1.0.jar muss dann in den plugins-Ordner von Eclipse abgelegt werden. Eclipse sollte dabei nicht laufen.

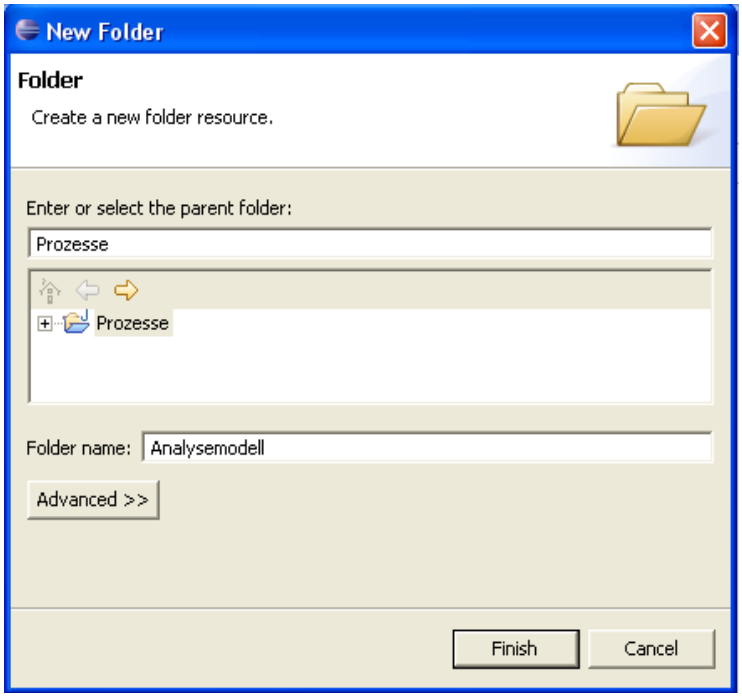


Zum Aufruf des Werkzeugs ohne Eclipse wird die standalone-Variante von der Web-Seite geladen und umlet.jar mit einem Doppelklick gestartet. Falls dann ein Entpackprogramm aufgeht, haben Sie eine fehlerhafte Einstellung, dass Dateien mit der Endung .jar mit diesem Programm geöffnet werden sollen. Diesen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

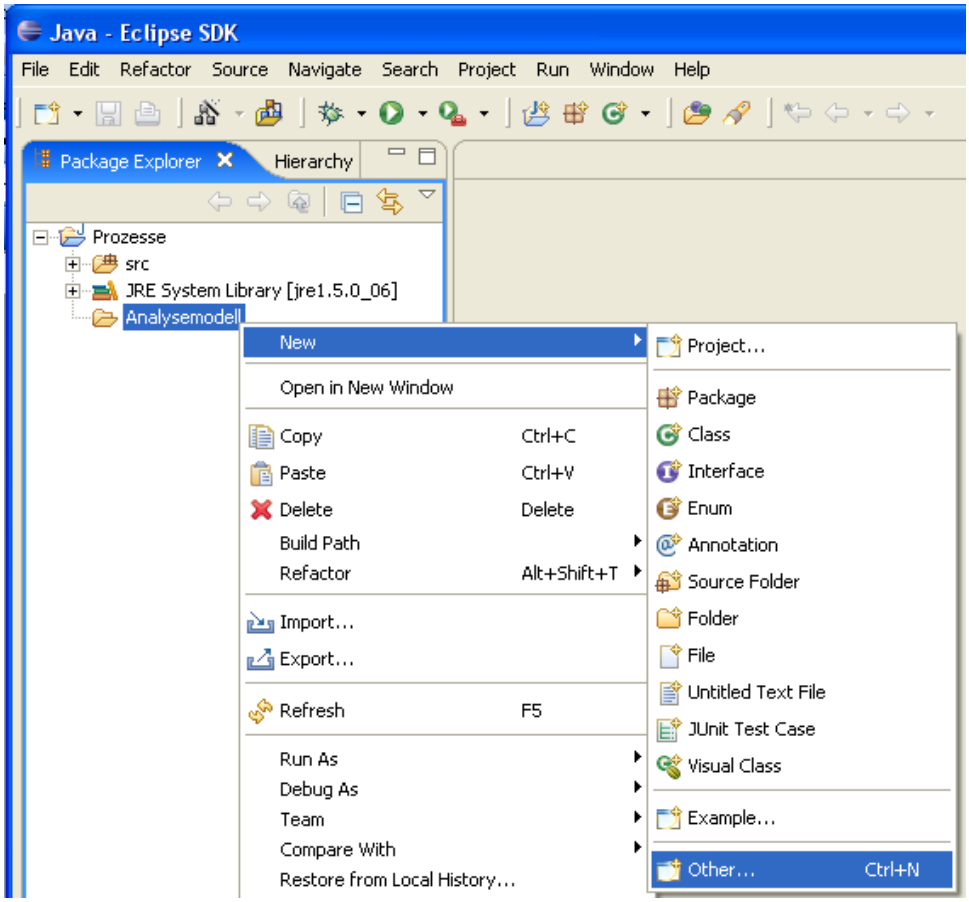
In Eclipse ist es sinnvoll, zunächst ein Projekt und in dem Projekt einen Ordner „Analysemodell“ anzulegen. Dies geschieht durch einen Rechtsklick auf dem Projekt und den Auswahlen „New > Folder“.



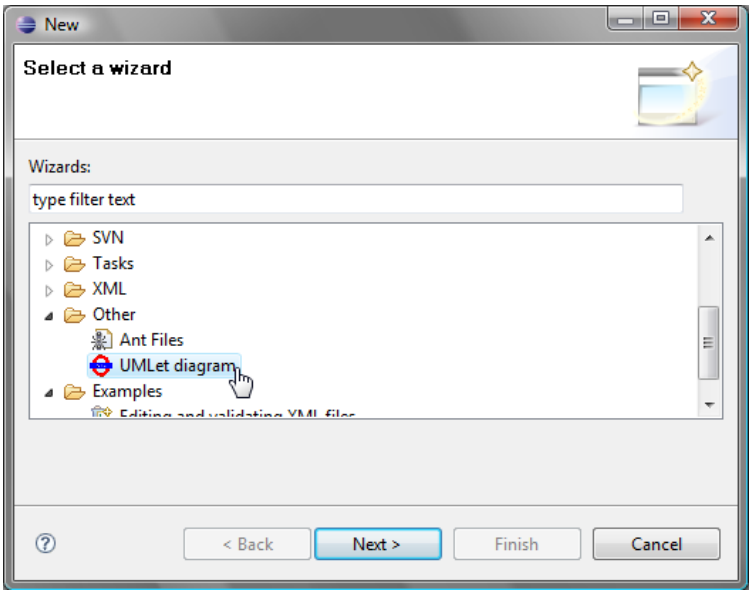
Hier muss nur noch der Name des Ordners eingegeben werden.



Zum Anlegen des Analysemodells wird nach einem Rechtsklick auf den Ordner Analysemodell „New>Other“ ausgewählt. Man beachte, dass abhängig von der genutzten Eclipse-Version die dargestellten Ordner abweichen können. Die Platzierung innerhalb der existierenden Ordner bleibt aber identisch.

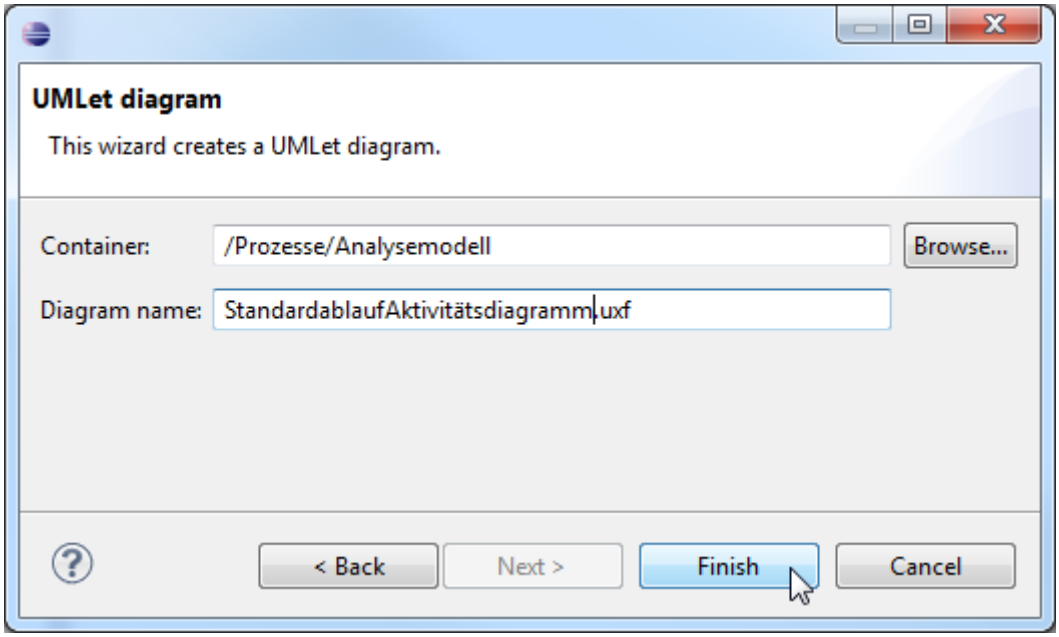


Hier wählt man den Ordner „Other“ und dann „Umlet diagram“. Bitte hier nicht den eventuell existierenden Ordner „UML Diagrams“ auswählen, da diese Diagramme zu einem anderen Werkzeug gehören.

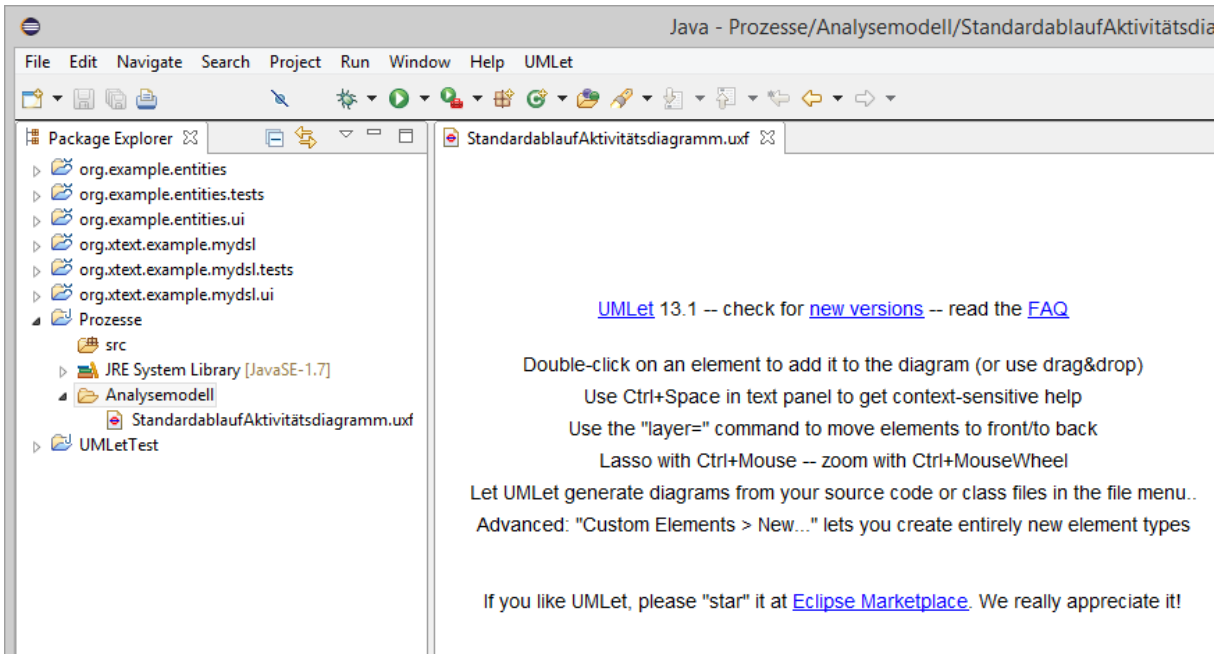


Danach wird „Next>“ gewählt. Im folgenden Fenster kann dann der Name des Diagramms bei „Diagram name:“ angegeben werden. Die Art des Diagramms, also z. B. Use Case Diagram oder Aktivitätsdiagramm, wird erst später im Werkzeug eingestellt. Deshalb ist es sinnvoll, die

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.

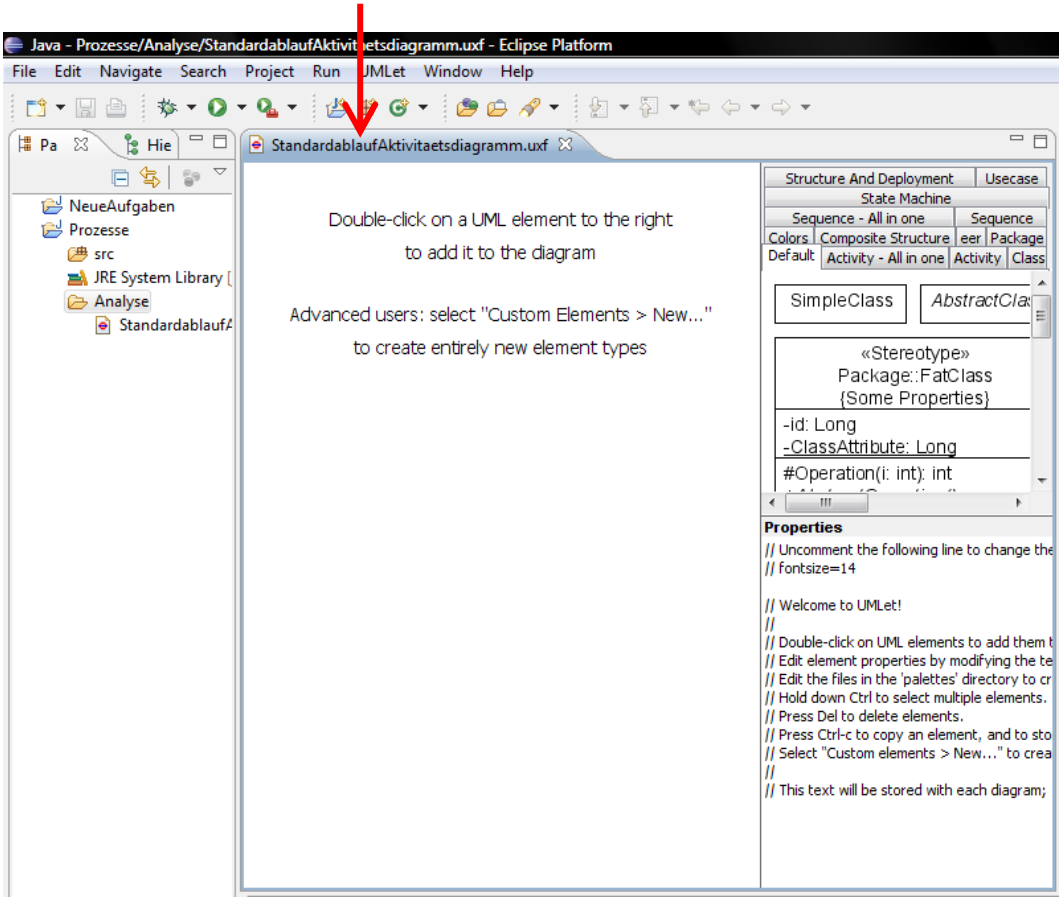


Neben dem Projekt-Browser wird das Diagramm bereits geöffnet. Später wird durch einen Doppelklick auf die uxf-Datei UMLet im zentralen Fenster von Eclipse geöffnet.

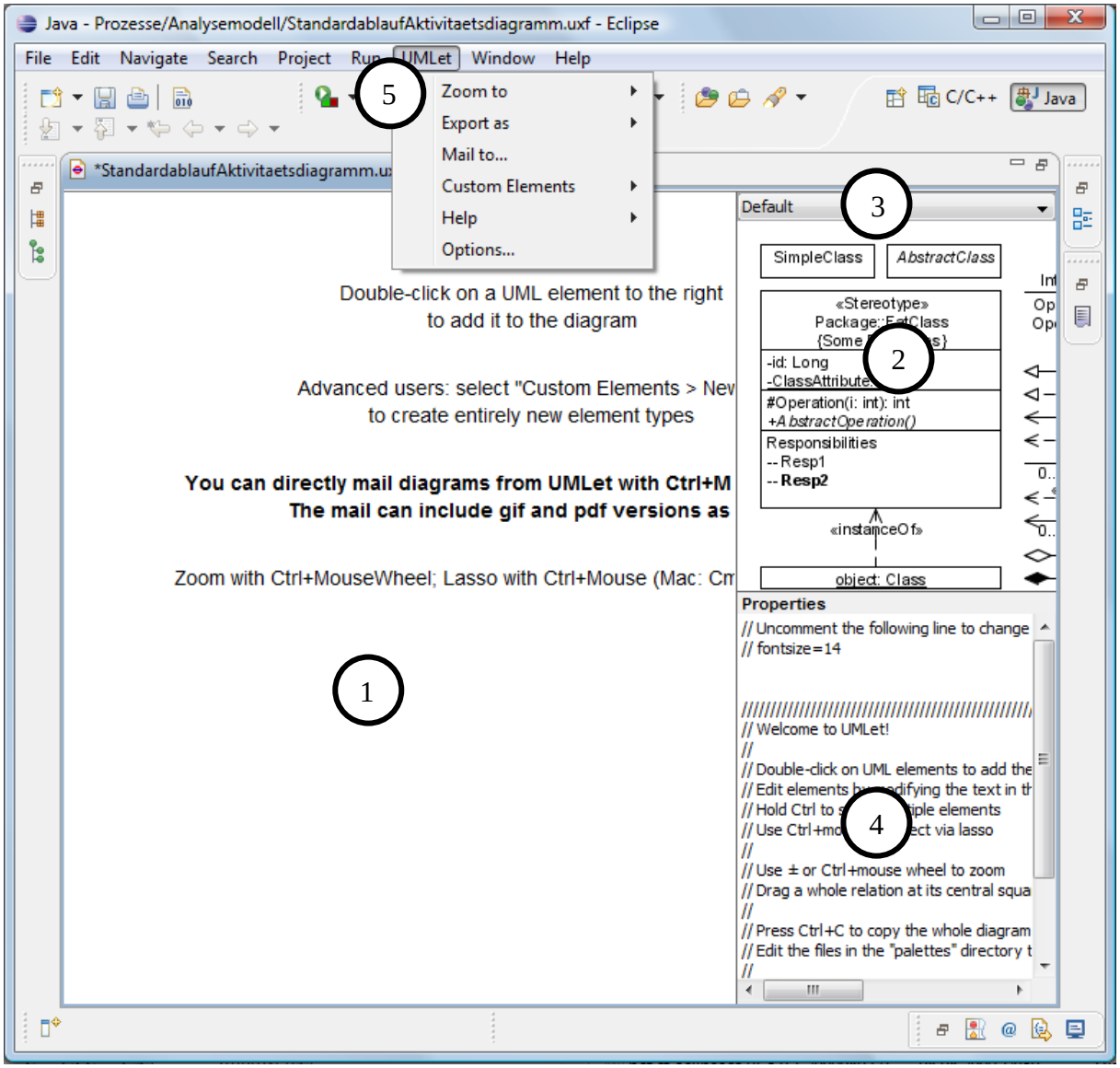
> Falls dies nicht der Fall sein sollte, da das Zusammenspiel Java, Eclipse und UMLet nicht klappt, kann zunächst folgender Trick genutzt werden. Eclipse wird mit der unvollständig geöffneten Datei geschlossen. Danach wird Eclipse erneut geöffnet und

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 nachfolgenden 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.

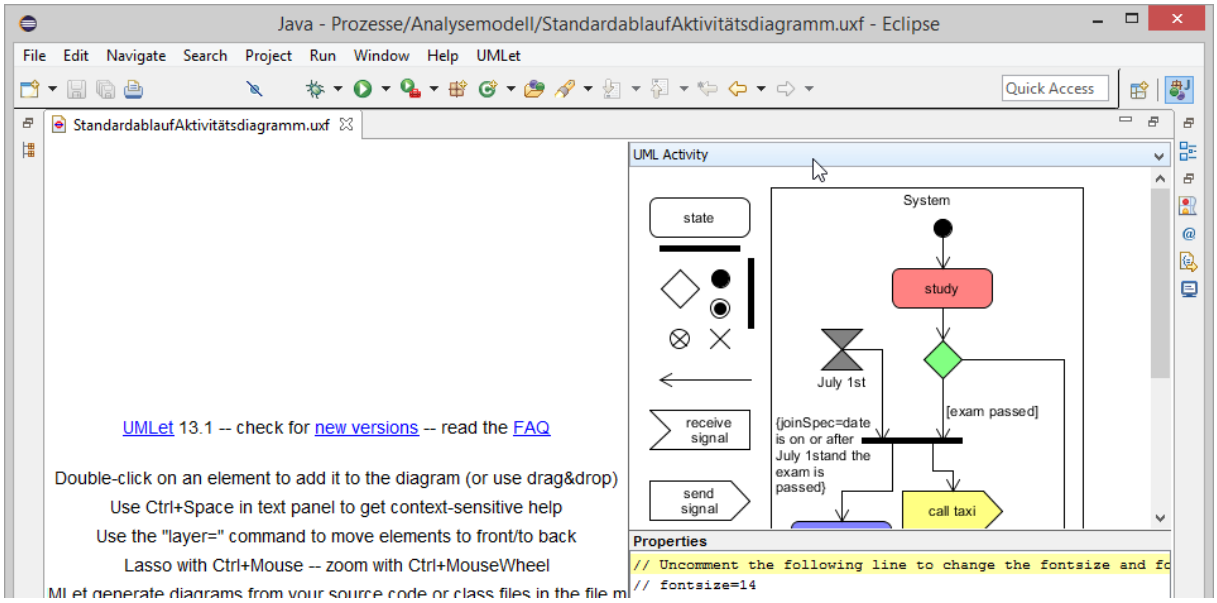
1. Dies ist der Zeichenbereich, in dem alle Elemente verknüpft und angezeigt werden.
2. In diesem Bereich wird eine Palette vom UML-Zeichenelementen angezeigt, die durch einen Doppelklick in den Zeichenbereich übernommen werden. (Diese Art der Steuerung ist etwas gewöhnungsbedürftig, aber recht effizient.)
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) und weitere Bearbeitungsmöglichkeiten (Abbildung ist veraltet).

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

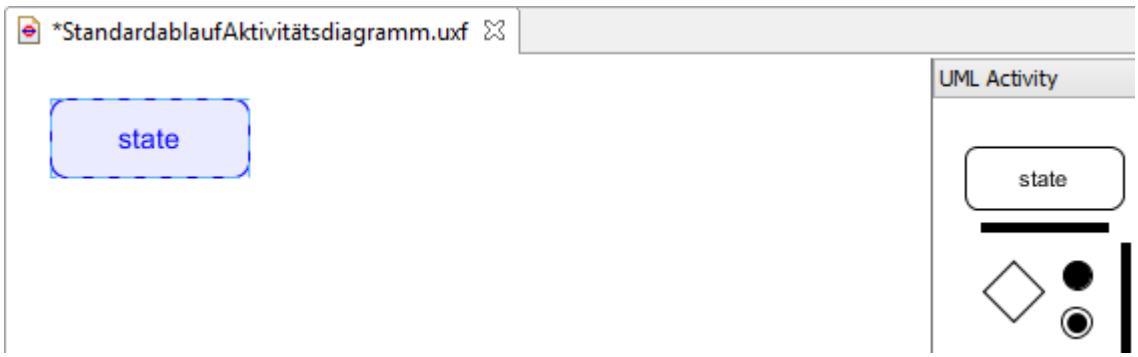
man sich hier frei für eine Darstellungsform entscheidet. Wichtig ist nur, dass sie von jedem Projektbeteiligten gelesen werden kann und dass die Grundideen der Darstellungsweise dokumentiert sind.

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, so dass auch ein Drag-and-Drop von Elementen von der rechten Seite in das Zeichenfeld auf der linken Seite möglich ist. Da dabei allerdings auch Elemente der rechten Seite unabsichtlich verschoben oder gelöscht werden können, ist der Weg mit dem Doppelklick vorzuziehen.

Im Folgenden ist skizziert, wie ein Aktivitätsdiagramm angelegt werden kann, dabei ist jeder Nutzer zum selbst Experimentieren aufgerufen. Zunächst wird die zu Aktivitätsdiagrammen gehörende Palette „UML Activity“ ausgewählt.

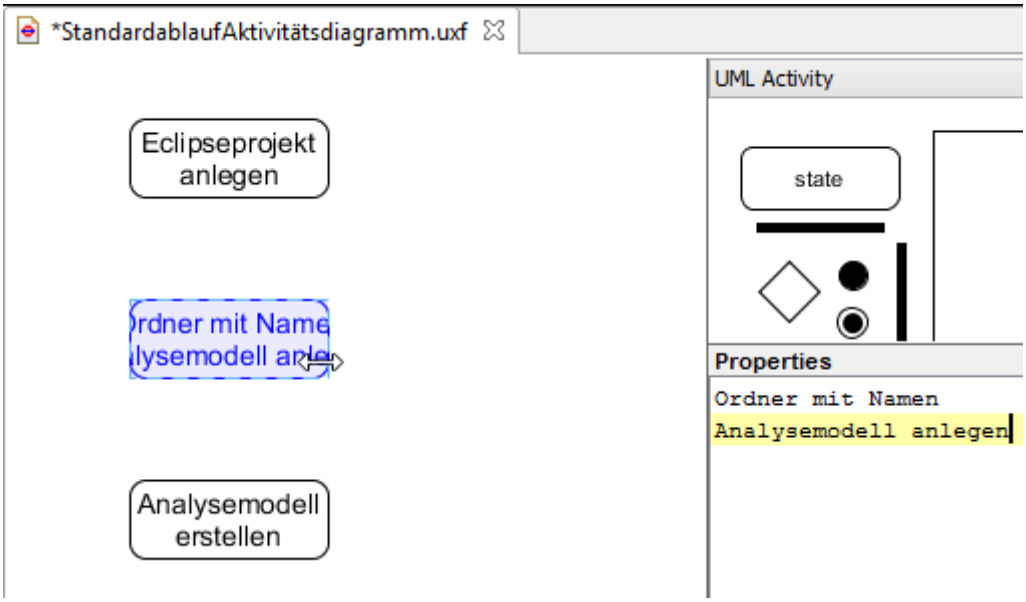


Danach wird auf der rechten Seite die Aktion, etwas unsauber als „state“ bezeichnet, doppelt angeklickt und so in den Zeichenbereich übernommen.

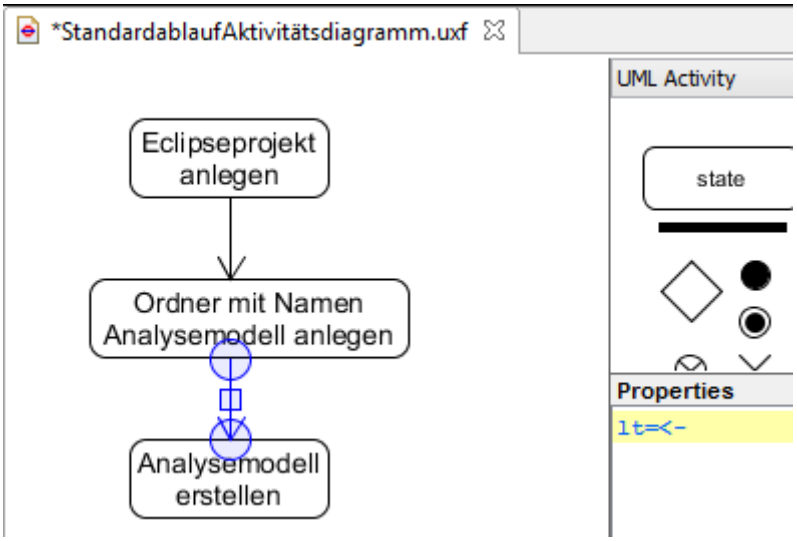


Um mehrere Aktionen zu erzeugen, wird dann ein Doppelklick auf der Aktion im Zeichenbereich ausgeführt. Graphische Elemente können generell mit gedrückter linker Maustaste verschoben und durch einen Doppelklick kopiert werden.

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

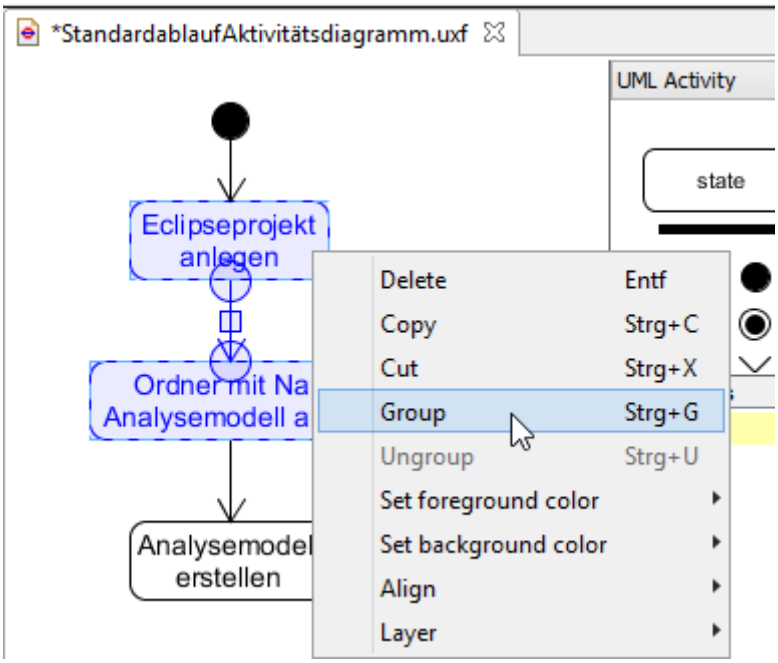


Danach werden die anderen Elemente eingefügt und miteinander verbunden, Pfeile werden ebenfalls mit einem Doppelklick kopiert. Die mit Kreisen markierten Enden können über einen Linksklick verschoben werden. Das Quadrat in der Mitte des Pfeils ermöglicht die Verschiebung des gesamten Pfeils. Sitzen Pfeilende am Rand eines Objektes, werden später mit verschoben.

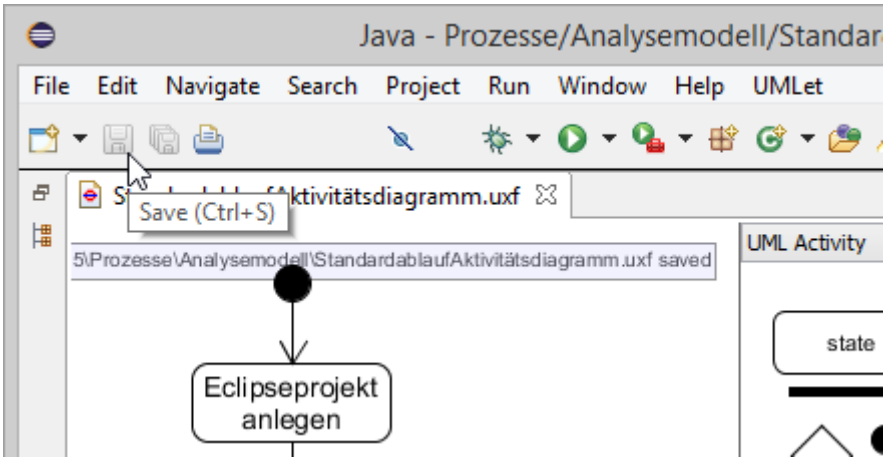


Klickt man neben das Diagramm und verschiebt bei gedrückter linker Maustaste die Maus, so wird das gesamte Diagramm verschoben. Man kann mit gedrückter Strg-Taste mehrere Elemente selektieren und diese dann mit einem Rechtsklick im dann sichtbaren Menü zu einer

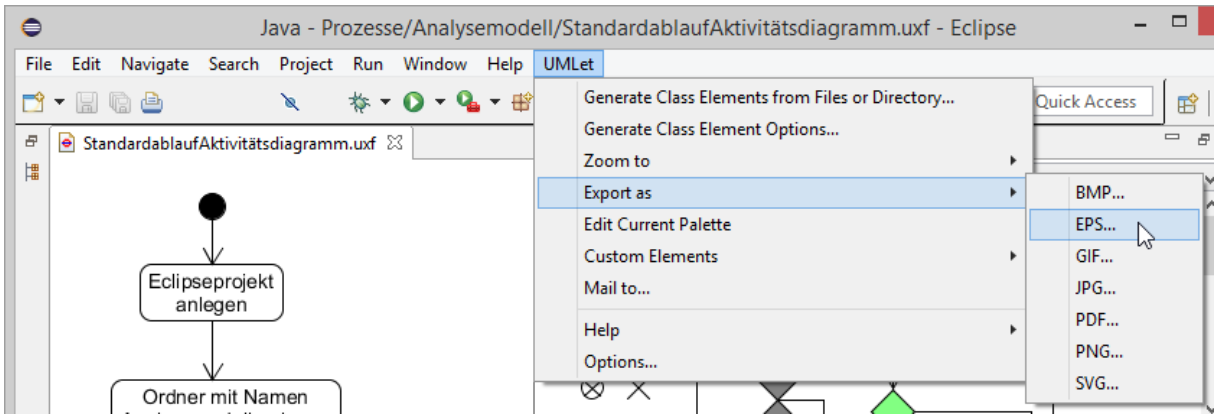
Gruppe zusammenfassen. Danach wird diese Gruppe immer zusammenbehandelt, was z. B. für das Verschieben einer Teilmenge der Diagrammelemente sehr hilfreich ist.



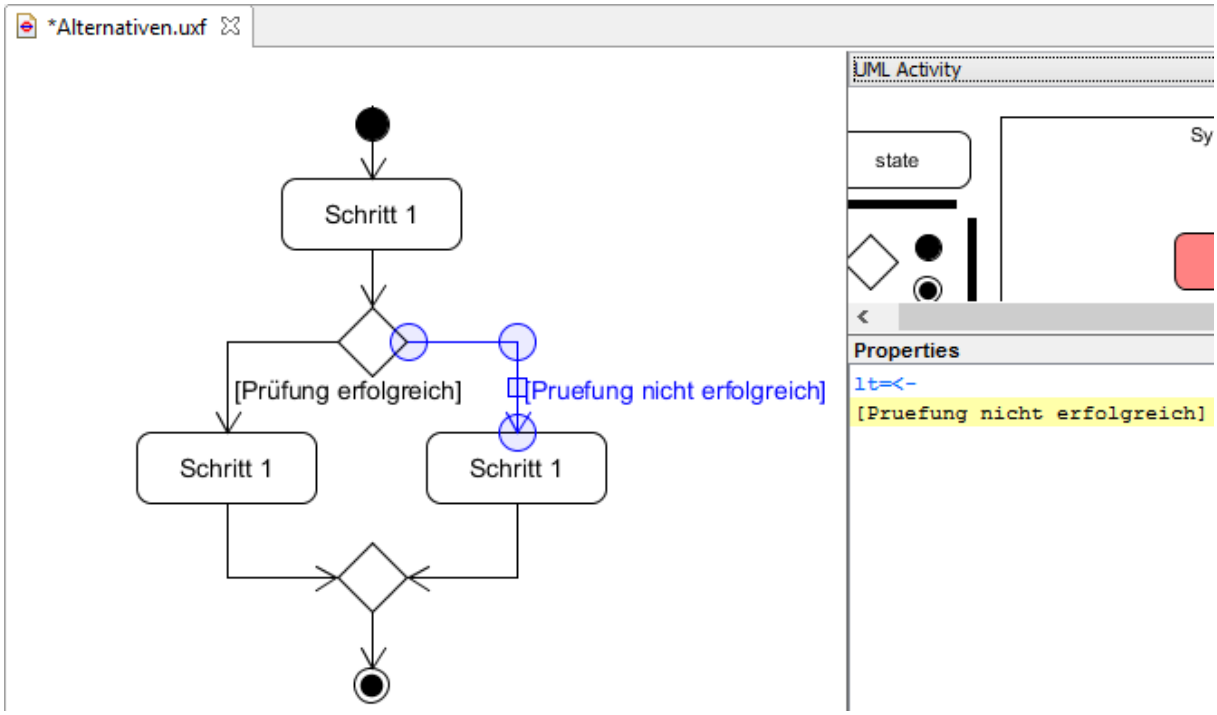
Nachdem ein Modell vollständig eingegeben wurde, muss es mit dem Punkt „Save“ im File-Menü der Eclipse-Umgebung gespeichert werden. Eine Umbenennung ist hier nicht möglich! Leider wird auch der Stern für noch nicht gespeicherte Dateien nicht immer weggenommen.



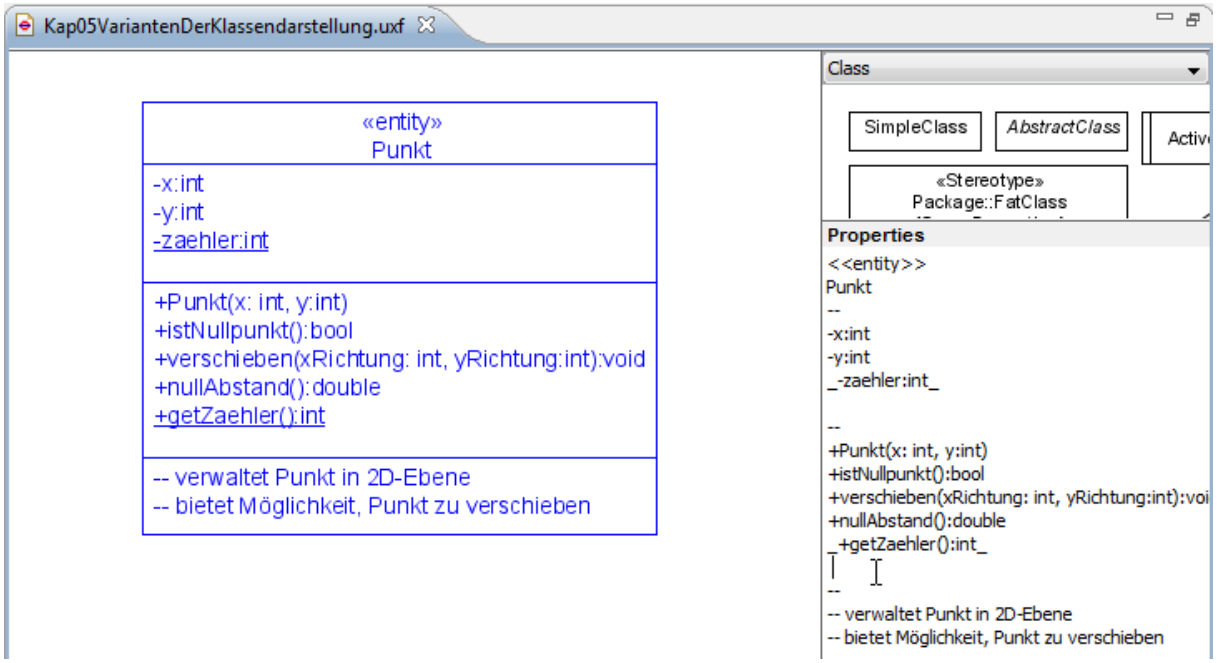
Im UMLet-Menü der Hauptzeile kann man das Bild in andere Formate exportieren, dabei können diese Formate nicht wieder eingelesen werden. Für Ausdrucke ist besonders Encapsulated Postscript zu empfehlen, dass zwar in der Bildschirmdarstellung in Word und Powerpoint sehr schlecht, aber in Ausdrucken und 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).



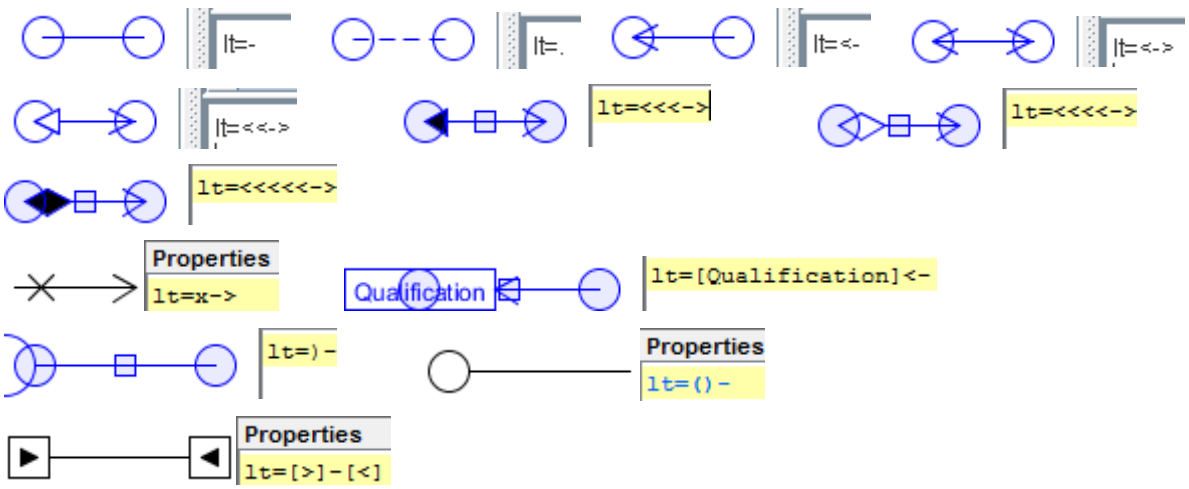
Möchte man Kanten z. B. bei Entscheidungen beschriften, ist zu beachten, dass die erste Zeile in der Beschriftungszeile gleich bleibt, da mit ihr angegeben wird, an welchen Enden Pfeilspitzen stehen sollen. Die folgende Abbildung zeigt links eine selektierte Kante und rechts das Textfeld, in dem die Beschriftung eingetragen wird. Knicke in Kanten erzeugt man, wenn man auf einer selektierten Kante an dem Punkt, an dem 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 und kann nicht zur Knickerstellung genutzt werden.



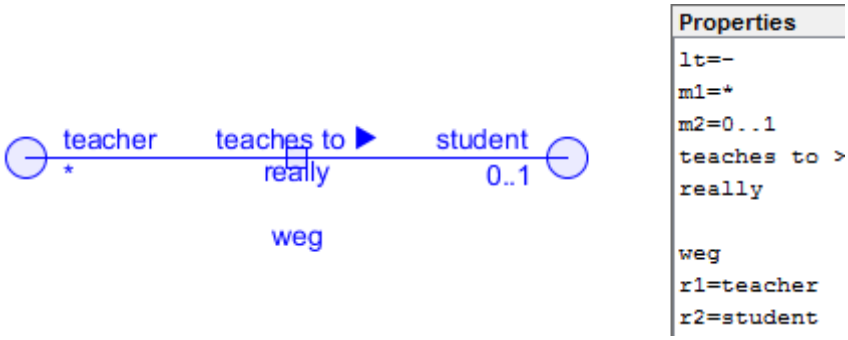
Bei der Bearbeitung von Klassen ist noch zu beachten, dass bei Texteingaben eine leere Zeile mit zwei Minuszeichen dazu führt, dass im Klassendiagramm ein Strich gemalt wird. Möchte man Leerzeilen einfügen, müssen diese mindestens ein Leerzeichen enthalten. Klassenvariablen und Klassenmethoden werden 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.



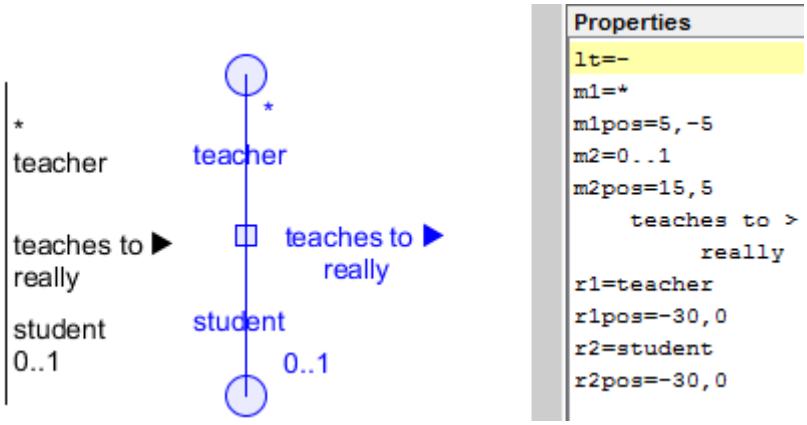
Beschäftigt man sich etwas genauer mit UMLet, gibt es noch einige wenige Steuerbefehle, die in der Beschriftungsbox eingegeben werden können. Die folgenden Bilder zeigen zunächst links die Linienart und rechts den eingegebenen Text, die Kreise zeigen nur, dass die Linie selektiert wurde.



Weiterhin gibt es verschiedene Plätze, an denen Eigenschaften an Linien notiert werden können.

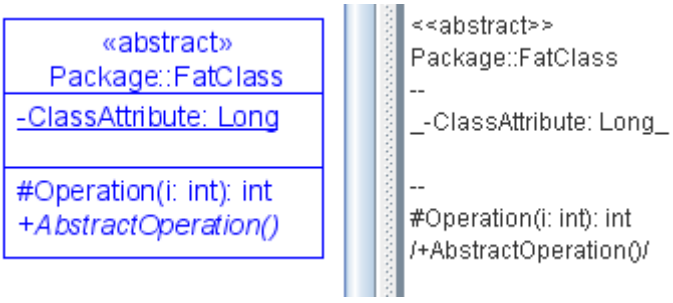


Da die Platzierung nicht immer optimal ist, kann man die Texte mit einzelnen Leerzeichen oder der Angabe von Pixelverschiebungen platzieren.

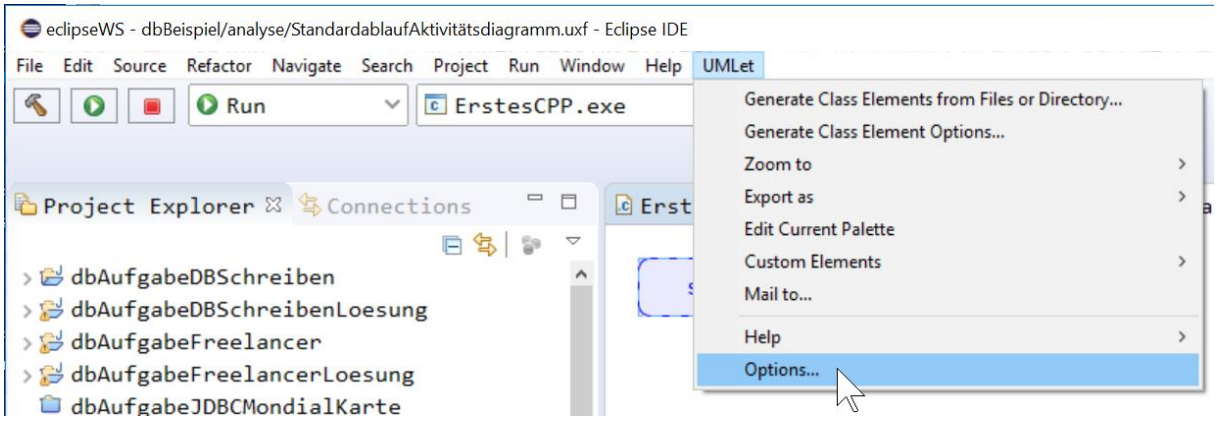


Ausgefüllte Elemente können eine Farbe haben.

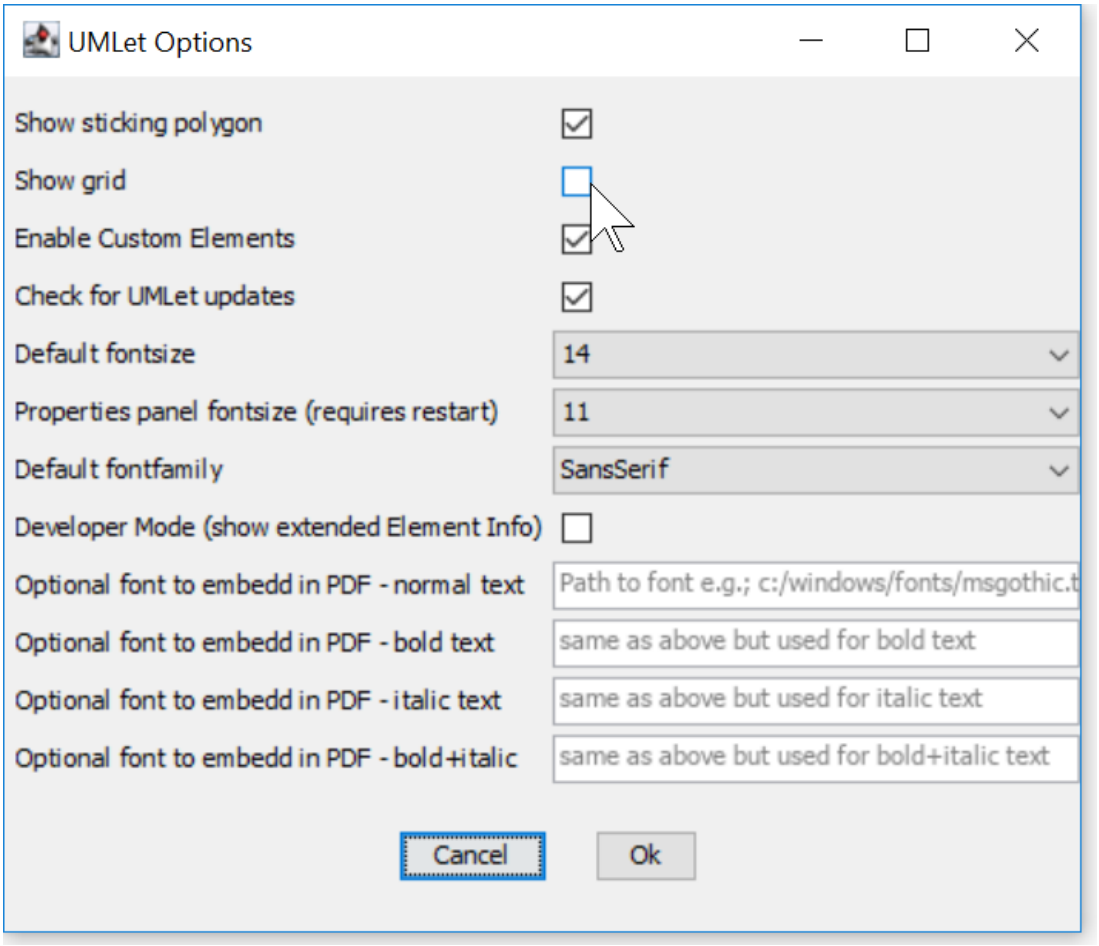
Für abstrakte Methoden müssen Schrägstriche um den Namen herum stehen, damit dieser kursiv dargestellt wird. Beo Klassenmethoden und Klassenvariablen steht unmittelbar vor und nach dem Namen ein Unterstrich.



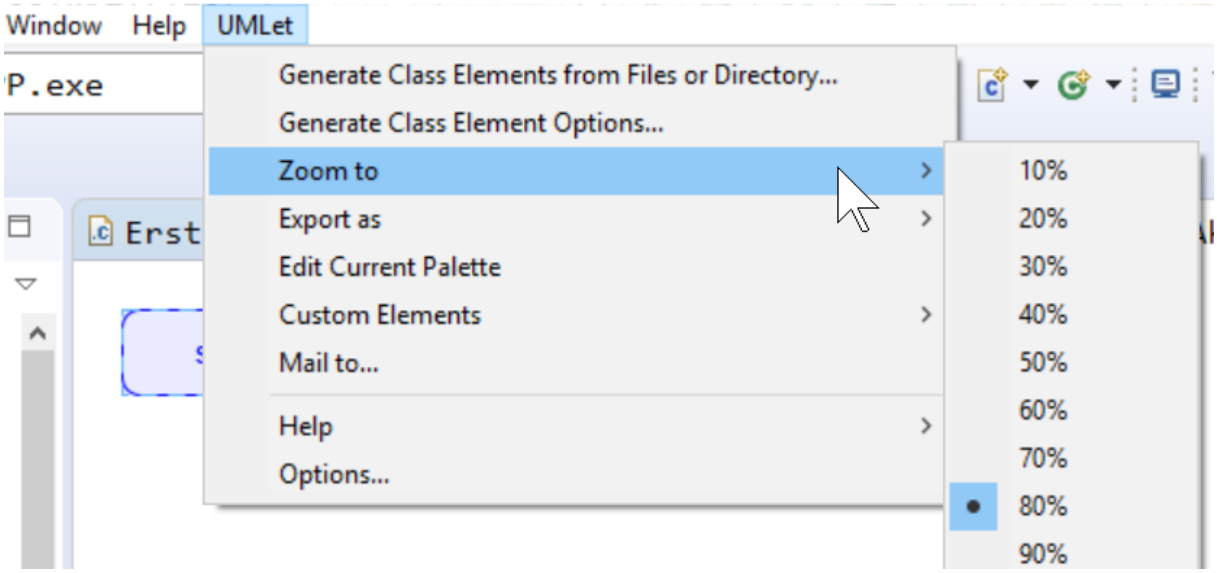
Weiterhin gibt es unter „UMLet > Options“ das Einstellungs-Menü.

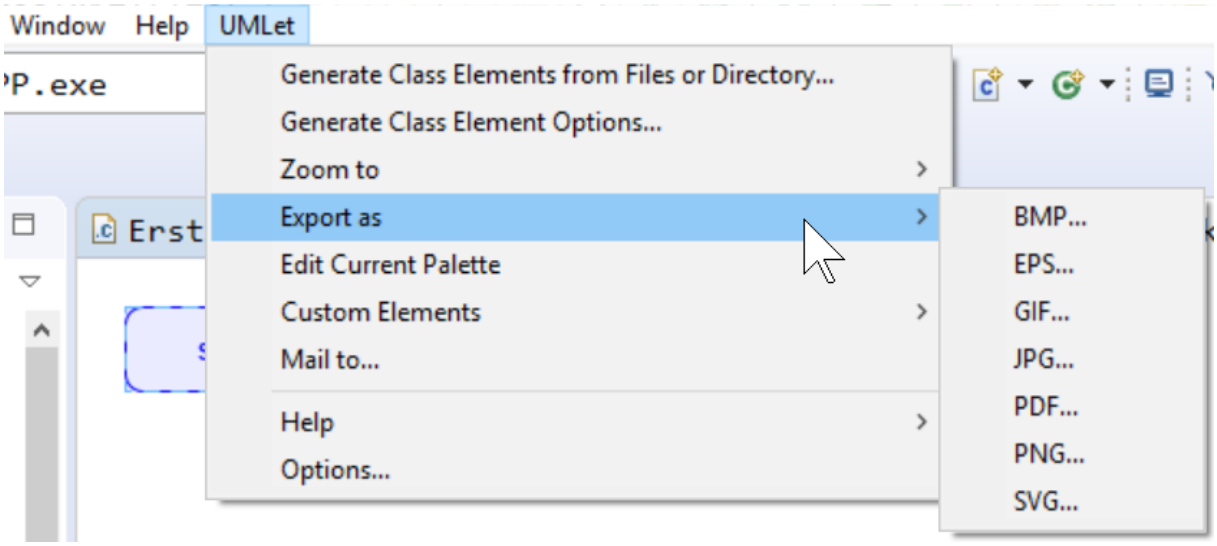


Bei diesem Menü können u. a. Gitterlinien eingeblendet werden, die bei der Ausrichtung von Elementen helfen.



Über die obere Menü-Leiste sind unter UMLet weitere Bearbeitungen möglich, die meist selbsterklärend sind.


Window
Help
UMLet
P.exe
Erst
Generate Class Elements from Files or Directory...
Generate Class Element Options...
Zoom to
Export as
Edit Current Palette
Custom Elements
Mail to...
Help
Options...
10%
20%
30%
40%
50%
60%
70%
80%
90%


Window
Help
UMLet
P.exe
Erst
Generate Class Elements from Files or Directory...
Generate Class Element Options...
Zoom to
Export as
Edit Current Palette
Custom Elements
Mail to...
Help
Options...
BMP...
EPS...
GIF...
JPG...
PDF...
PNG...
SVG...

13 Nutzung von EclipseUML (Omondo) (uralt)

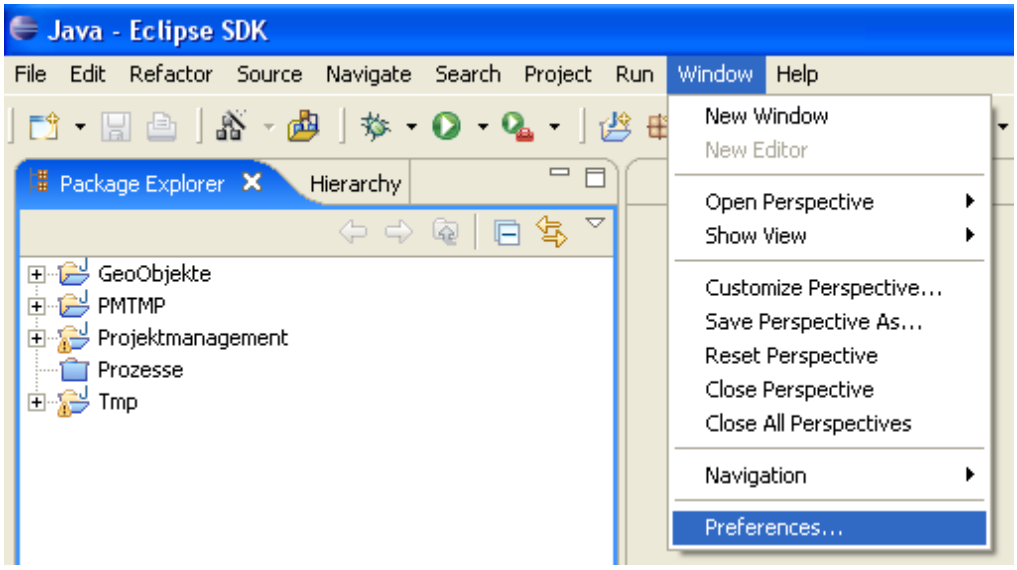
Omondo ist ein UML-Werkzeug, das eng in die Java-Entwicklung eingebunden ist. Hier wird die Omondo-Version genutzt, die zum Lernen zur freien Verfügung steht. Wie für jedes UML-Werkzeug gilt, dass man sich mit der Arbeitsweise des Werkzeugs vertraut machen muss. Da nicht immer alle UML-Notationsmöglichkeiten 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. Eventuell ist eine alte Version unter http://www.uml2.org/eclipse-java-galileo-SR2-win32_eclipseUML2.2_package_may2010.zip erhältlich.

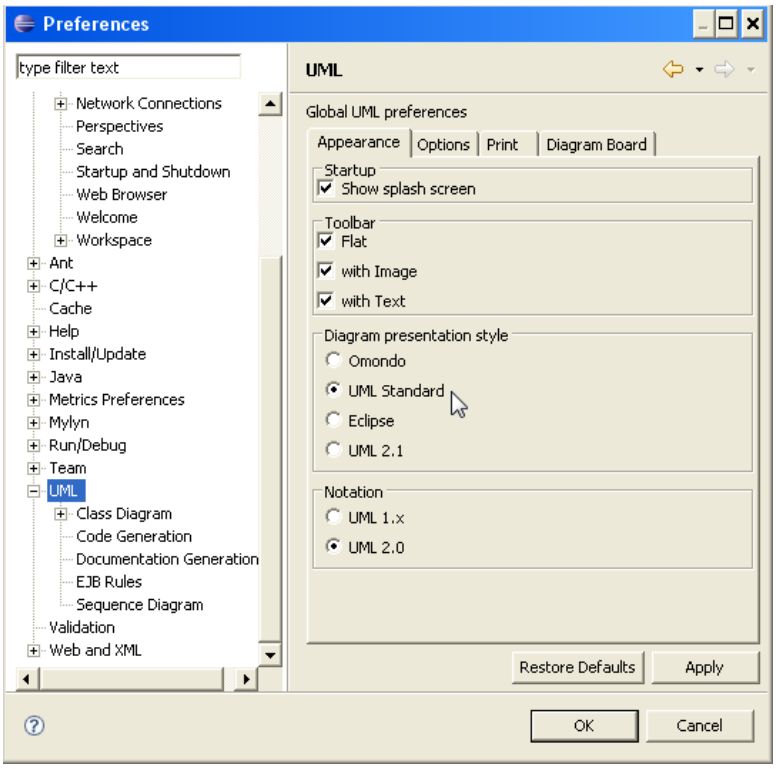
In dieser Notiz wird ein Klassendiagramm mit den drei Klassen Punkt, Linie und Polygon erstellt.

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.

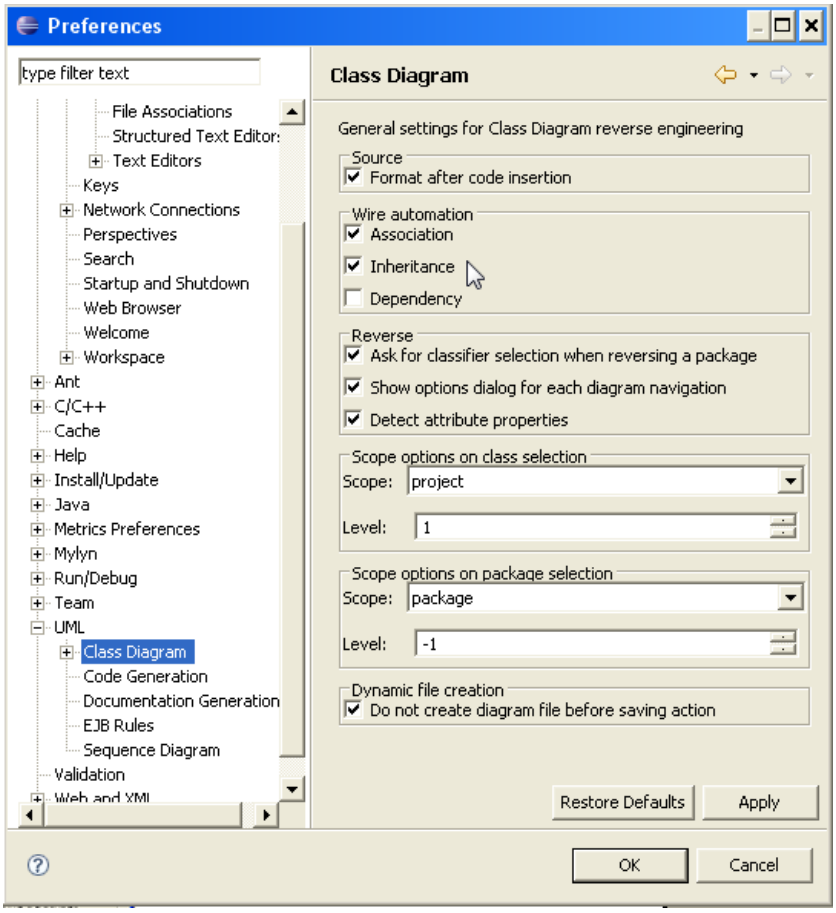
Vor der eigentlichen Nutzung von Omondo sollten einige Eigenschaften eingestellt werden. Wie alle generellen Eclipse-Eigenschaften werden diese unter „Window“ und dann „Preferences“ eingestellt.



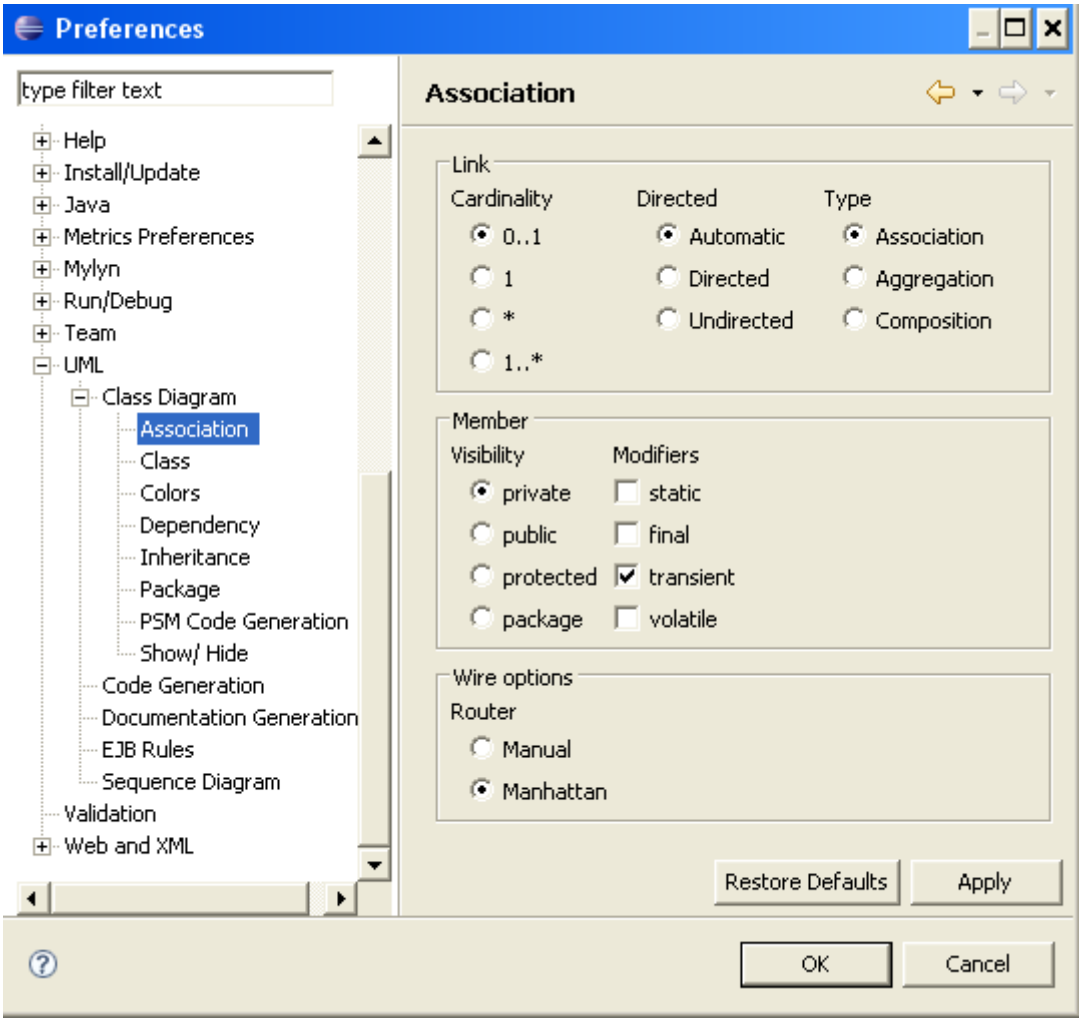
Die Einstellungen zu Omondo finden sich unter dem allgemeinen Punkt „UML“, hier sollten folgende Einstellungen unter dem Reiter „Appearance“ vorgenommen werden. Die Einstellungen bei den anderen Reitern können so gelassen werden.



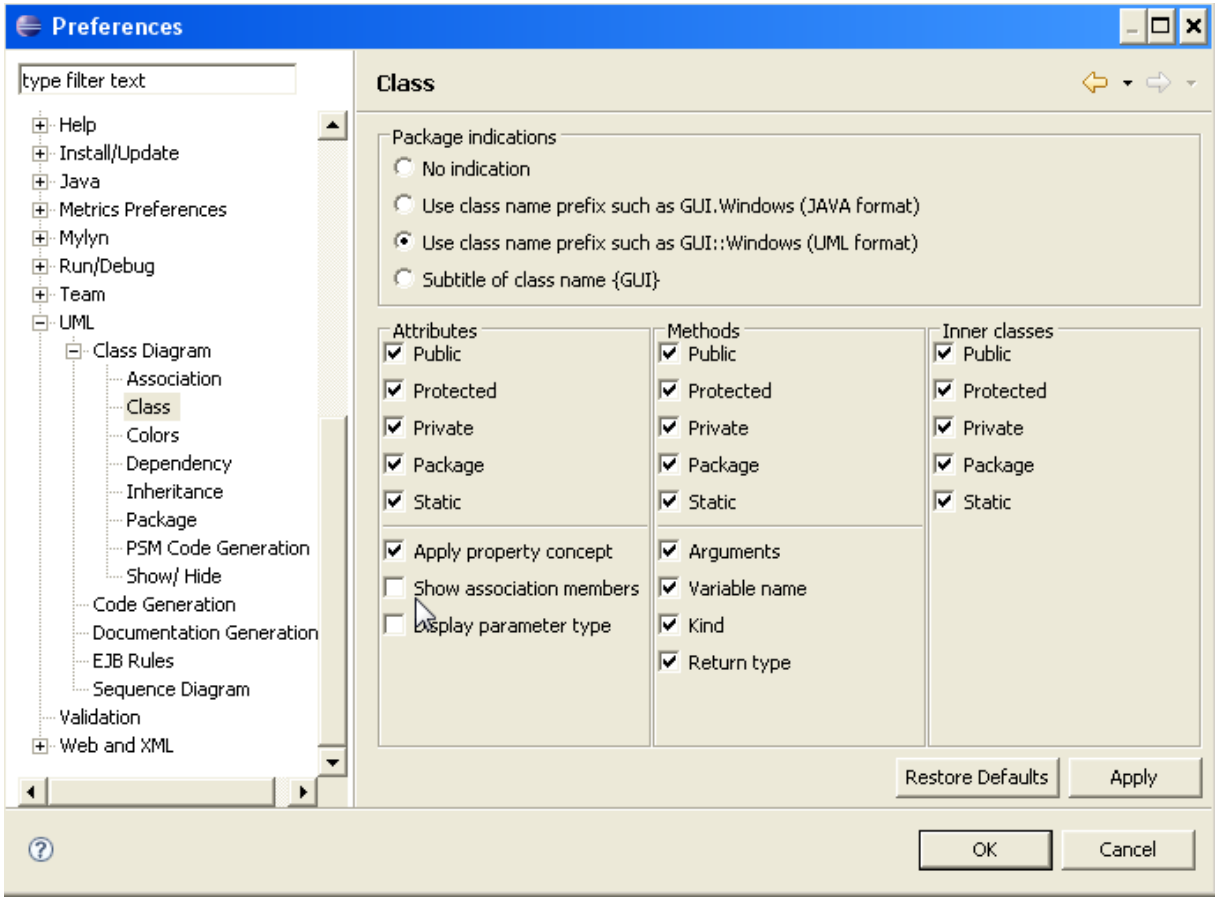
Die Einstellungen zum Unterpunkt “Class Diagramm“ können dem folgenden Bild entnommen werden.



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.

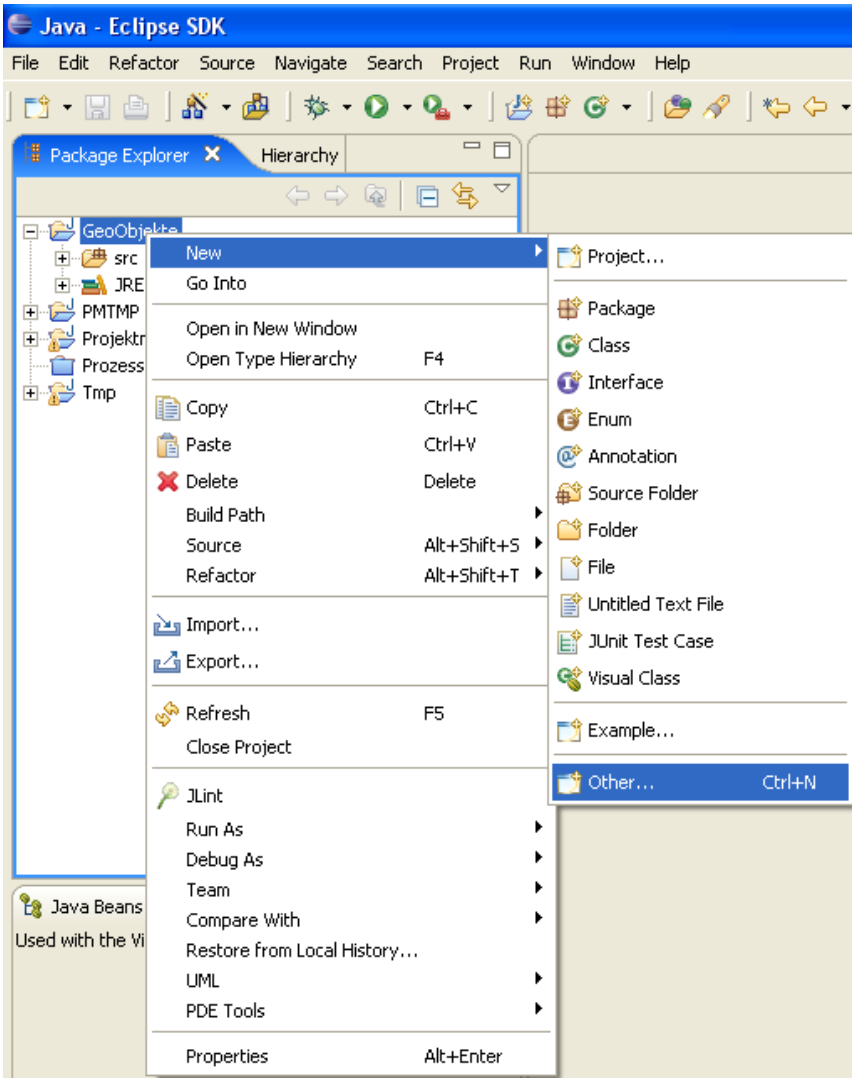


Beim Punkt Class müssen einige Haken gesetzt werden, die später individuell zur Darstellung von Klassen verändert werden können. Das so genannte „property concept“ sorgt dafür, dass get- und set-Methoden nicht explizit im Diagramm erscheinen. Dies hat Vor- und Nachteile abhängig davon, wie das Diagramm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

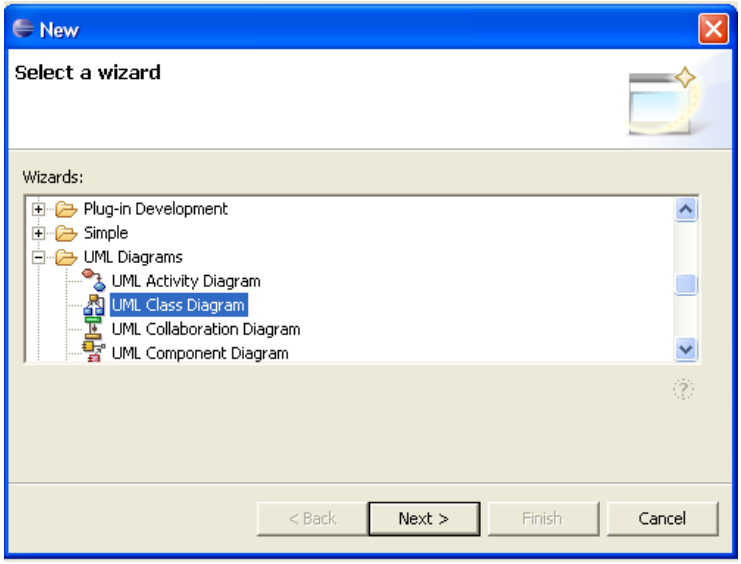


Bei den Punkten Dependency und Inheritance kann wieder das Layout-Verfahren angepasst werden.

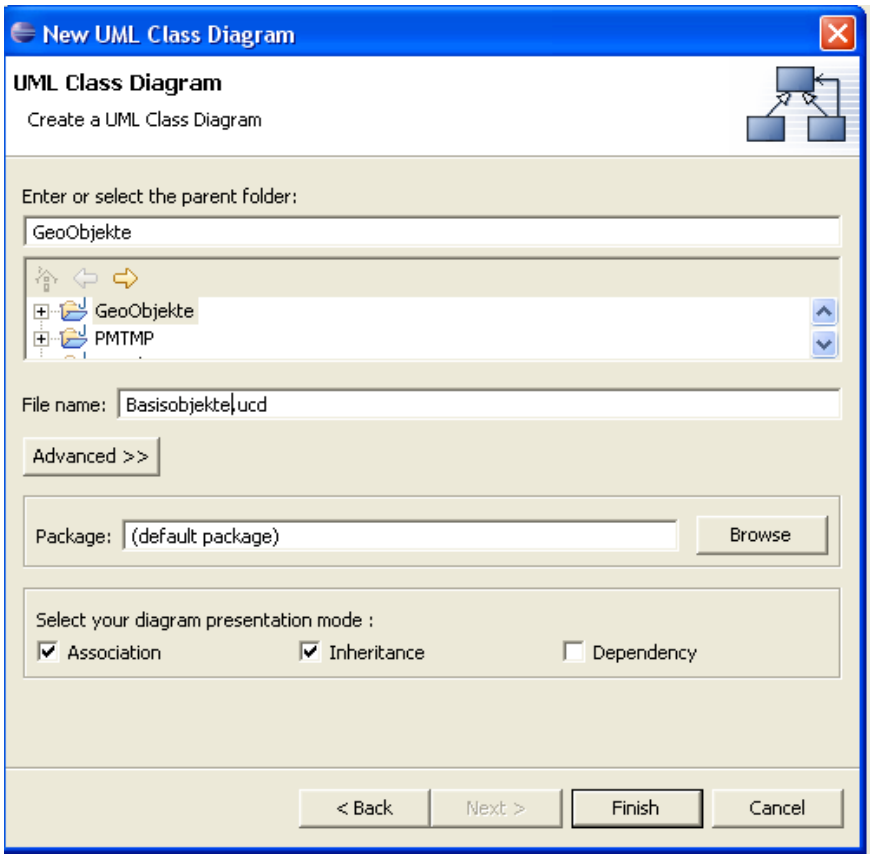
Für die Spezifikation wird im neuen Java-Projekt GeoObjekte ein Klassendiagramm angelegt. Dazu wird nach einem Rechtsklick auf das Projekt im Projekt-Browser „New“ und dann „Other“ ausgewählt.



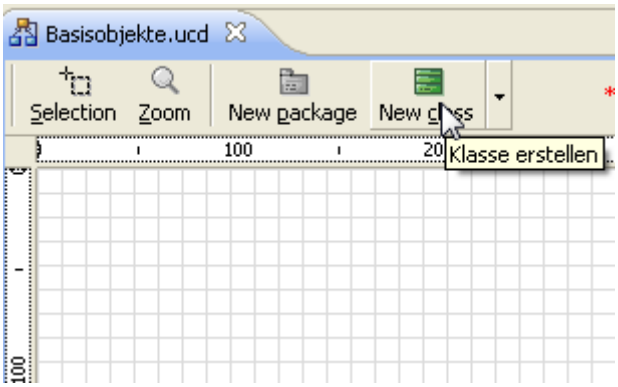
Im aufgehenden Fenster wird der Ordner „UML Diagrams“ geöffnet, „UML Class Diagramm“ ausgewählt und „Next“ gedrückt.



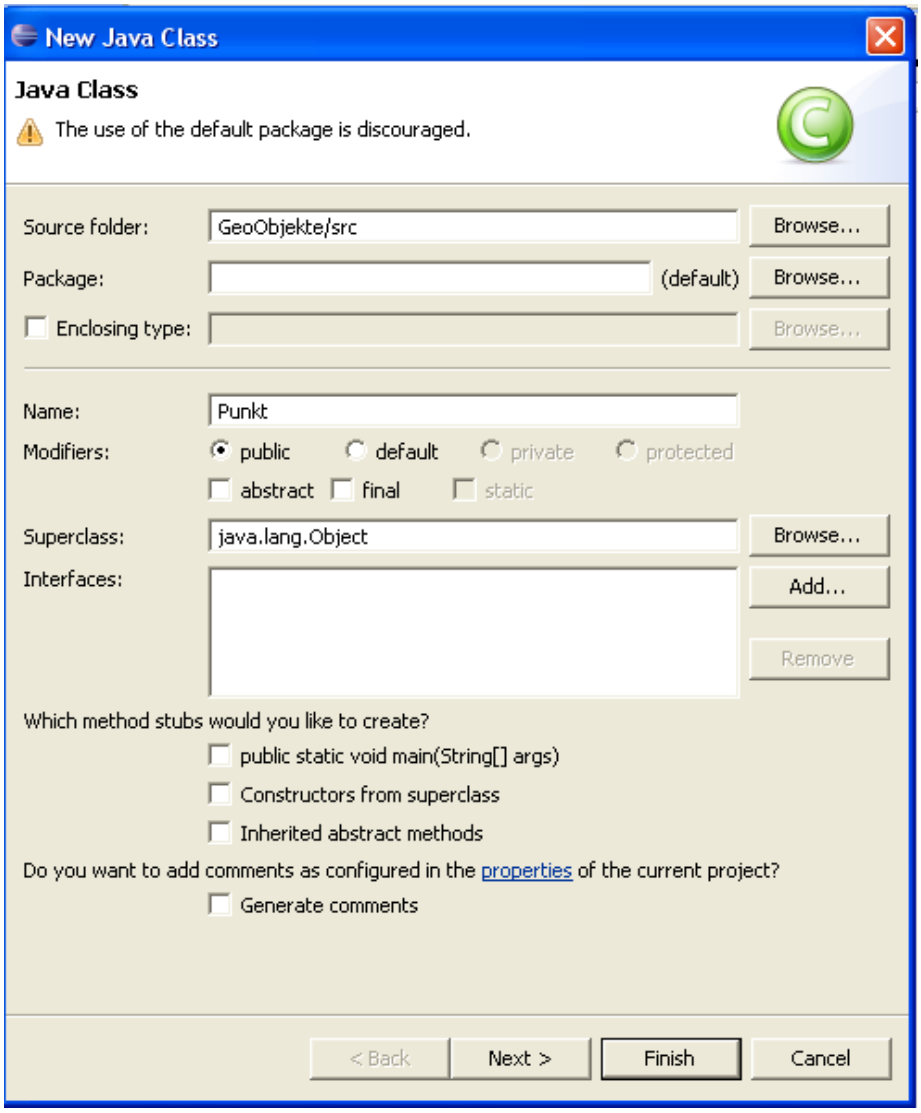
Im aufgehenden Fenster reicht es aus, bei „File Name“ einen sinnvollen Dateinamen einzugeben und die Haken in der letzten Zeile zu prüfen. Grundsätzlich besteht die Möglichkeit, dass Diagramme zu verschiedenen Paketen in den zugehörigen Ordnern entwickelt werden. Diese Möglichkeit sollen erfahrene Nutzer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



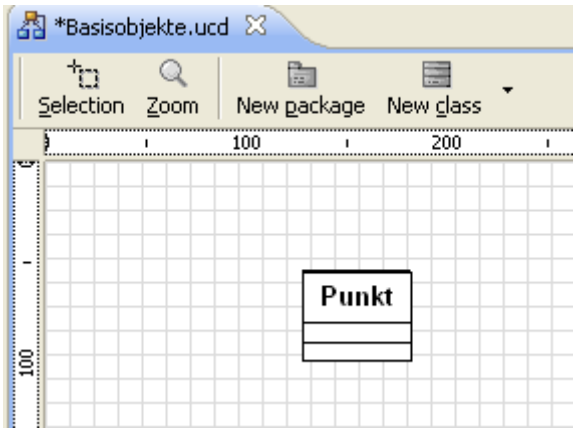
Die grundsätzliche Bedienung des Werkzeugs sieht für die Erstellung neuer Elemente so aus, dass die Art des neuen Elements in der Kopfzeile angeklickt wird und dass man dann durch einen Klick mit der Maus in der Arbeitsfläche das neue Element platziert. Für dieses Element wird dann ein Dialog zur Eingabe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



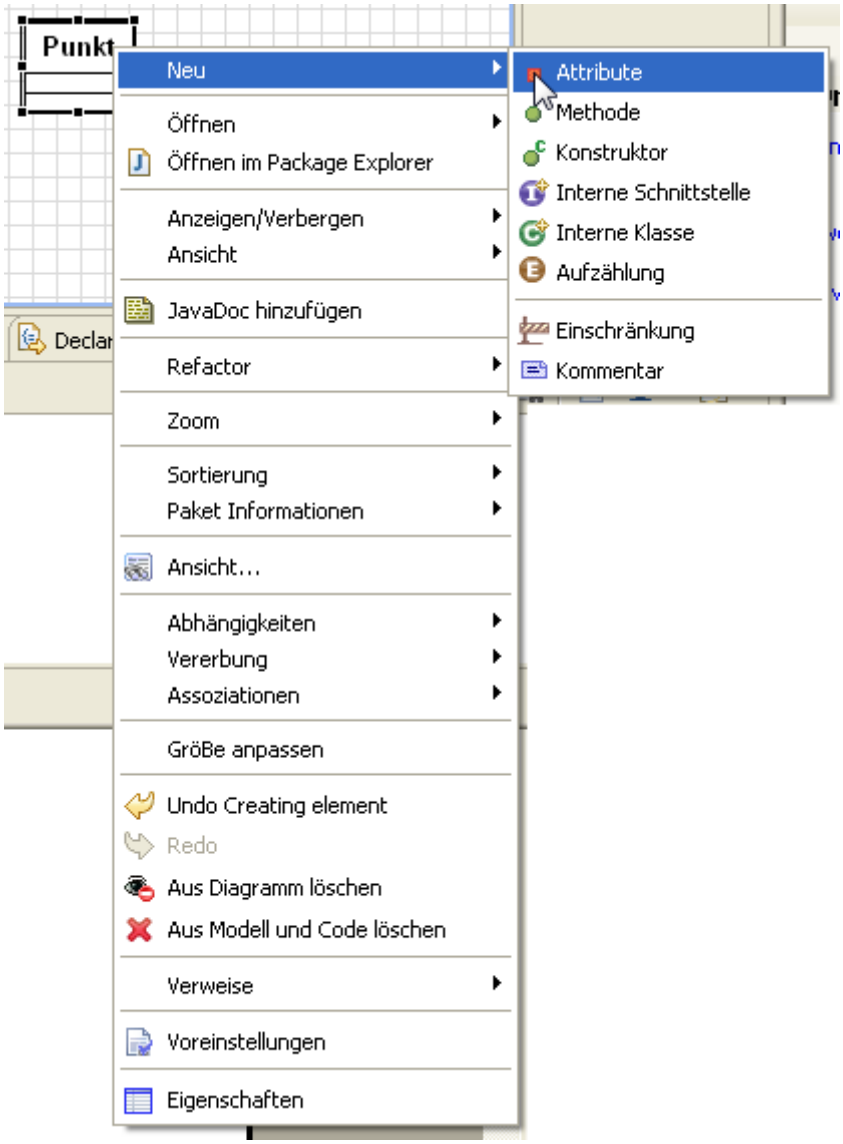
Danach erscheint der Eclipse-Standarddialog zur Erstellung einer Java-Klasse. Im Beispiel wird nur ein Klassenname eingegeben. Erfahrenere Java-Programmierer werden die Paketstruktur nutzen. Die Eingabe wird mit „Finish“ abgeschlossen.



Man erhält eine einfache Klasse im Klassendiagramm.

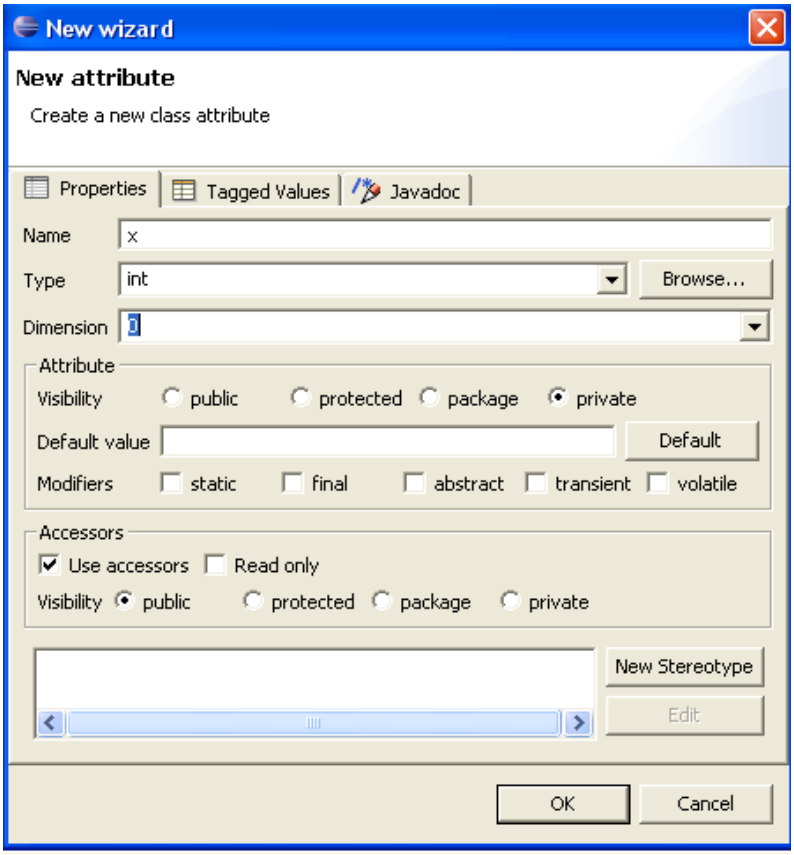


Durch einen Rechtsklick auf die Klasse werden die vielfältigen Bearbeitungsmöglichkeiten für Klassen in Omondo angezeigt, die gerade bei der Analyse komplexer Systeme, siehe z. B. „Dependencies“ (oder „Abhängigkeiten“ abhängig von der installierten Variante), sehr hilfreich sein können. Im konkreten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

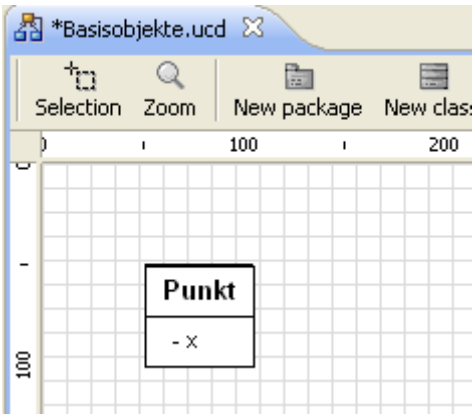


Im aufgehenden Fenster muss der Name und der Typ (Type) der Exemplarvariable angegeben werden. Im Abschnitt Accessors wird eingestellt, welche get- und set-Methoden automatisch erzeugt werden sollen. Mit „Use accessors“ wird eingestellt, dass es get- oder/und set-Methoden geben soll. Mit „Read only“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

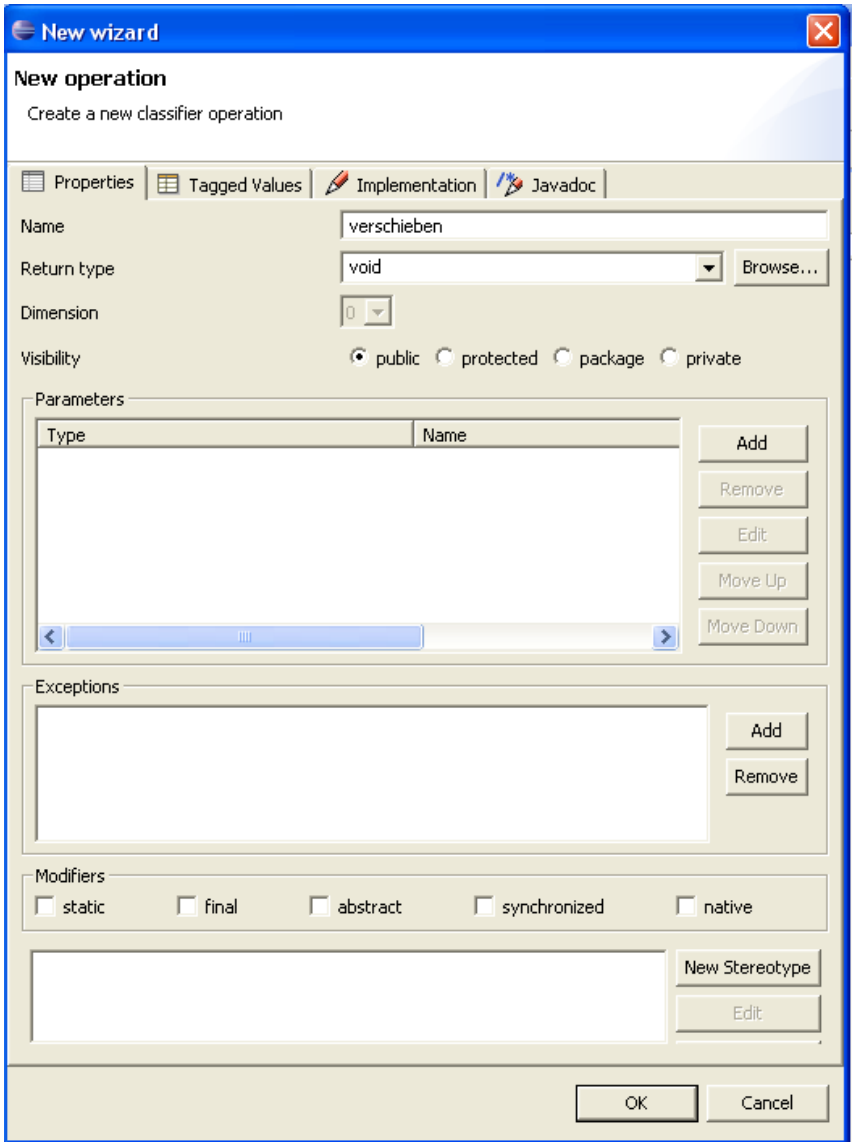
Mit Dimension wird festgelegt,ob es sich um einen Array handeln soll, dabei steht der Wert eins für einen eindimensionalen Array x[] und ein Wert drei für einen dreidimensionalen Array x[][][]. Die weiteren Einstellungsmöglichkeiten sollten durch die Java-Vorlesung inhaltlich bekannt sein. Unter dem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



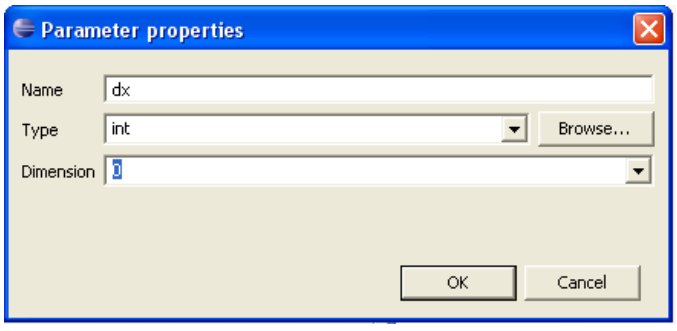
Im Klassendiagramm wird die neue Exemplarvariable sichtbar, wobei auf die Angabe von generierten get- und set-Methoden verzichtet wird. (Nimmt man Java-Quellcode und lässt daraus ein Klassendiagramm erstellen, muss dies leider nicht der Fall sein, weiterhin sollte der Typ sichtbar sein, ist es aber anscheinend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



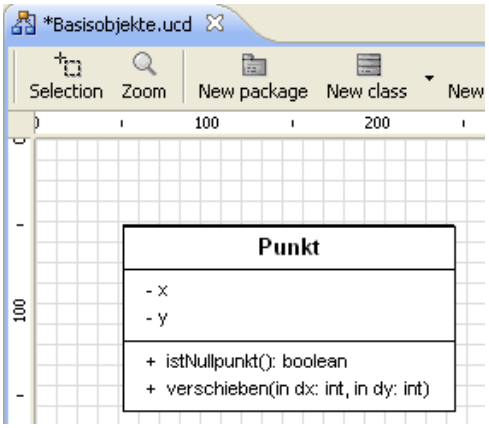
Im nächsten Schritt wird eine neue Methode ergänzt. Dazu wird im Bearbeitungsmenü „Neu“ und dann „Methode“ ausgewählt. Im dann aufgehenden Fenster muss der Methodenname (Name) und der Rückgabetyp (Return type) angegeben werden.



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“.

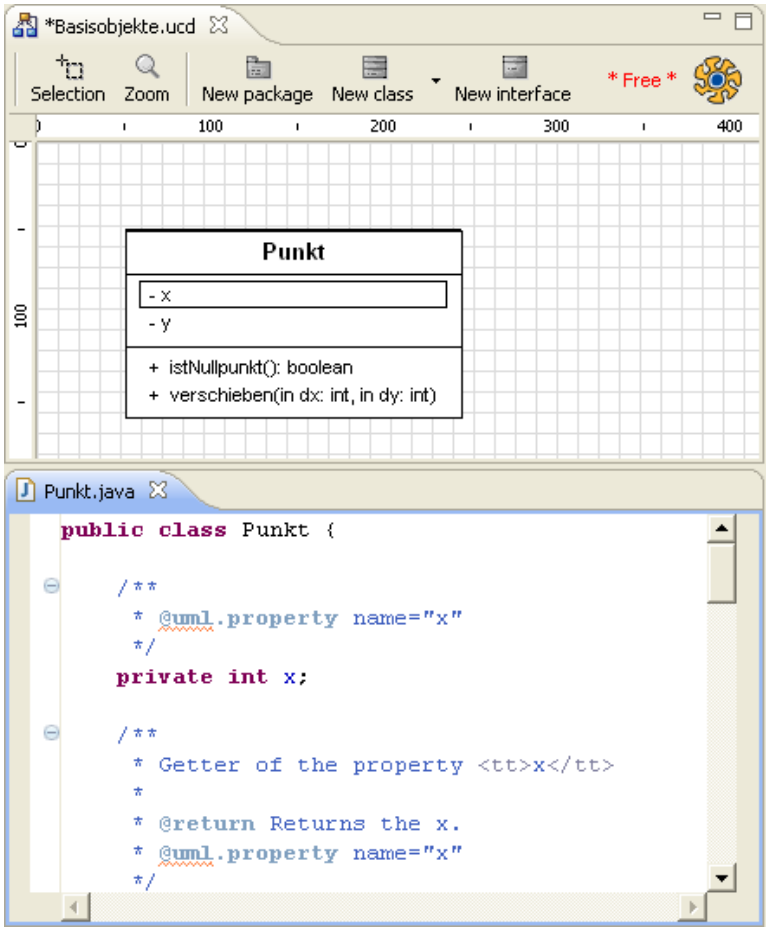


Nachdem alle Parameter und weitere Methodeneigenschaften eingegeben wurden, wird die Methodenerstellung mit „Ok“ abgeschlossen. Im folgenden Klassendiagramm wurde eine weitere Methode ergänzt.



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.



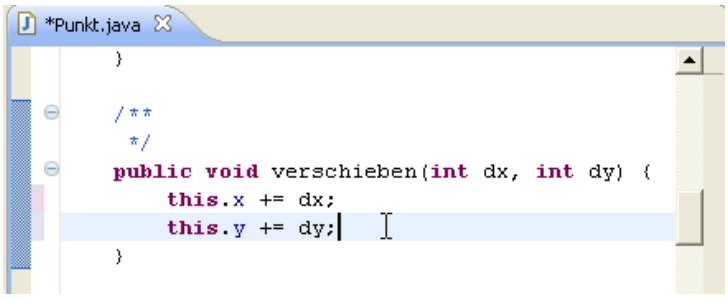
Generell gilt, dass Omondo das Programmgerüst erstellt und der Entwickler dann in den vorgesehenen Abschnitten den Programmcode ergänzt. Der von Omondo im Beispiel generierte Programmcode sieht wie folgt aus:

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

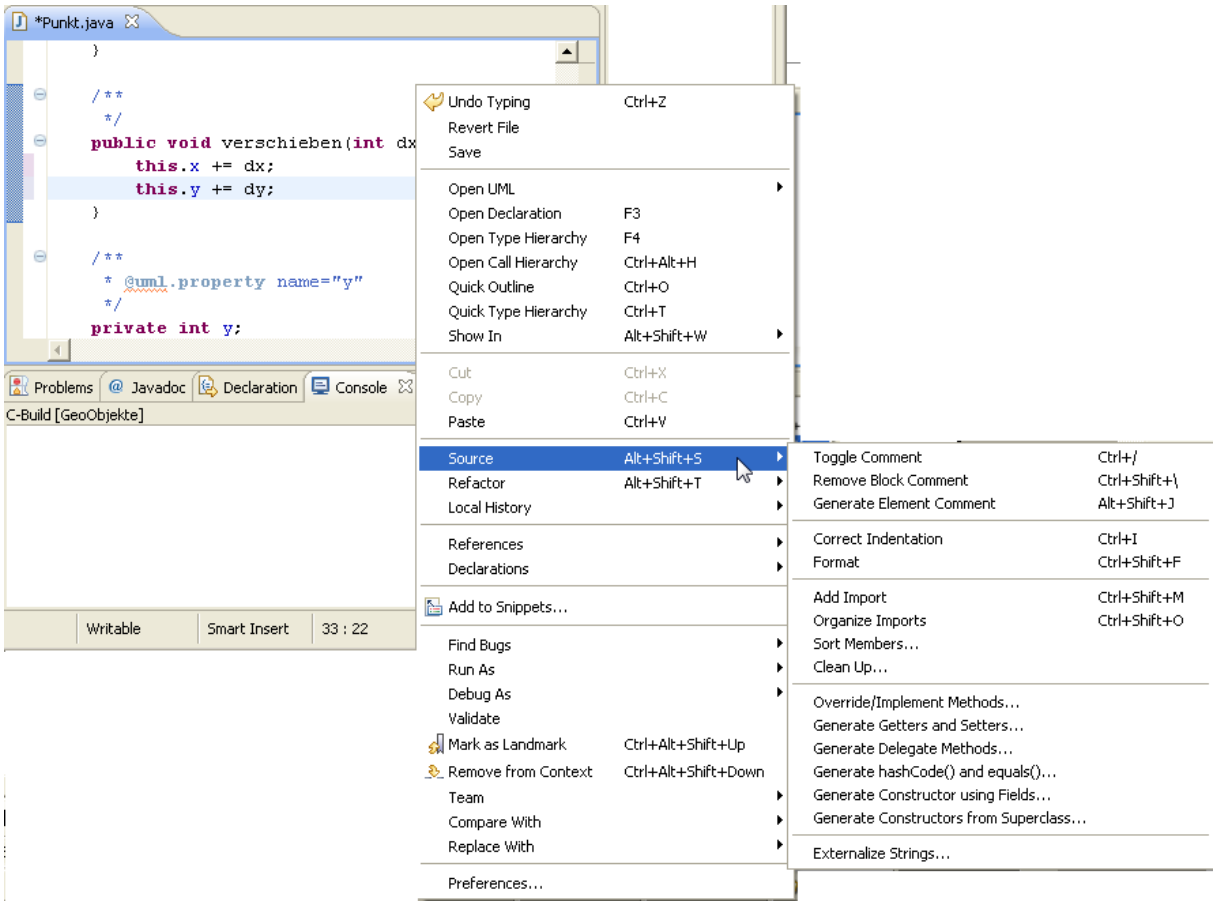
```
 */
private int x;

/**
 * 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.

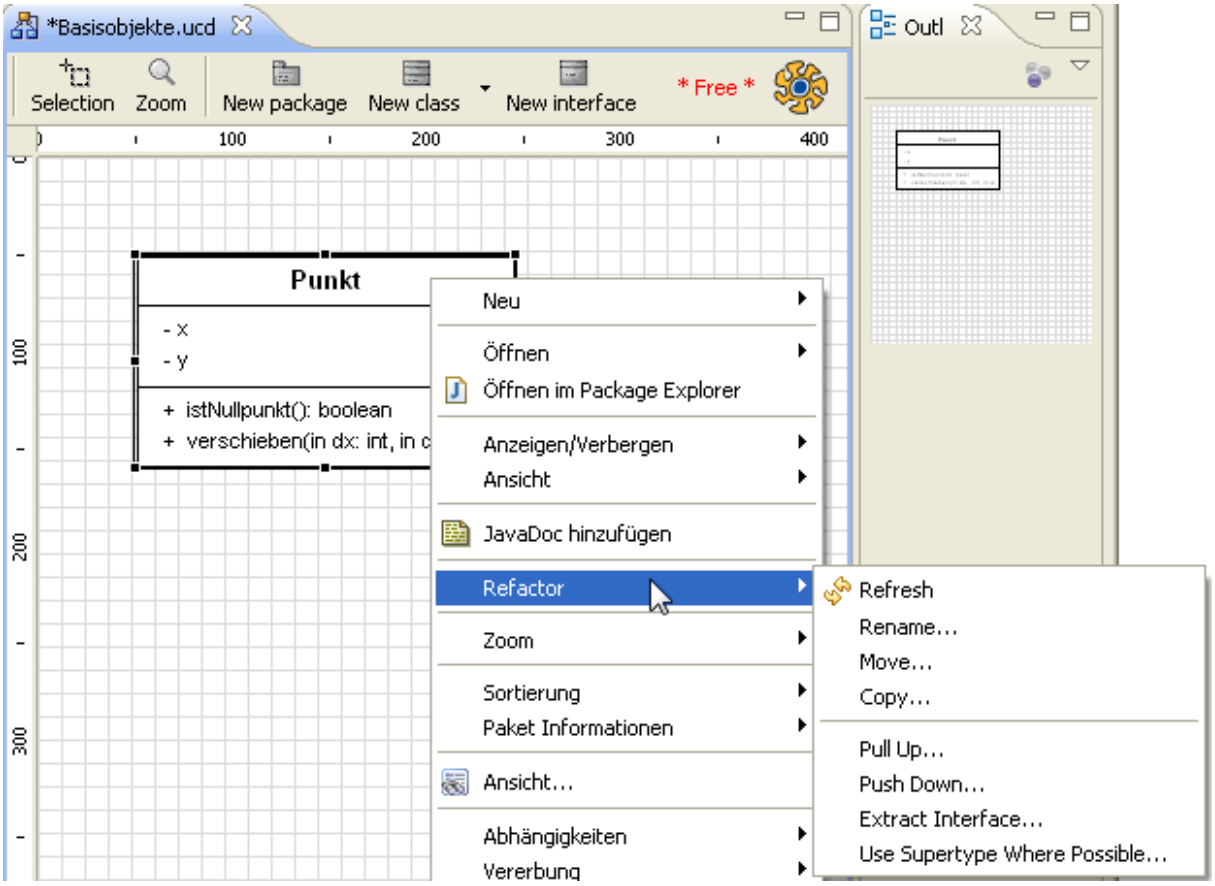


Da Omondo ab und zu einige unglückliche Einrückungen macht, kann auch die Code-Formatierung von Eclipse genutzt werden. Dazu wird im Editor ein Rechtsklick ausgeführt, dann „Source“ und „Format“ ausgewählt. Für Eclipse-Neulinge sei auf die vielfältigen Möglichkeiten hingewiesen, die noch in dem „Source“-Menü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

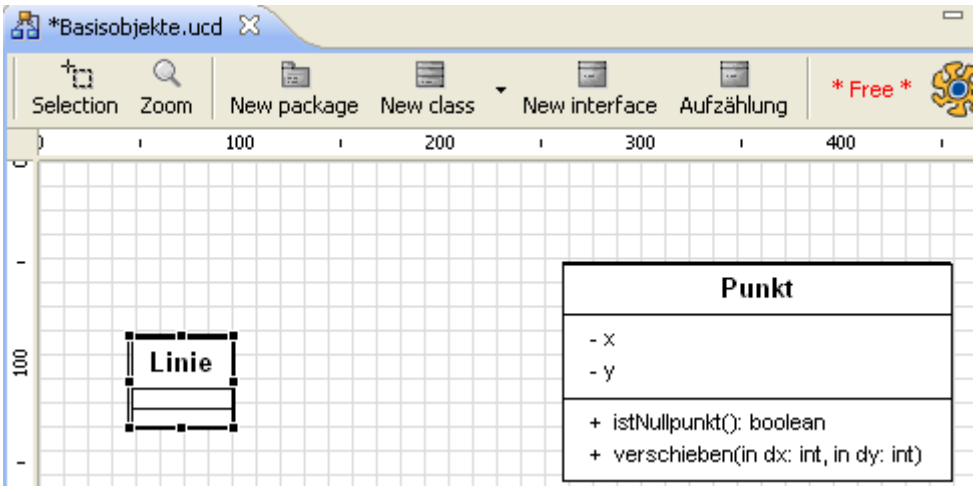


Möchte man die Eigenschaft einer Klasse, einer Exemplarvariable oder einer Methode ändern, so wird diese im Klassendiagramm ausgewählt und ein Menü über Rechtsklick geöffnet. In

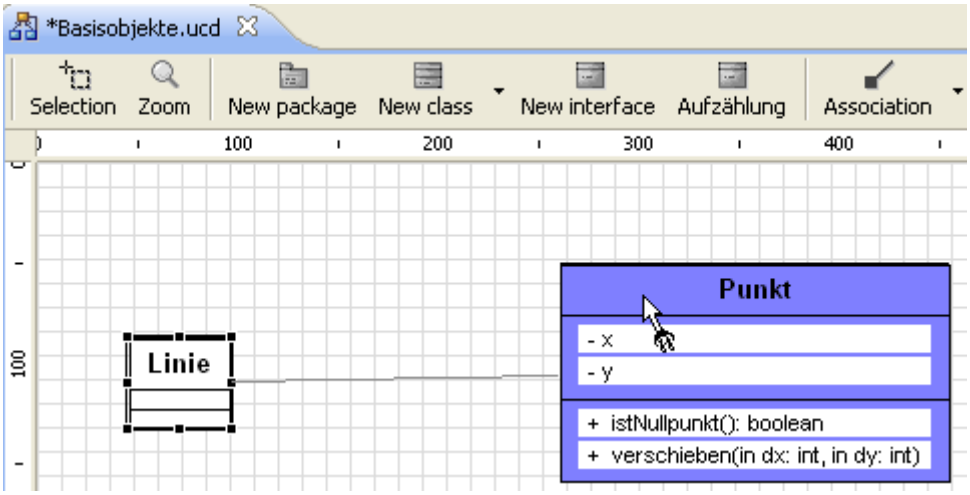
diesen individuellen Menüs gibt es immer einen Punkt „Refactor“, der, genau wie generell in Eclipse, die Änderung von Eigenschaften ermöglicht.



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.)

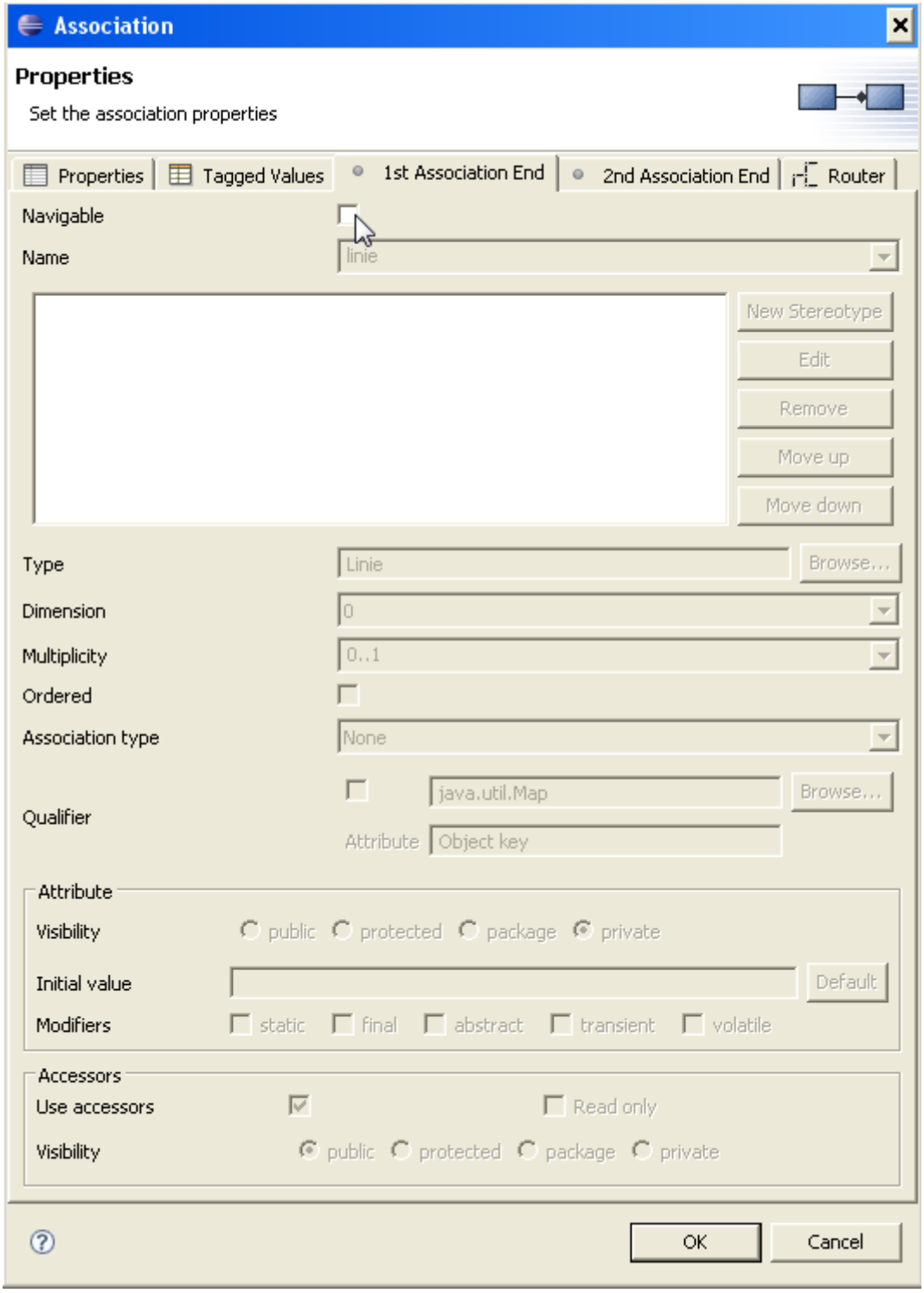


Aus diesem Grund wird der Startpunkt als Assoziation eingetragen. Dazu wird oben „Assoziation“ ausgewählt und dann auf die Startklasse, genauer den Klassennamen und kein anderes Teil der ausgehenden Klasse Linie und dann auf die Zielklasse Punkt, genauer den Klassennamen, geklickt.

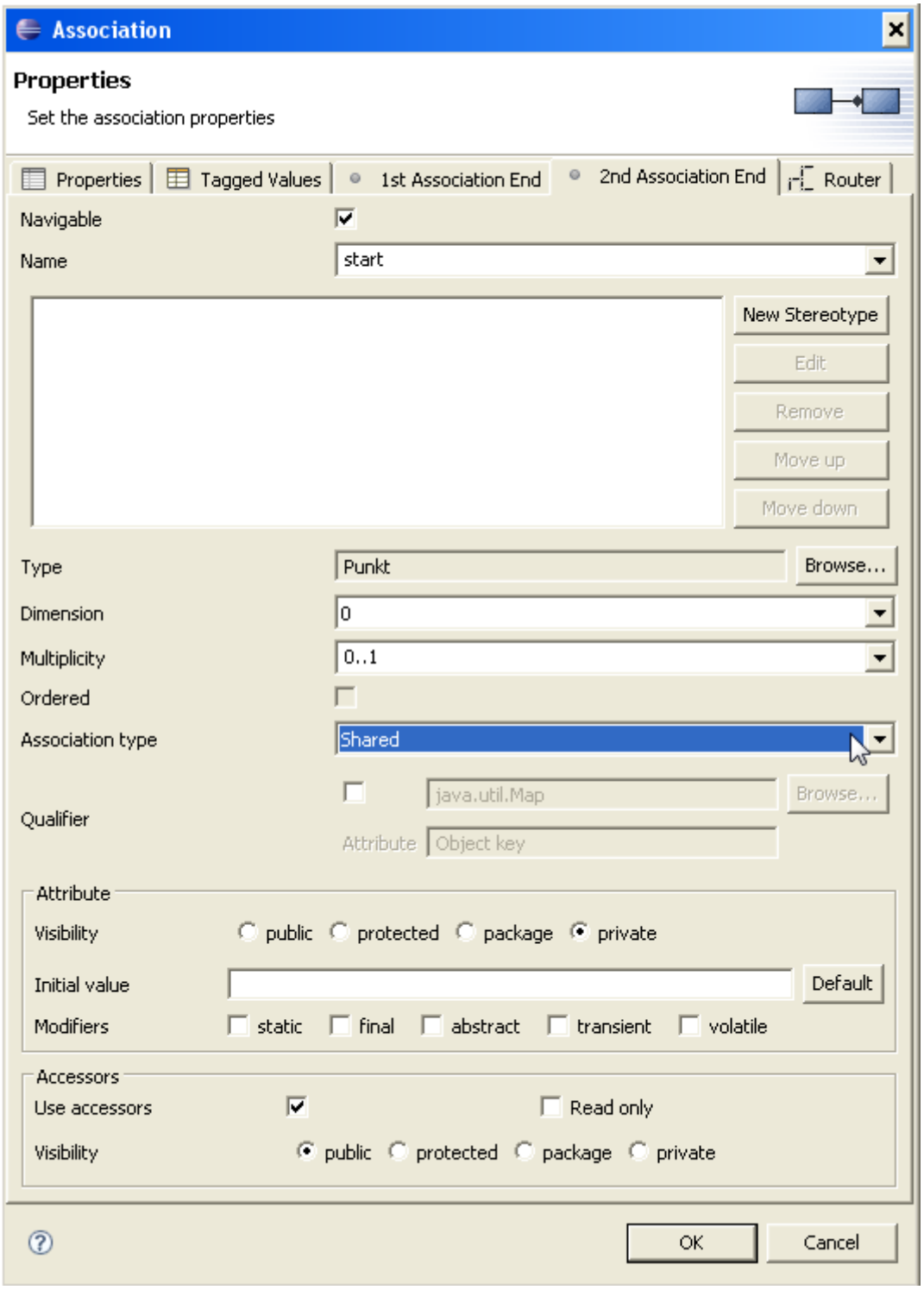


Es öffnet sich ein Fenster, in dem die Eigenschaften der Assoziation eingegeben werden sollen. Wie für Java typisch, soll die Linie eine Referenz auf den Punkt erhalten, wobei der Punkt nicht ausschließlich zur Linie gehören muss. Man spricht in der UML von einer Aggregation.

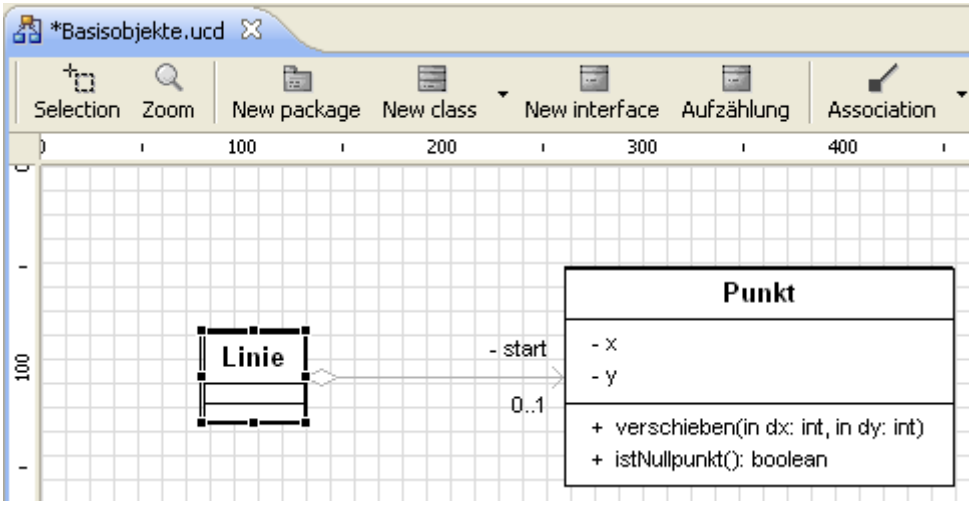
Interessant ist zunächst der Reiter „1st Association End“, dabei geht es um die Rolle der Linie in der Assoziation aus Sicht des Punktes. Da der Punkt nicht wissen soll, zu welcher Linie er gehört, wird einfach der Haken „Navigable“ weggenommen, wodurch hier keine weiteren Eingaben mehr möglich sind.



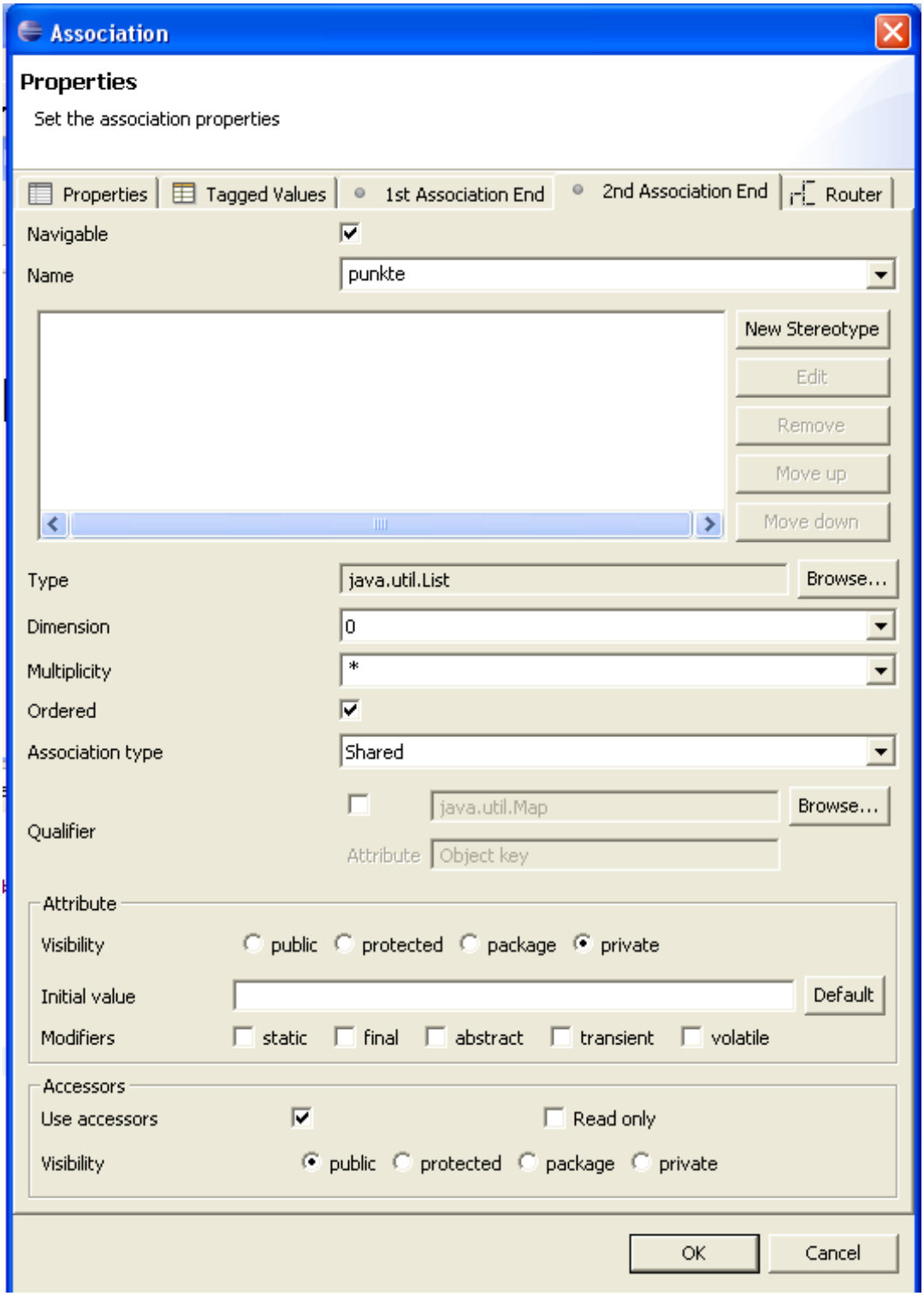
Beim „2nd Association End“ wird die Rolle des Punktes aus Sicht der Linie eingetragen. Dazu wird zunächst der Name der Exemplarvariablen start eingegeben. Die „Multiplicity“ für die Kardinalität kann im Beispiel so gelassen werden. Der „Association type“ wird auf „Shared“ gesetzt, was in der UML einer 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.



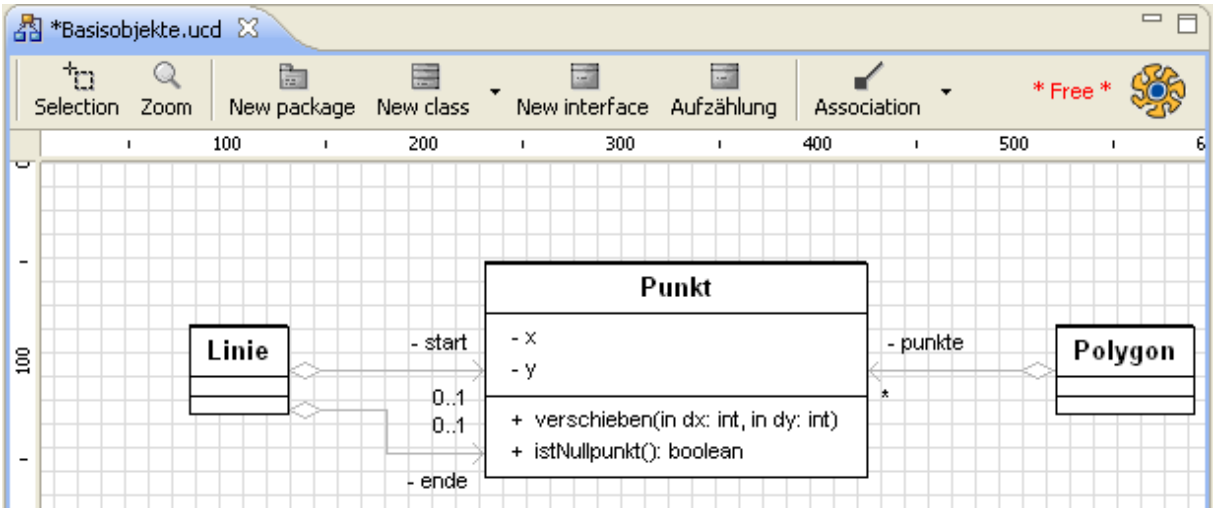
Nachdem die Eingabe mit „Ok“ abgeschlossen wurde, ist die Assoziation im Diagramm sichtbar.



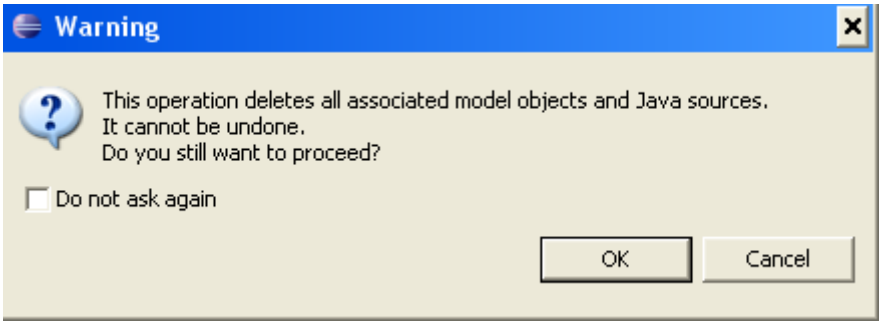
Im nächsten Beispiel wird eine Klasse Polygon angelegt, wobei ein Polygon von mehreren Punkten begrenzt wird. Grundsätzlich ist das Vorgehen identisch mit der vorherigen Eingabe einer Assoziation, nur einige Details müssen geändert werden. Dabei wird die „Multiplicity“ auf „*“ gesetzt. Man beachte, dass 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.



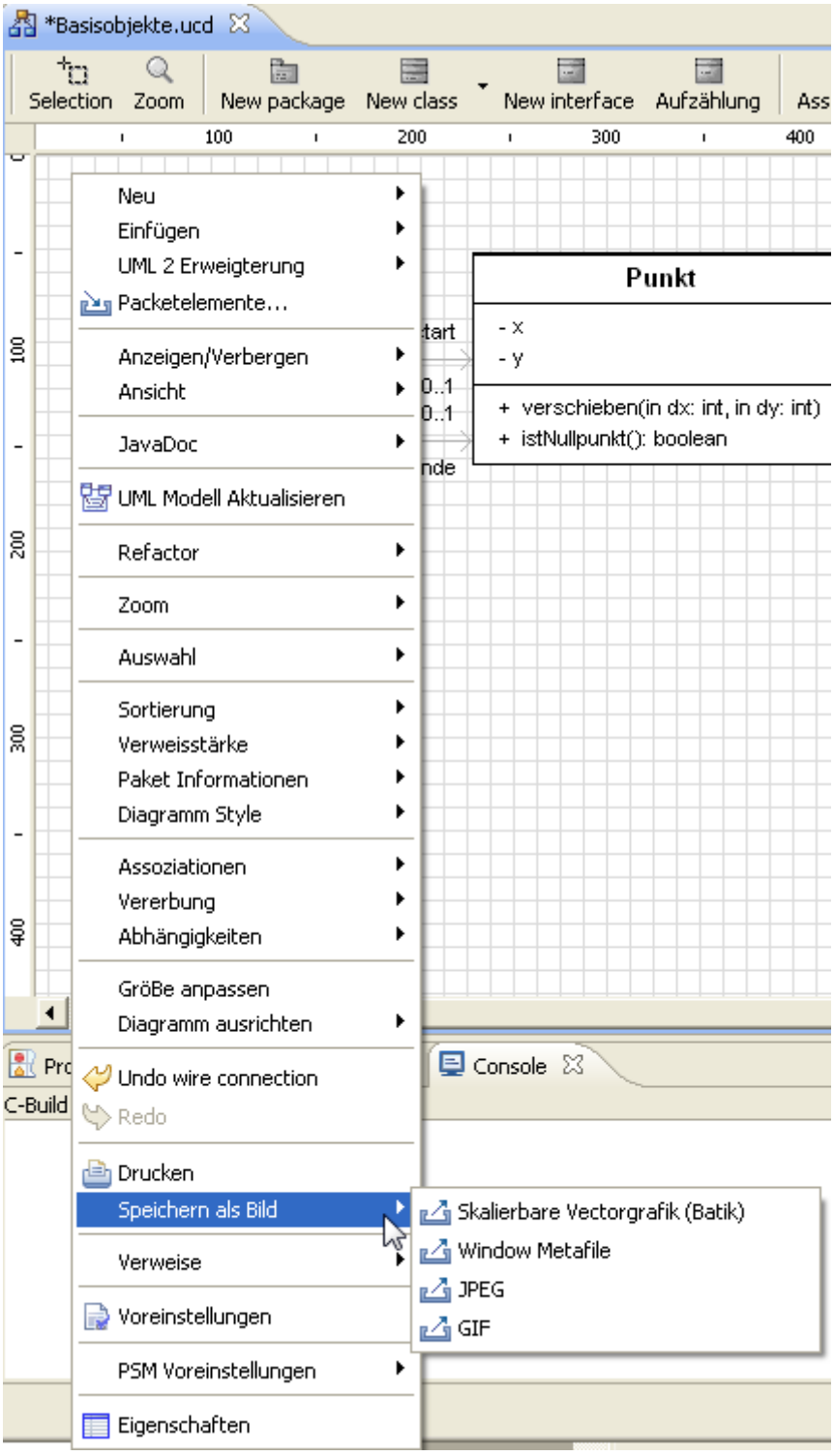
Das resultierende Diagramm sieht wie folgt aus.



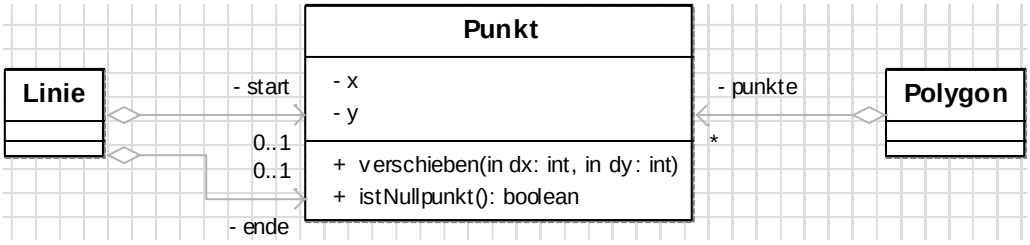
Warnung: Man kann im Diagramm Elemente löschen. Da das Diagramm mit dem Java-Code verknüpft ist, bedeutet dies, dass damit auch der Java-Code gelöscht wird. Will man Klassen in Klassendiagrammen nicht mehr sehen, so kann im Menü „Hide“ gewählt werden.



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.



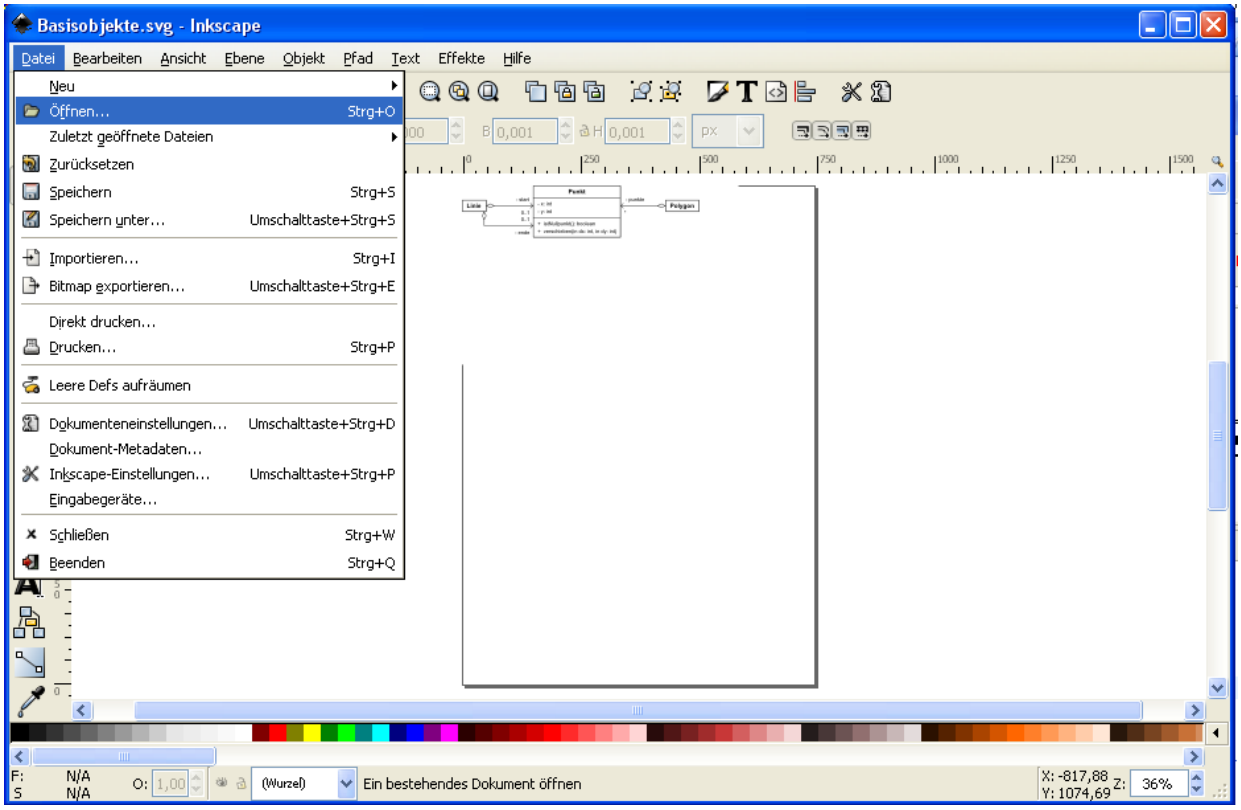
Die Typen WMF, JPG und GIF, die z. B. in Präsentationen genutzt werden können, haben allerdings das Karo-Raster im Hintergrund, wie folgendes WMF-Bild zeigt.



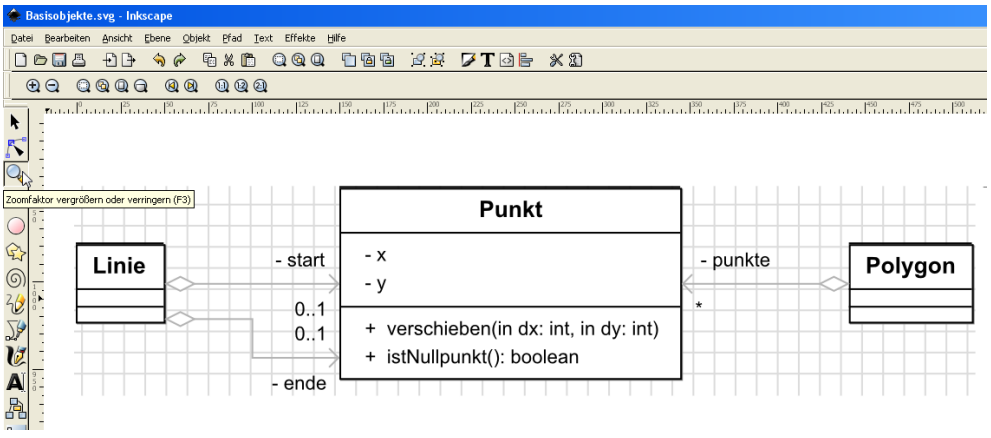
Alternativ steht „Scalable Vector Graphics (Batik)“ zur Verfügung.

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.

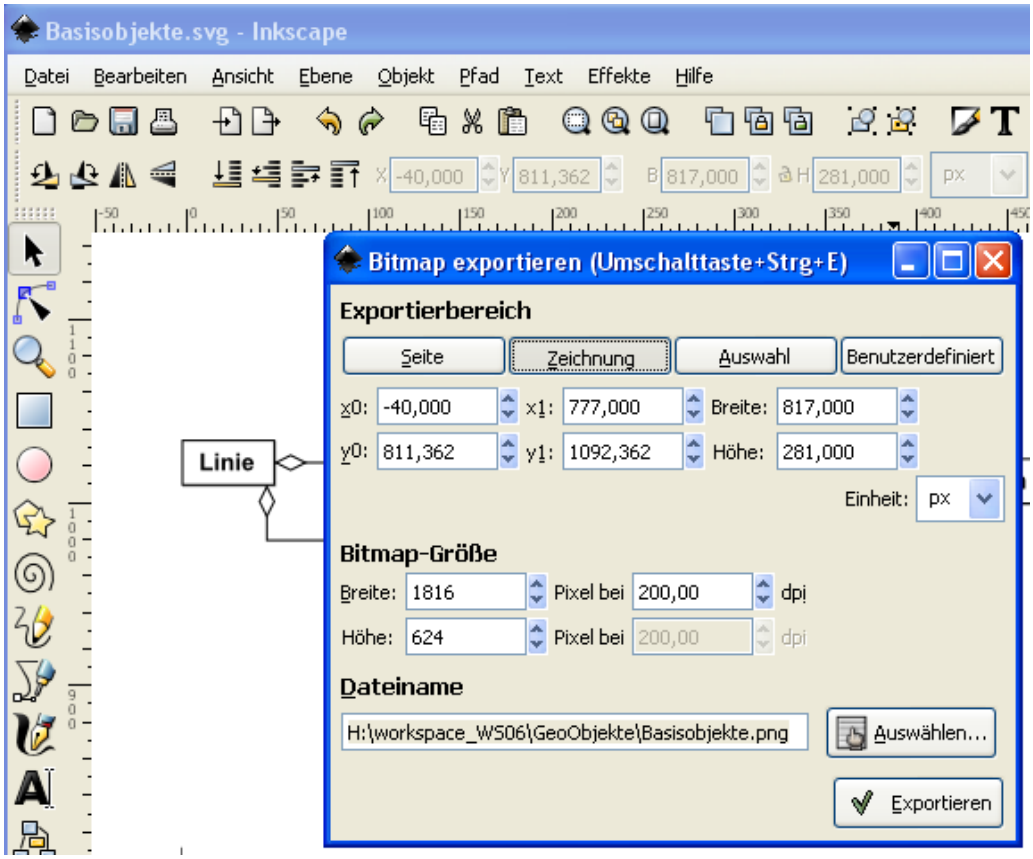
Nach dem Einlesen der svg-Datei über „Datei“ und „Öffnen...“ steht das Bild zur Verfügung.



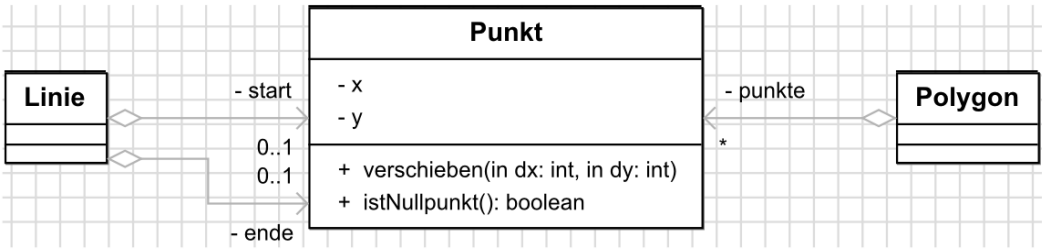
Leider gehört auch hier der Hintergrund zur Datei.



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.

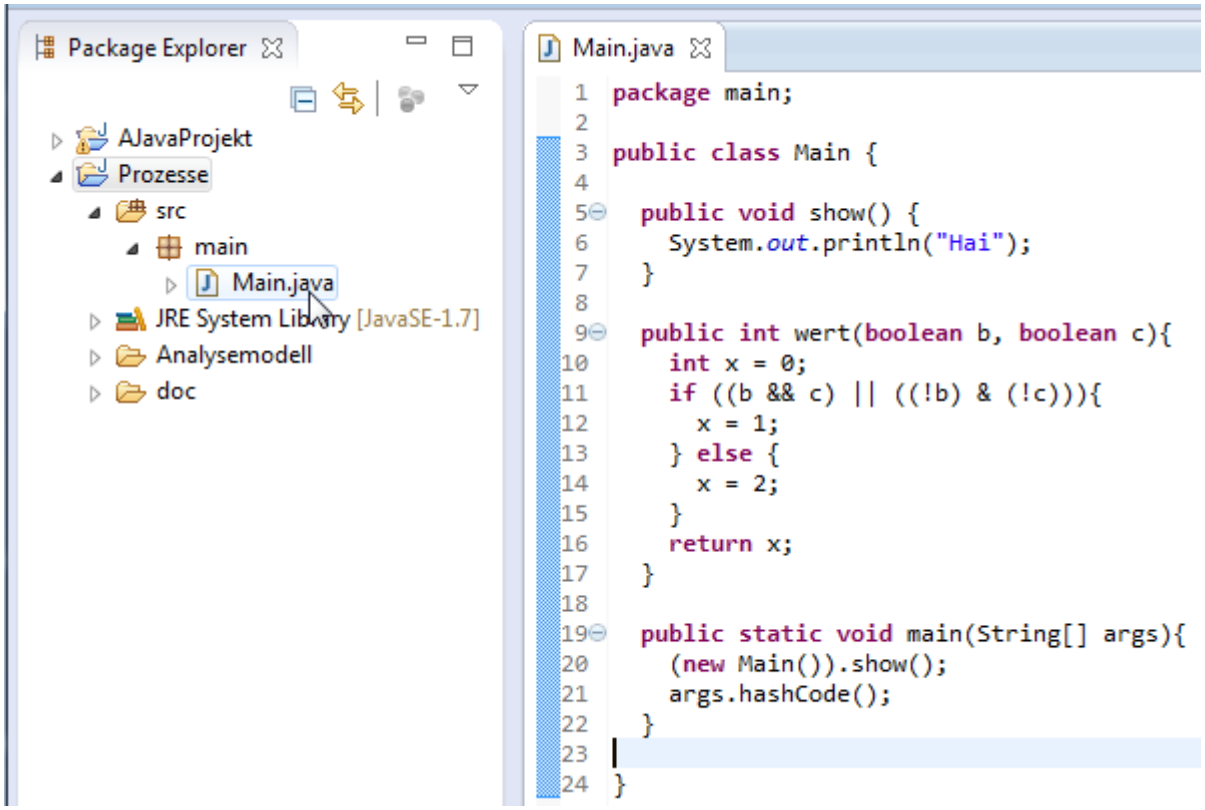


14 JUnit und Testüberdeckung (4.11.0)

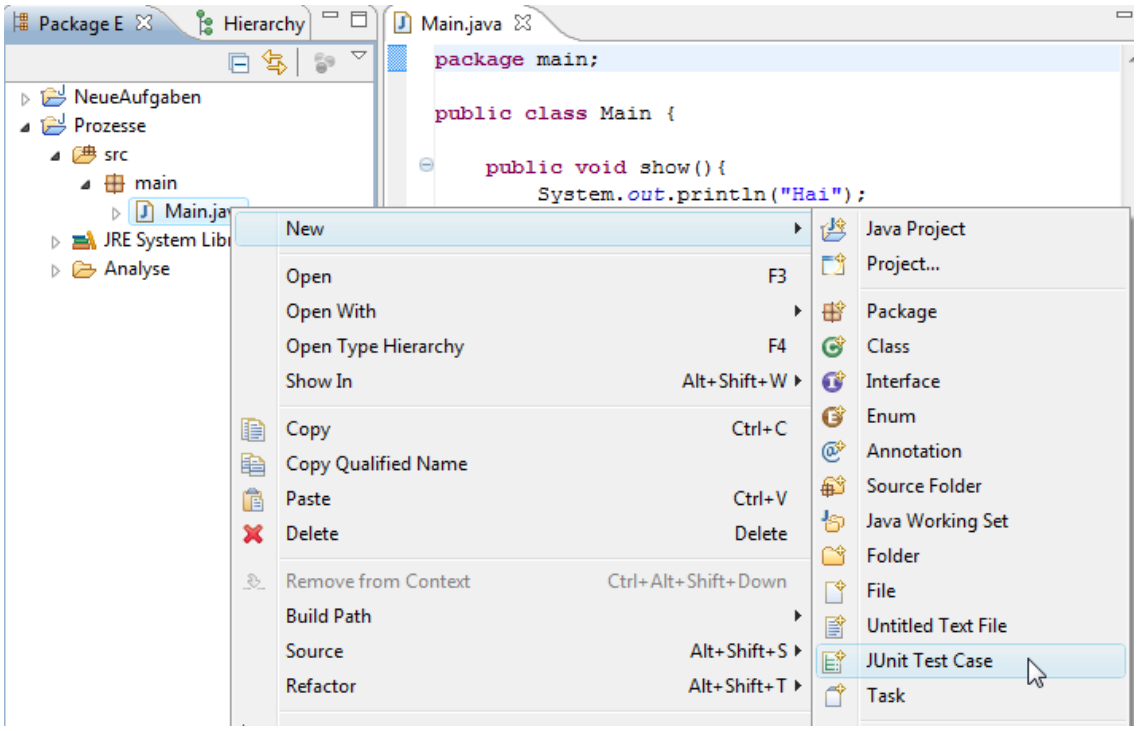
14.1 JUnit-Nutzung

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

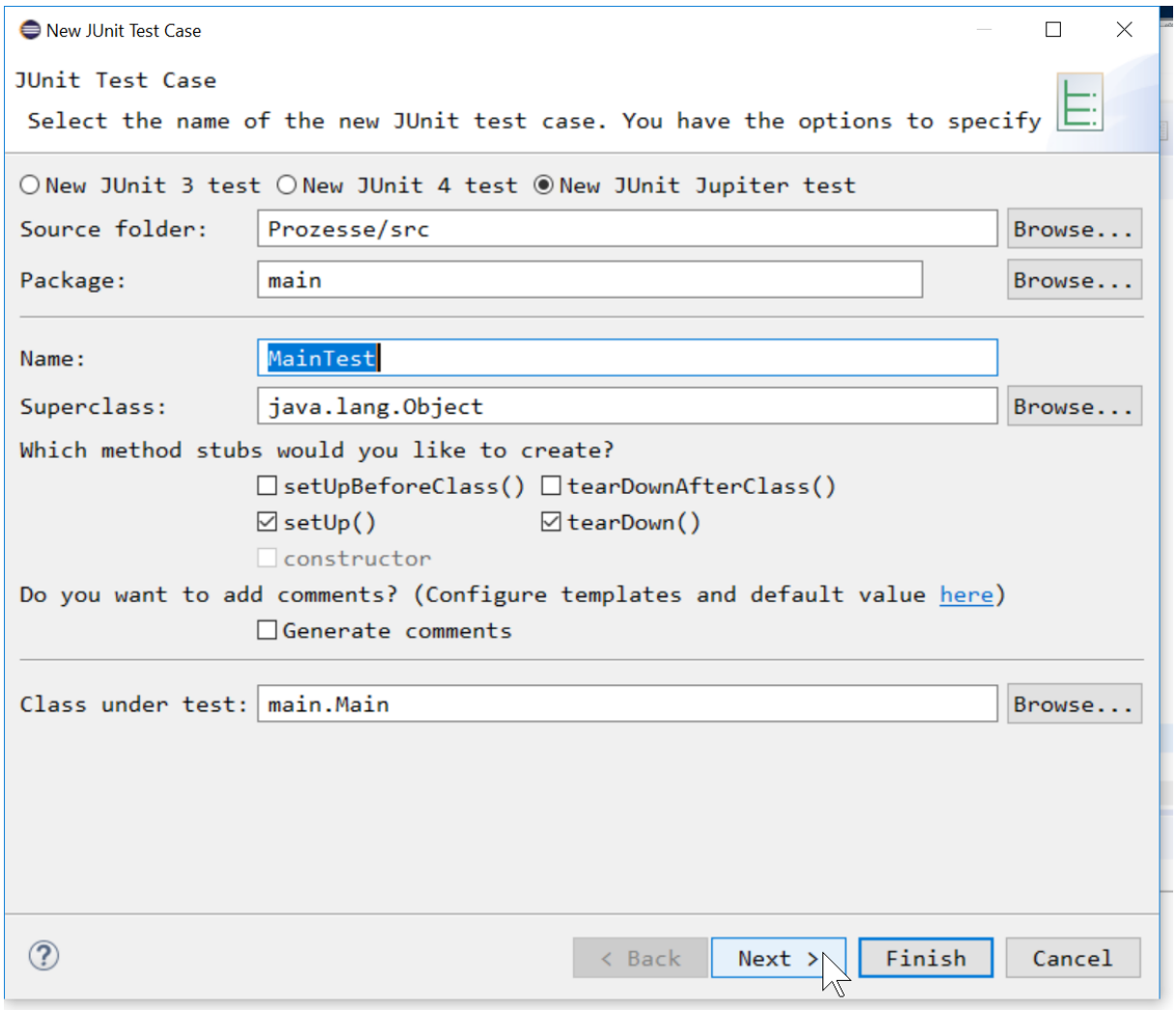
```
package main;

public class Main {

	public void show() {
		System.out.println("Hai");
	}

	public int wert(boolean b, boolean c){
		int x = 0;
		if ((b && c) || ((!b) & (!c))){
		  x = 1;
		} else {
		  x = 2;
		}
		return x;
	}

	public static void main(String[] args){
	  (new Main()).show();
	  args.hashCode();
	}

}
```



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.

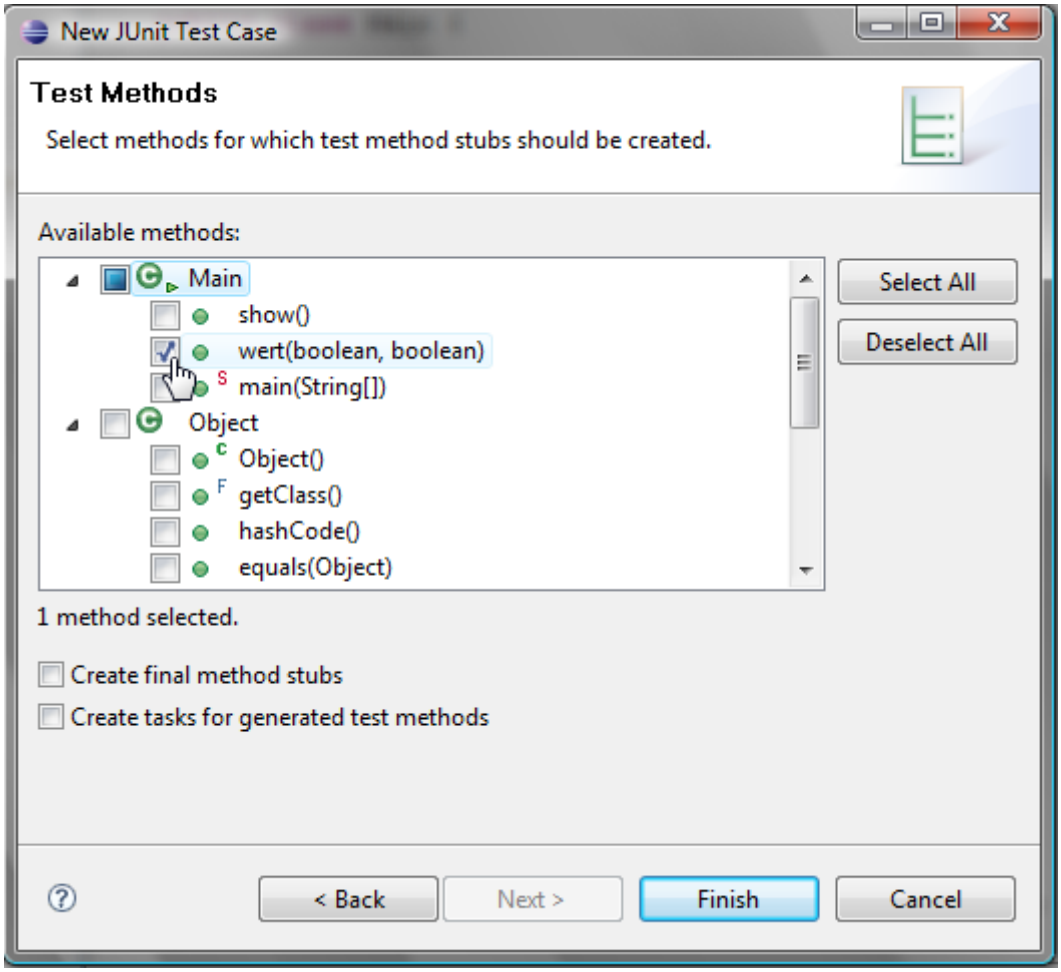


Im folgenden Fenster kann man sich für eine JUnit-Version entscheiden, dabei wird JUnit Jupiter oft auch mit JUnit 5 bezeichnet. Weiterhin kann man die Initialisierungsmethoden für JUnit auswählen. Generell kann man die Auswahl mit „Finish“ abschließen, Hier werden mit „Next>“ weitere Einstellungsmöglichkeiten betrachtet.

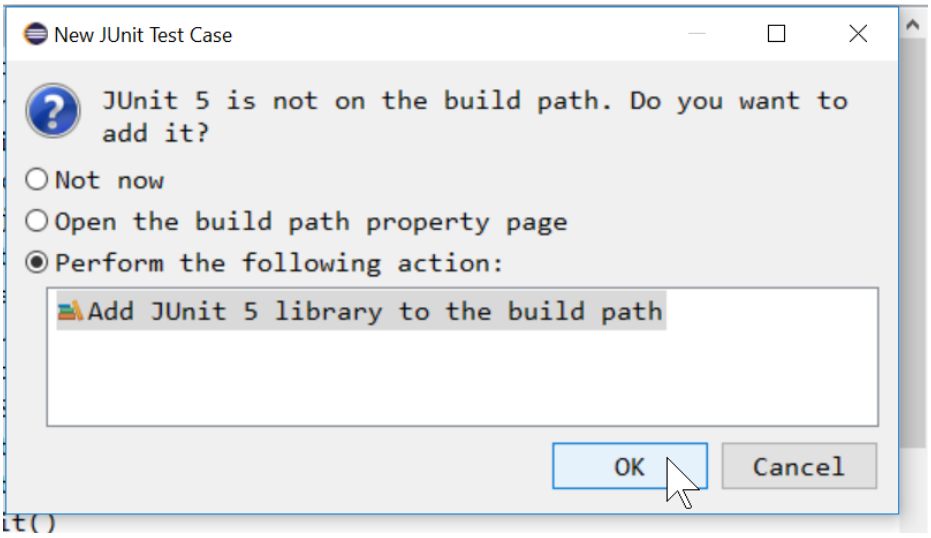


Man kann jetzt auswählen, für welche Methoden sogenannte Stubs erstellt werden sollen. Dies ist in diesem Fall ein leerer Testfall, der daran erinnern soll, dass hier noch ein Test fehlt.

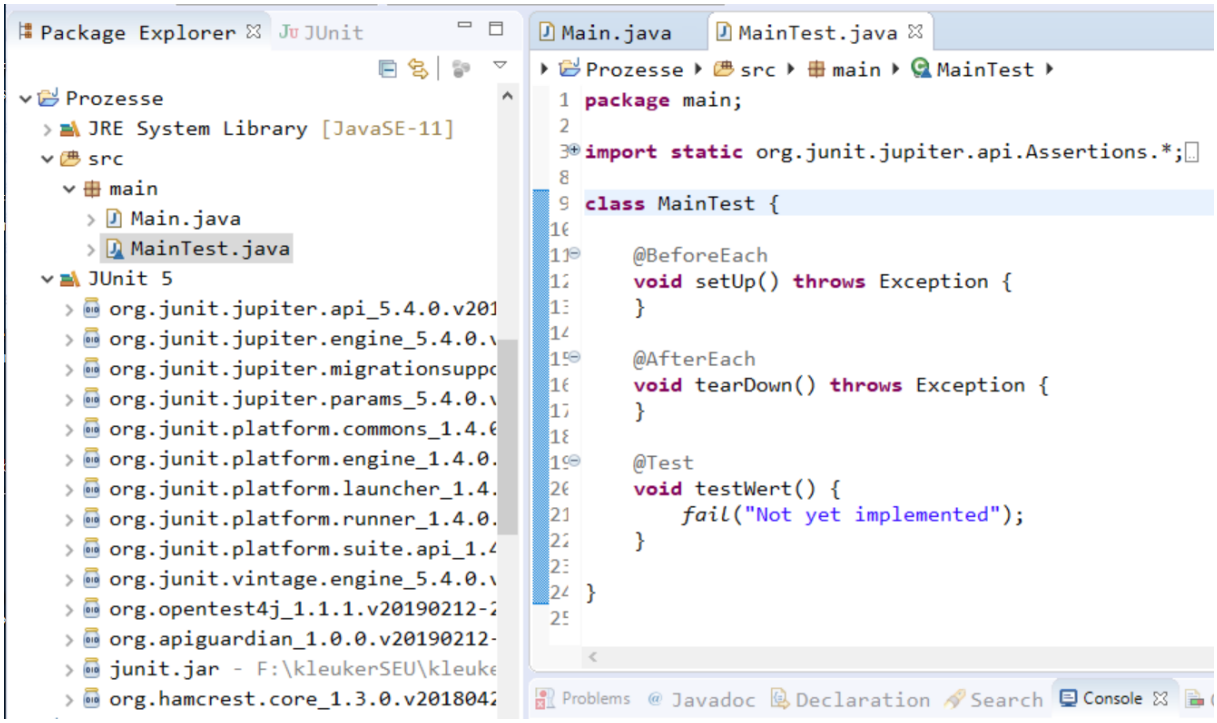
Im Beispiel wird die Methode wert ausgewählt und dann „Finish“ geklickt. Generell sei angemerkt, dass man auf diese Stubs ganz verzichten kann, da man genauere Tests genauso schnell von Hand schreibt.



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



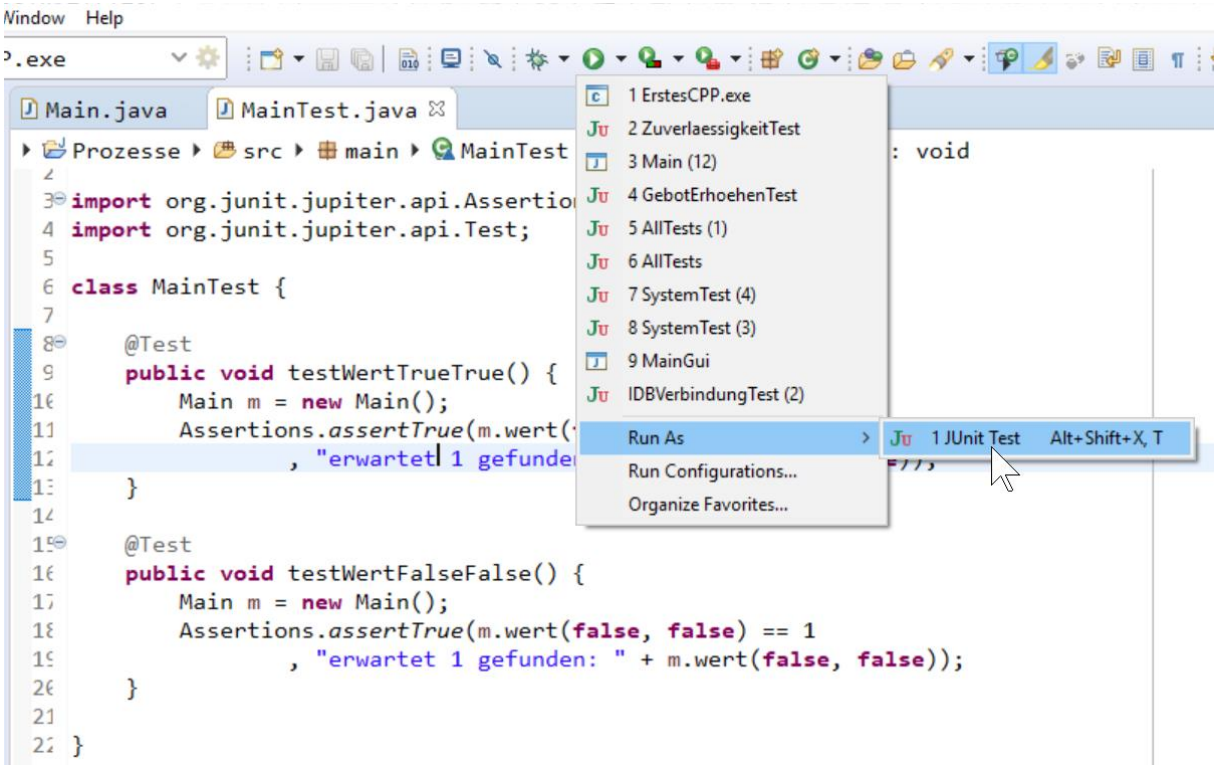
Man erhält dann z. B. folgende Testklasse.



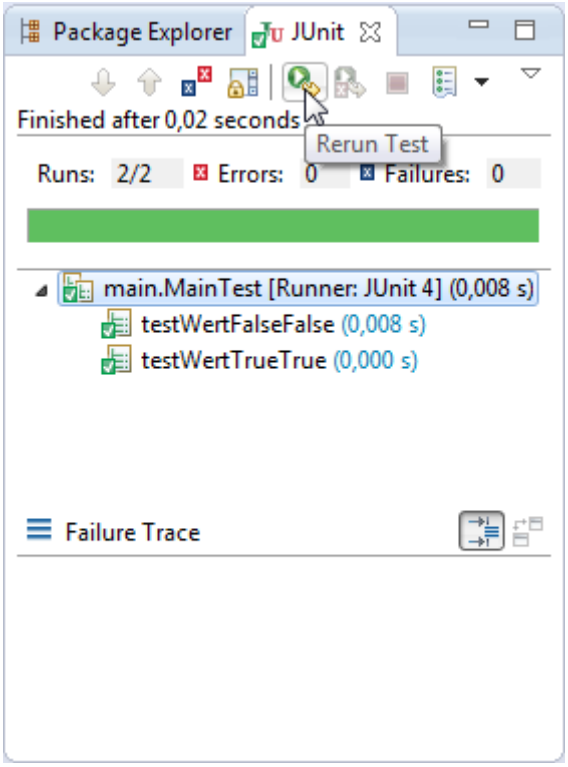
Man kann dann die Testklasse bearbeiten, weitere Testmethoden ergänzen und die Tests starten.

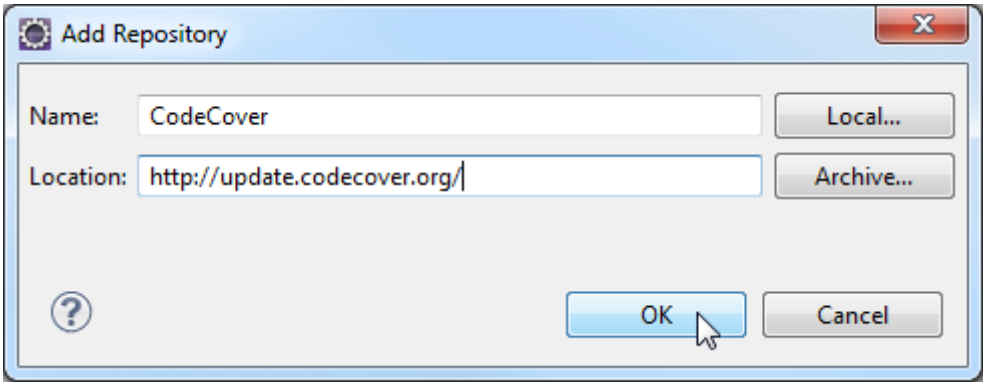
Dazu wird im Startmenü beim Pfeil nach unten neben  dann „Run As“ und „JUnit Test“ gewählt. Bei den folgenden Tests wurde ein irritierender static-Import ersetzt.

```
package main;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

class MainTest {

  @Test
  public void testWertTrueTrue() {
    Main m = new Main();
    Assertions.assertTrue(m.wert(true, true) == 1
        , "erwartet 1 gefunden: " + m.wert(true, true));
  }

  @Test
  public void testWertFalseFalse() {
    Main m = new Main();
    Assertions.assertTrue(m.wert(false, false) == 1
        , "erwartet 1 gefunden: " + m.wert(false, false));
  }
}
```



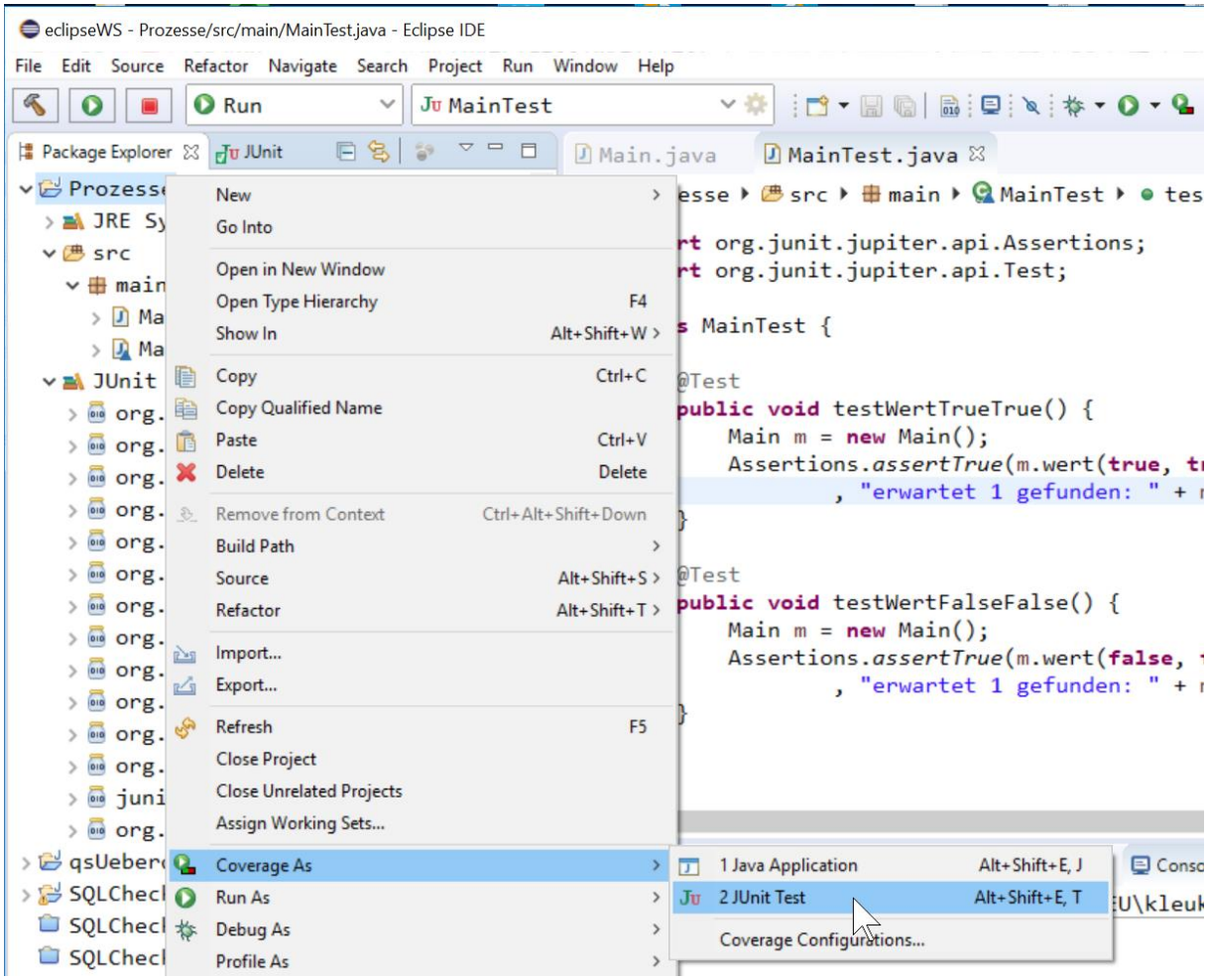
Nach Ablauf der Tests wird das Ergebnis in einem eigenen Reiter auf der linken Seite angezeigt. Hier können dann Tests auch wiederholt werden. Durch Anklicken der Tests erhält man weitere Informationen. Fehlerinformationen werden unter „Failure Trace“ ausgegeben.





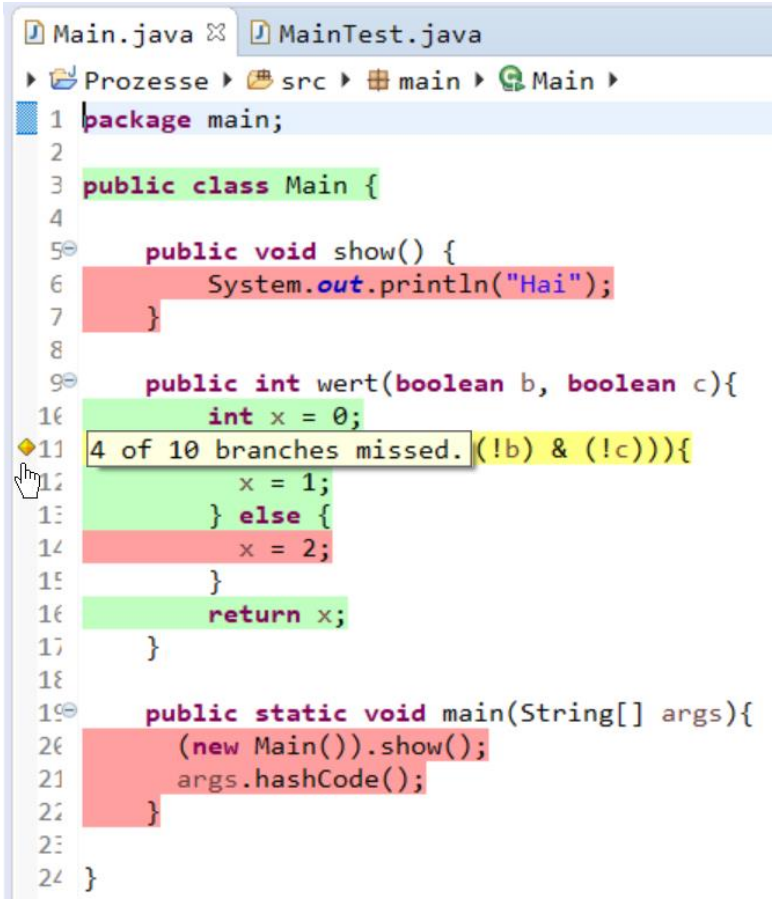
14.2 Erste Nutzung der Überdeckungsmessung

Es stellt sich nun die Frage, was alles im Test berücksichtigt wurde. Dies kann mit CodeCover analysiert werden. Es wird protokolliert, was alles ausgeführt wurde. Um CodeCover zu nutzen, wird ein Rechtsklick auf dem Projekt gemacht und „Coverage As > JUnit Test“ ausgewählt.



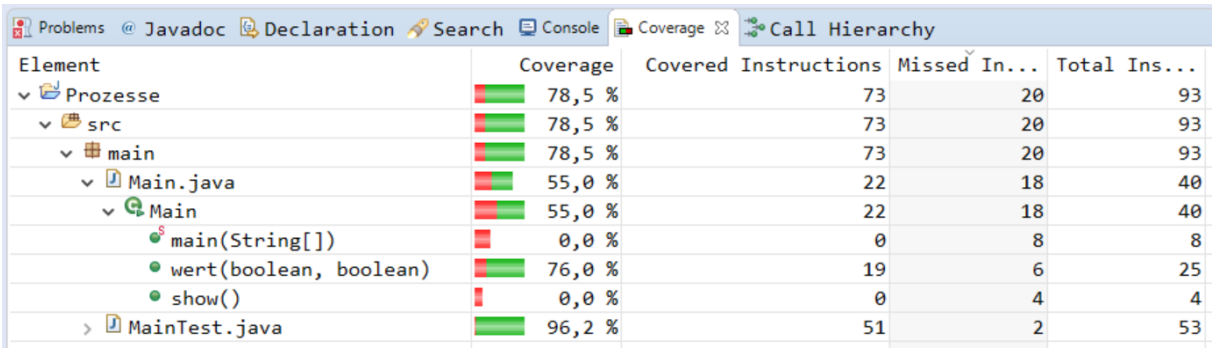
Nach der Ausführung wird die Überdeckung angezeigt, dabei steht „rot“ für nicht ausgeführten Code, „grün“ für ausgeführten Code und „gelb“ für teilweise ausgeführten Code, wenn z. B.

bei einer Alternative nicht alle Fälle oder bei zusammengesetzten Booleschen Ausdrücken nicht jeder atomare Teilausdruck einmal wahr und einmal falsch gewesen ist. Die kleinen Rauten am Rand informieren etwas genauer über die Überdeckungen der Booleschen Ausdrücke.




```
1  package main;
2
3  public class Main {
4
5      public void show() {
6          System.out.println("Hai");
7      }
8
9      public int wert(boolean b, boolean c){
10         int x = 0;
11 4 of 10 branches missed. (!b) & (!c))){
12             x = 1;
13         } else {
14             x = 2;
15         }
16         return x;
17     }
18
19     public static void main(String[] args){
20       (new Main()).show();
21       args.hashCode();
22     }
23
24 }
```

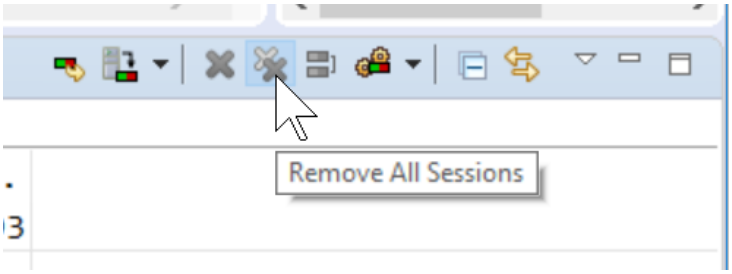
Im unteren Bereich werden mehrere Überdeckungsmaße angegeben, dabei können gezeigte Klassen aufgeklappt werden.



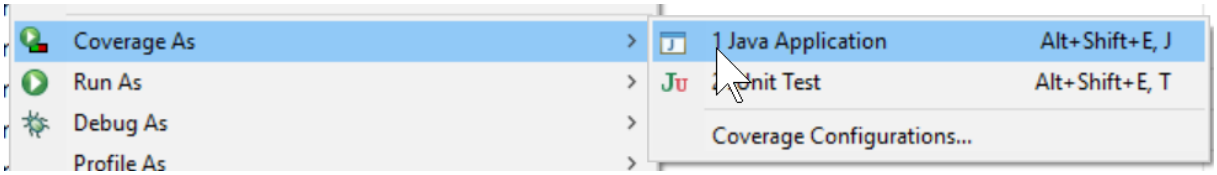
Element	Coverage	Covered Instructions	Missed In...	Total Ins...
Prozesse	78,5 %	73	20	93
src	78,5 %	73	20	93
main	78,5 %	73	20	93
Main.java	55,0 %	22	18	40
Main	55,0 %	22	18	40
main(String[])	0,0 %	0	8	8
wert(boolean, boolean)	76,0 %	19	6	25
show()	0,0 %	0	4	4
MainTest.java	96,2 %	51	2	53

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.

Auf der rechten Seite existieren einige Steuerungsmöglichkeiten, u. a. können über das doppelte X alle Markierungen gelöscht werden.

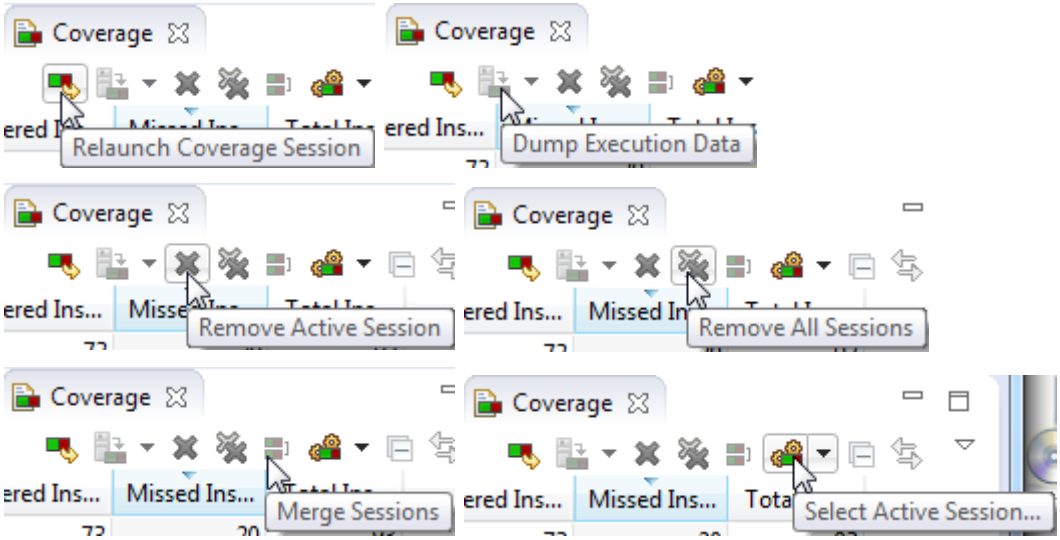


Wird das eigentliche Programm über „Coverage As > Java Application“ gestartet, wird die Überdeckung für die normale Programmausführung gemessen. Dies ist z. B. hilfreich, wenn die Testerstellung mit JUnit aufwändig ist und manuell Testspezifikationen abgearbeitet werden. Läuft das Programm mehrfach hintereinander, werden die Überdeckungsinformationen ergänzt.

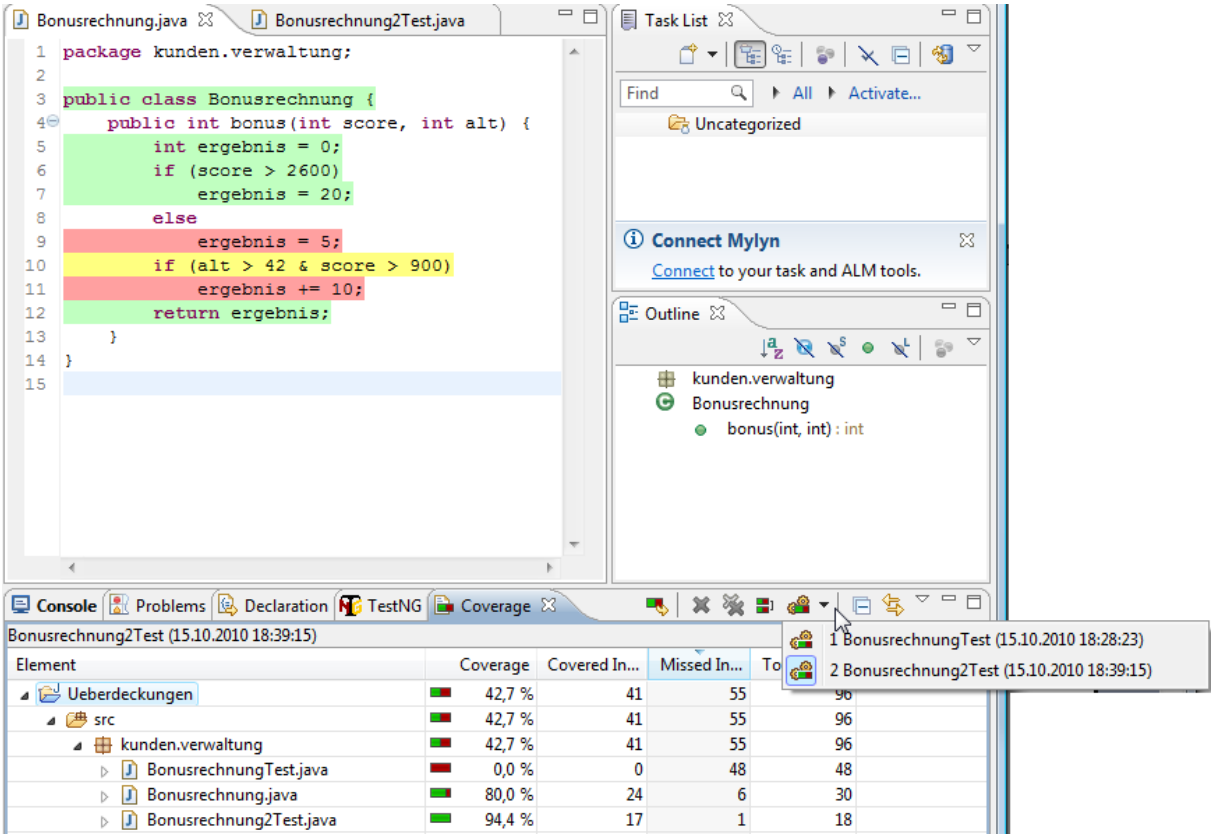


14.3 Vereinigung von Testdurchläufen

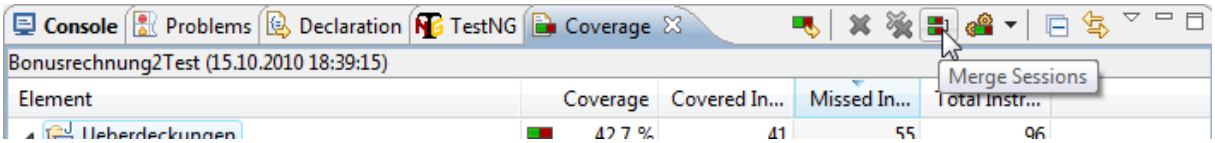
Man beachte, dass der Reiter Coverage einige Steuerungsmöglichkeiten für Testläufe anbietet.



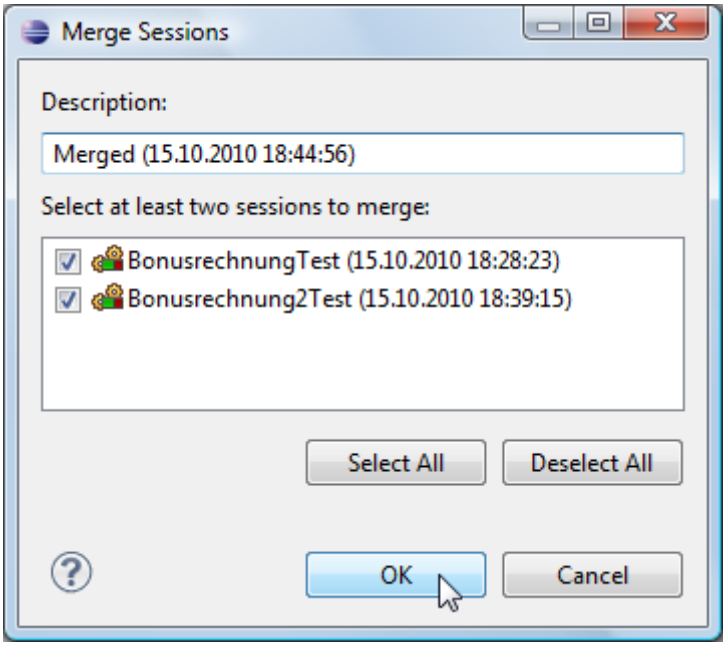
Lässt man verschiedene Tests laufen, kann man die Ergebnisse der Testdurchläufe vereinigen. Das folgende Bild zeigt das Überdeckungsergebnis für eine zweite Testklasse, man sieht rechts unten die Möglichkeit, über den kleinen Pfeil nach unten, zwischen den Ergebnissen der Testdurchführungen umzuschalten.



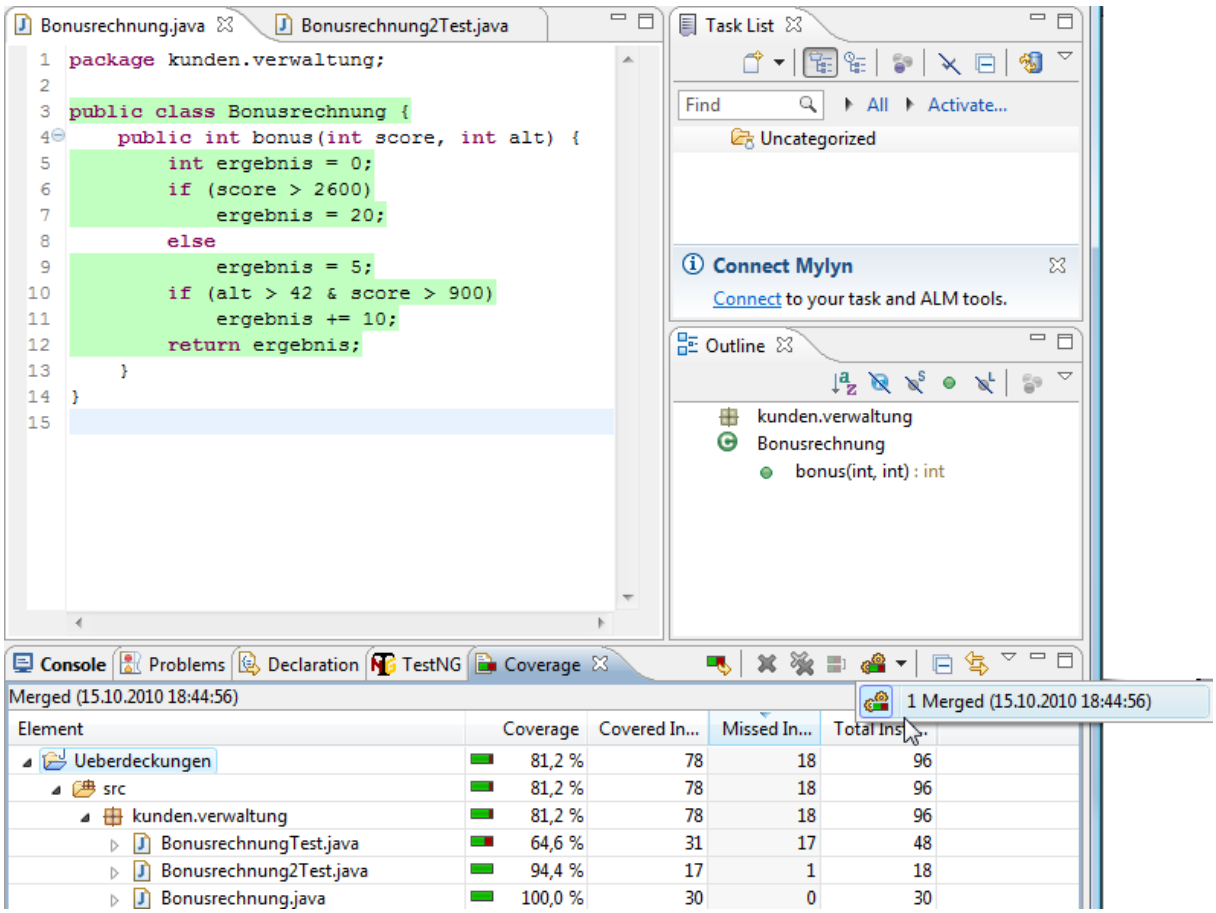
Der Merge-Knopf ermöglicht es, die Ergebnisse mehrerer Testdurchführungen zu vereinigen.



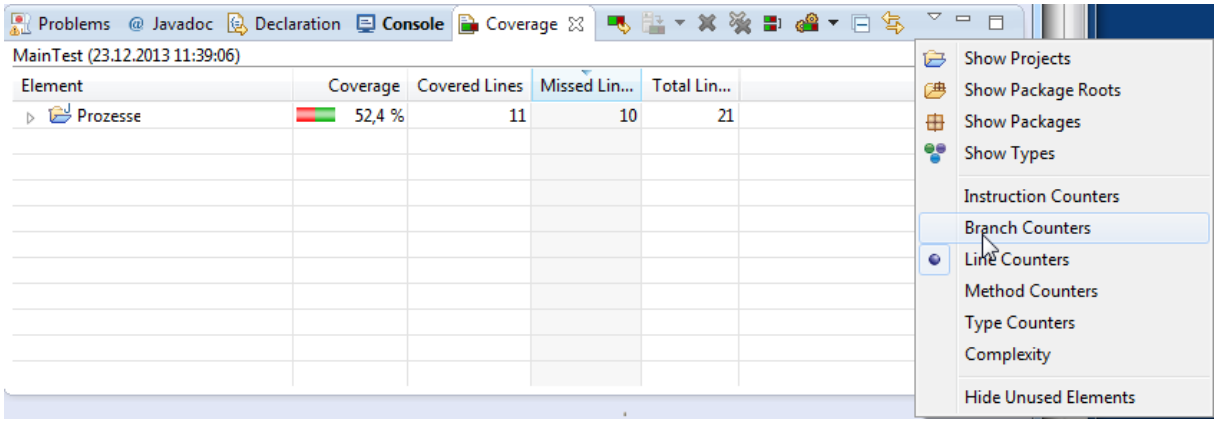
Man kann dann auswählen, welche Sessions vereinigt werden sollen.



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.

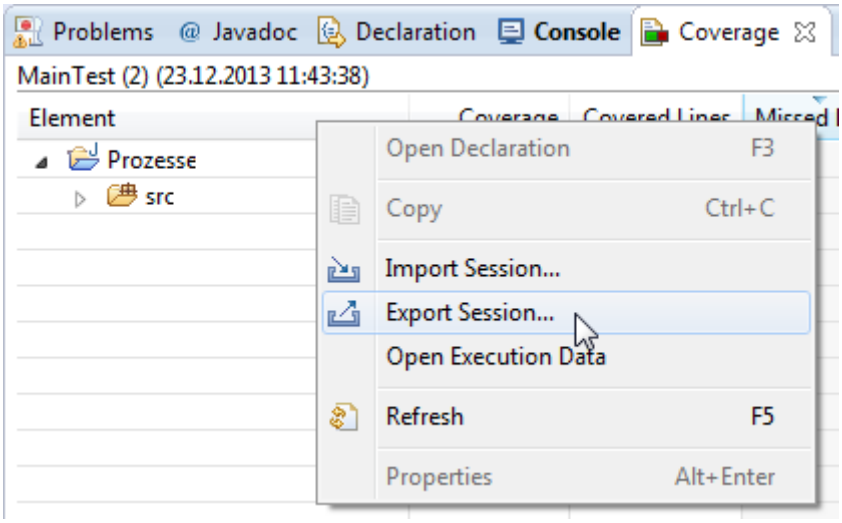


Mit dem weißen Pfeil nach unten, dritter von rechts, kann man im Coverage-Reiterverschiedene Zählweisen einstellen. Man beachte, dass man hier auch Verzeigungen (Branch) zählen lassen kann.

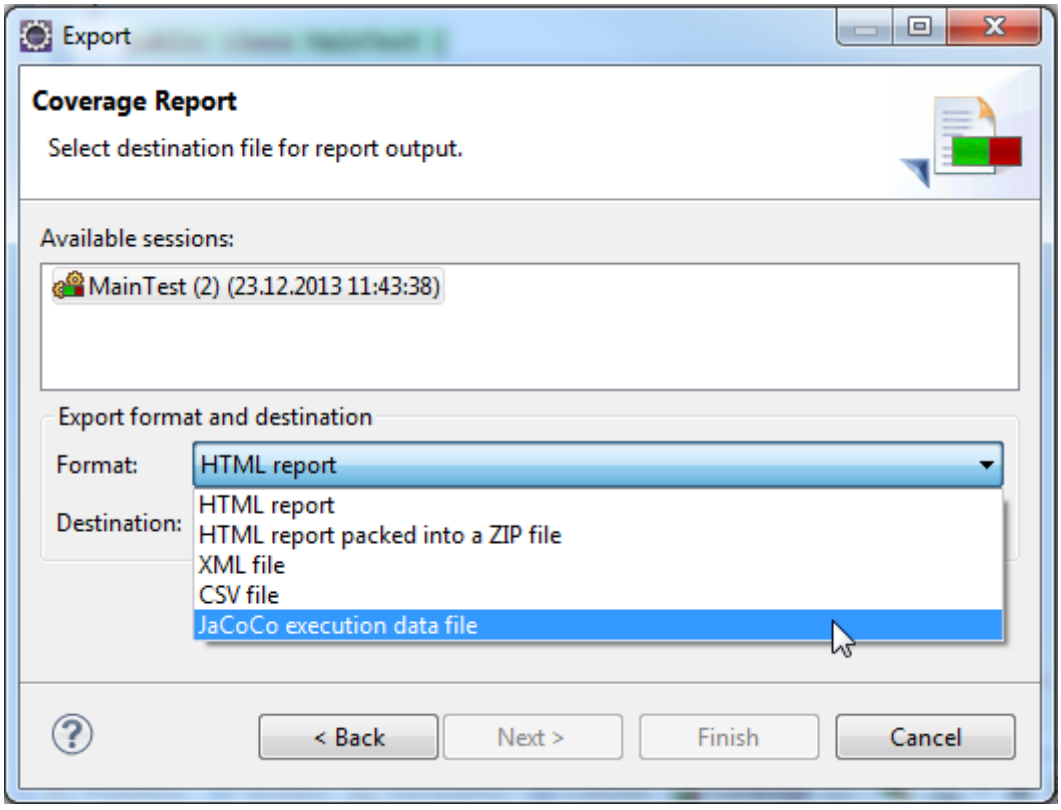


14.4 Verwaltung von Testdurchläufen

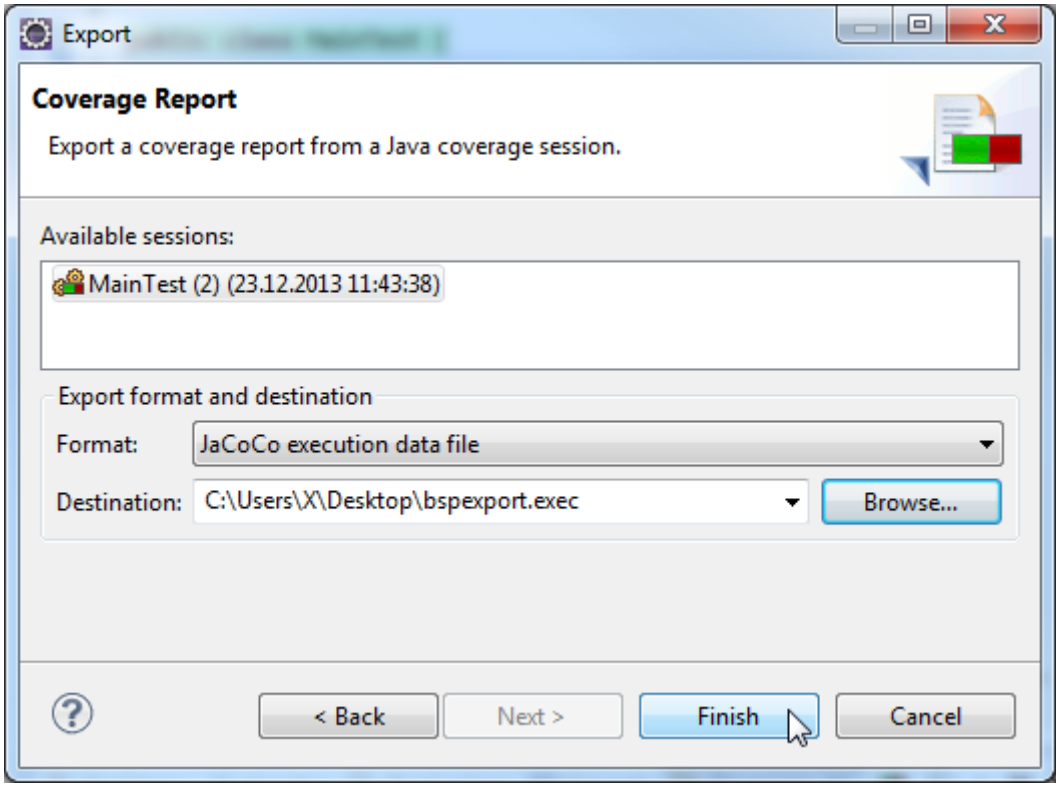
Zunächst erkennt man nicht, dass man die Daten der Testdurchläufe auch weiter verwalten und aufbereiten kann. Diese Möglichkeiten bestehen aber, wenn man einen Rechtsklick im Coverage-Fenster macht.



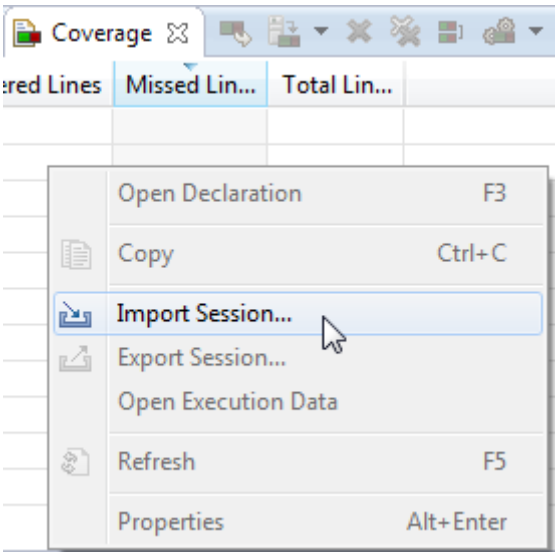
Nach der Wahl von „Export Session“ muss das Format ausgewählt werden. Es bestehen folgende Auswahlmöglichkeiten. Mit HTML gibt es eine einfache Webaufbereitung, die auch den farbig gekennzeichneten Code zeigt. Mit XML besteht die Möglichkeit zur Weiterverarbeitung. Interessant ist die Auswahl als „JaCoCo execution data file“, die hier weiter verfolgt wird.



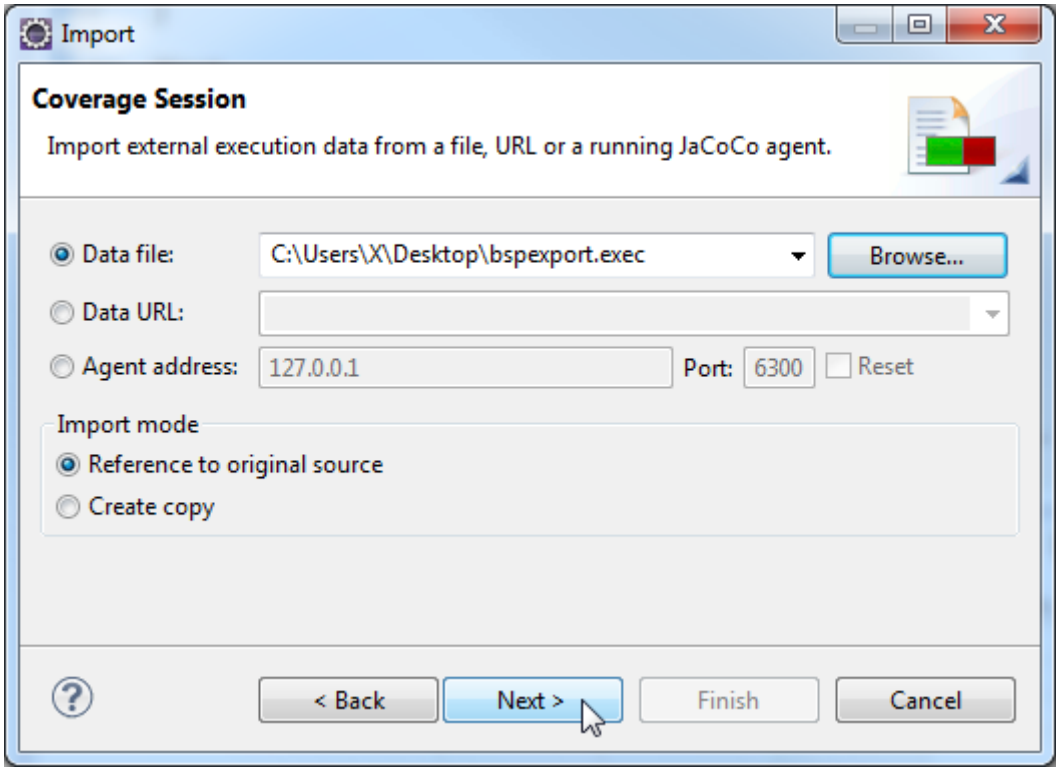
Der Speicherort kann frei und sinnvoller als in der Abbildung gewählt werden, die Endung „.exec“ sollte beibehalten werden.



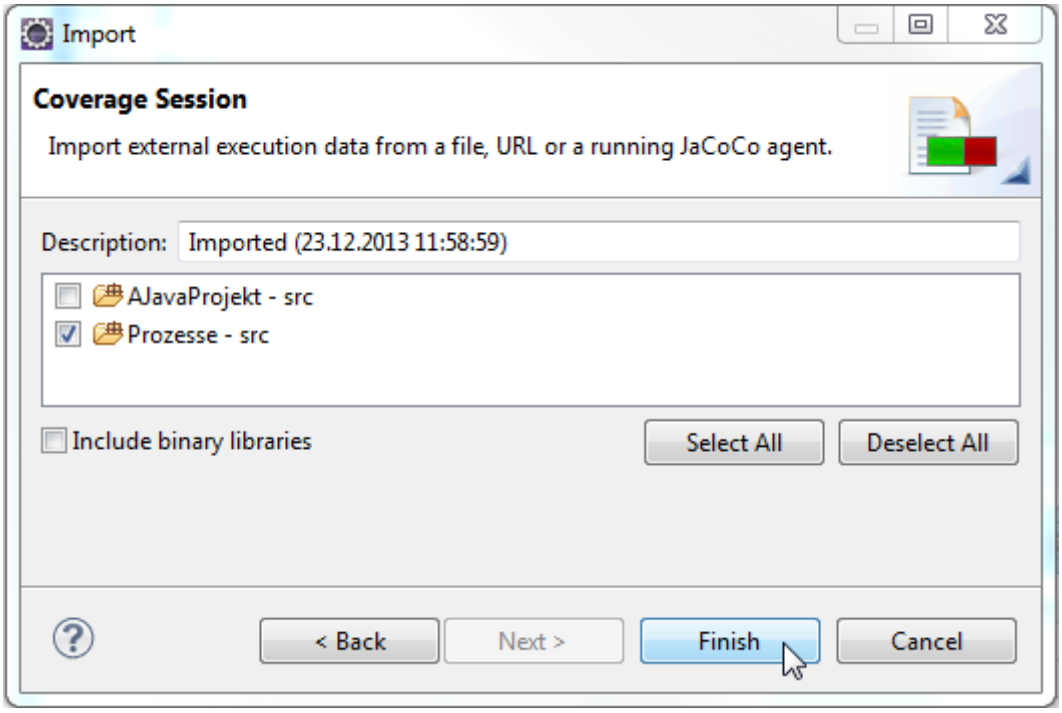
Beendet man Eclipse und öffnet den Workspace wieder, sind zunächst alle Testmarkierungen verloren. Hat man diese als „JaCoCo file“ gespeichert, kann man diese wieder wie folgt importieren.



Hier muss die gespeicherte Datei unter „Browse...“ gesucht werden.

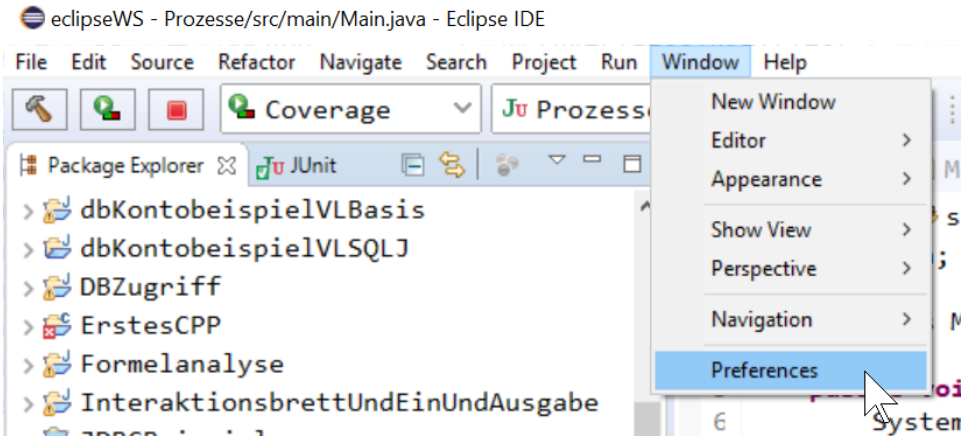


Den Namen des Eintrags kann man in der „Description:“ ändern, muss es aber nicht. Wichtig ist, dass man im mittleren Fenster das Projekt markiert, zu dem die aufgezeichneten Daten gehören. Die Einstellungen werden mit „Finish“ abgeschlossen.

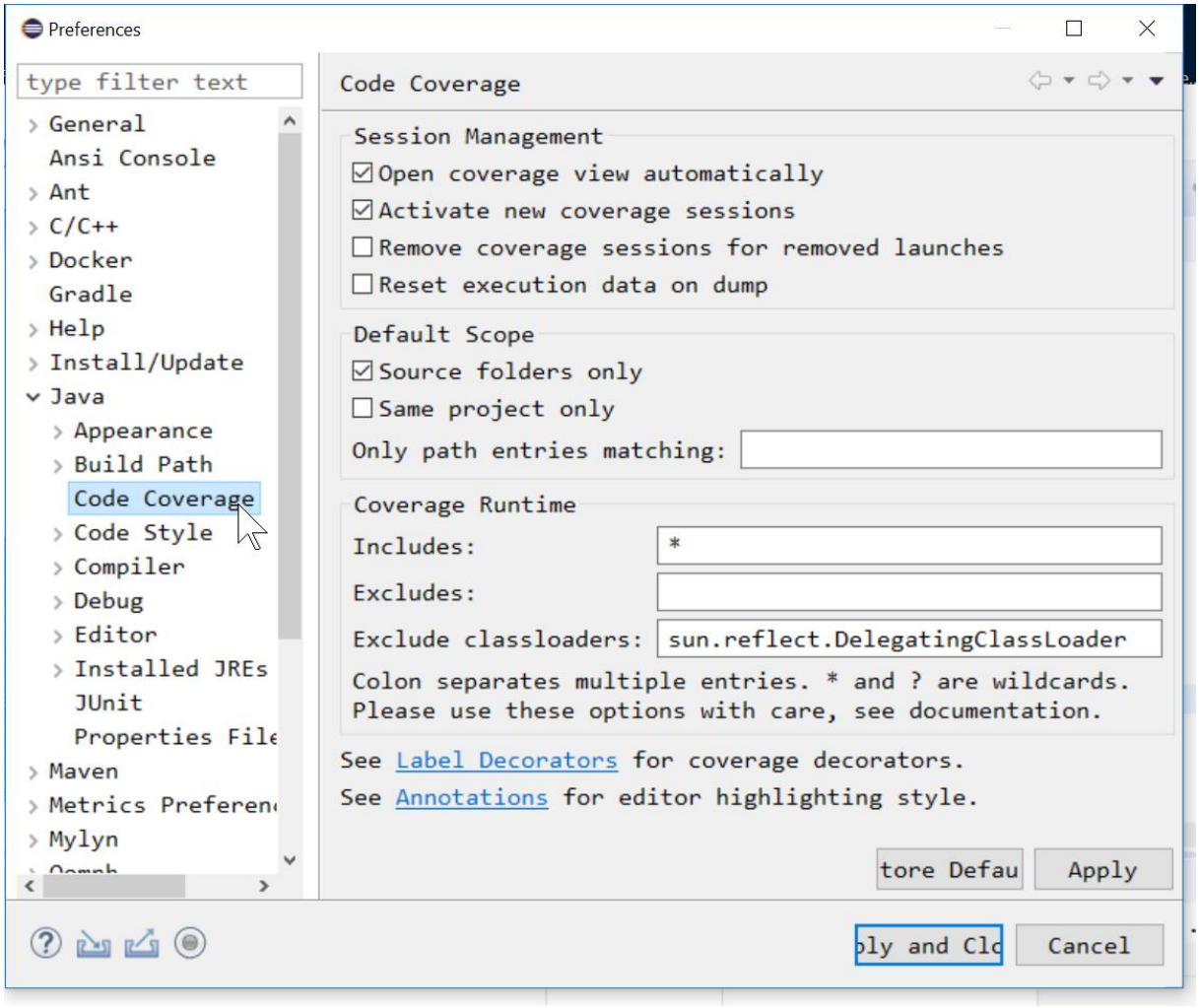


Die importierte Session wird zu den existierenden hinzugefügt und kann genauso genutzt werden.

Über „Window > Preferences“ ist das Überdeckungswerkzeug etwas konfigurierbar.



Die Einstellungen befinden sich direkt unter „Java > Code Coverage“ auf der linken Seite.


Preferences
type filter text
General
Ansi Console
Ant
C/C++
Docker
Gradle
Help
Install/Update
Java
Appearance
Build Path
Code Coverage
Code Style
Compiler
Debug
Editor
Installed JREs
JUnit
Properties File
Maven
Metrics Preferen
Mylyn
Code Coverage
Session Management
Open coverage view automatically
Activate new coverage sessions
Remove coverage sessions for removed launches
Reset execution data on dump
Default Scope
Source folders only
Same project only
Only path entries matching:
Coverage Runtime
Includes:
*
Excludes:
Exclude classloaders:
sun.reflect.DelegatingClassLoader
Colon separates multiple entries. * and ? are wildcards.
Please use these options with care, see documentation.
See Label Decorators for coverage decorators.
See Annotations for editor highlighting style.
tore Defau
Apply
Cancel

15 Testen mit TestNG (4.3.1)

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.

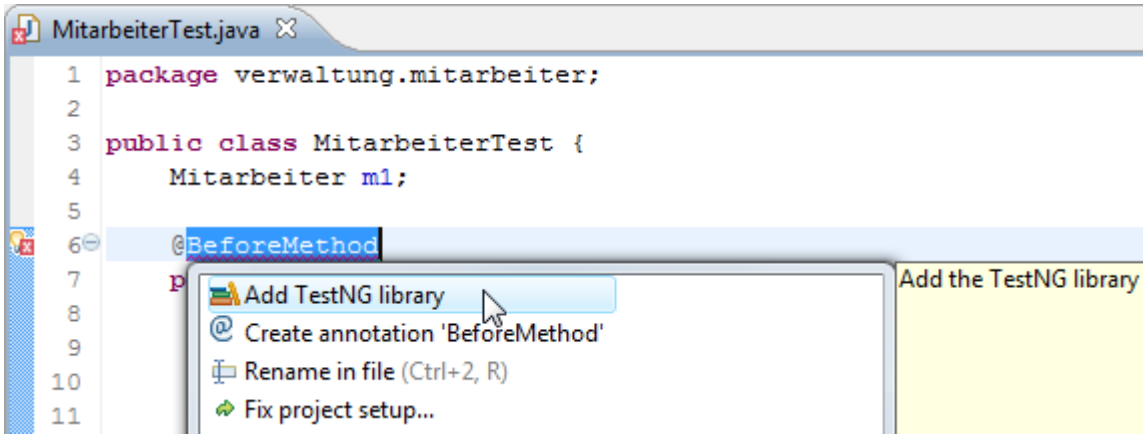
15.2 Erstellung eines ersten Testfalls mit TestNG

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.

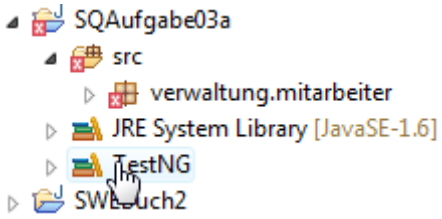
MitarbeiterTest.java

```
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     }
```

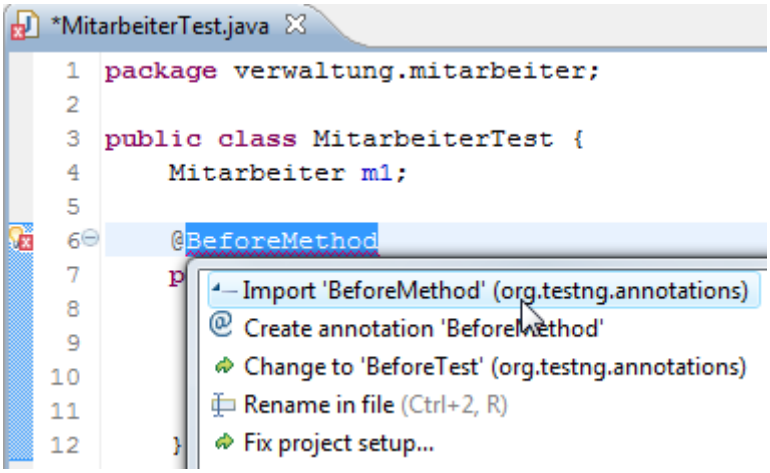
Macht man dann einen Linksklick auf die Fehlerkennzeichnung am linken Rand, so wird automatisch die Einbindung von TestNG vorgeschlagen, die dann ausgeführt wird.



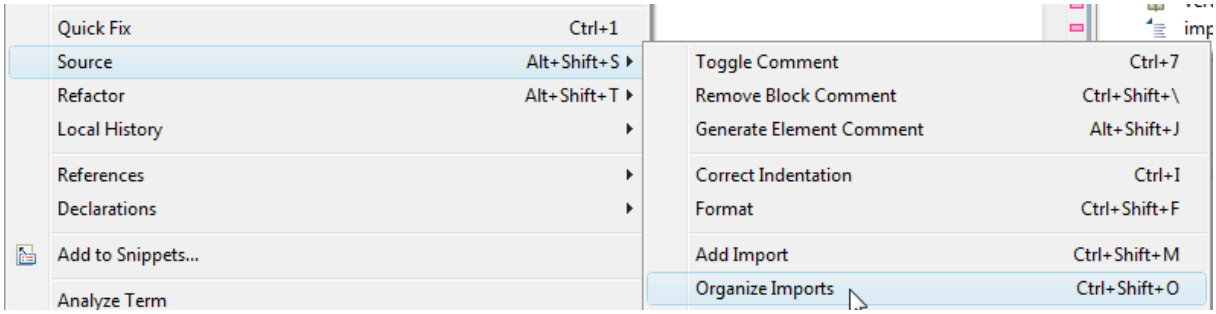
TestNG ist dann unter den benutzten Bibliotheken eingetragen.



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.

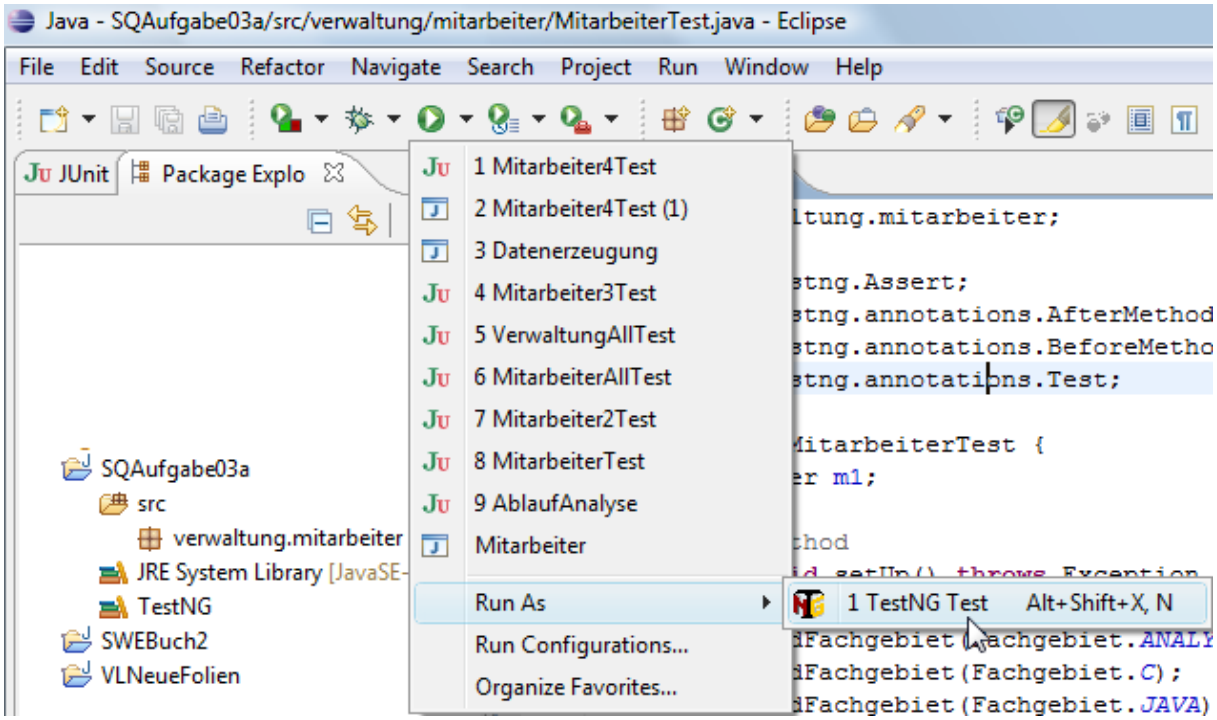


Die import-Anweisungen für eine Testklasse können dann wie folgt aussehen.

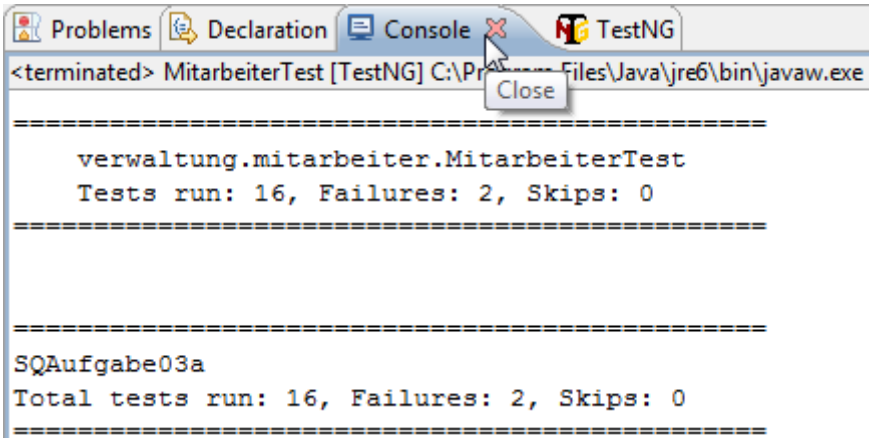
MitarbeiterTest.java

```
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);
```

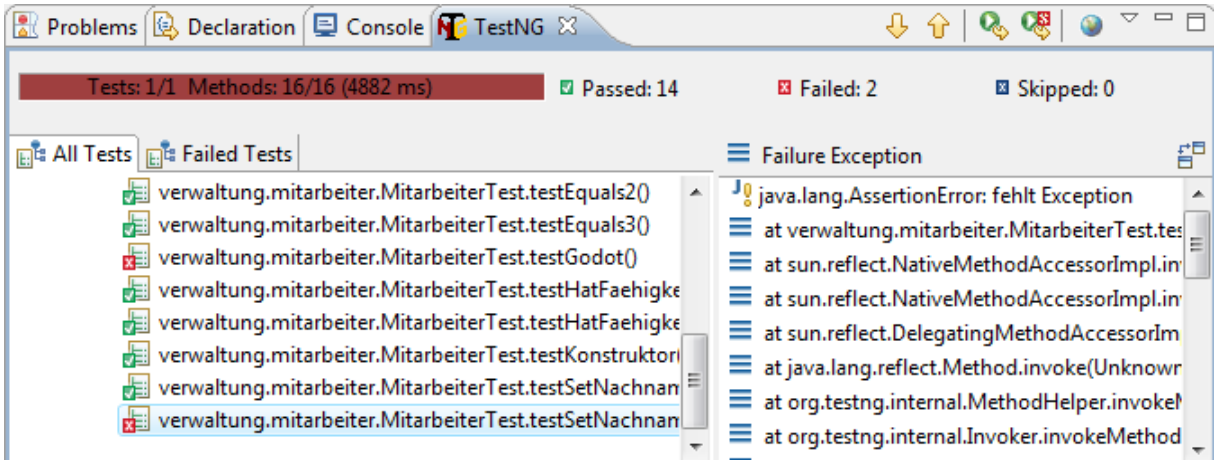
TestNG-Tests können wie normale Java-Programme und JUnit-Tests über viele Wege ausgeführt werden. Befindet sich die Klasse mit den Tests im aktuellen Editorfenster, kann der Aufruf über das Menü gestartet werden, dass man durch den kleinen Pfeil neben dem Run-Knopf erreicht. 



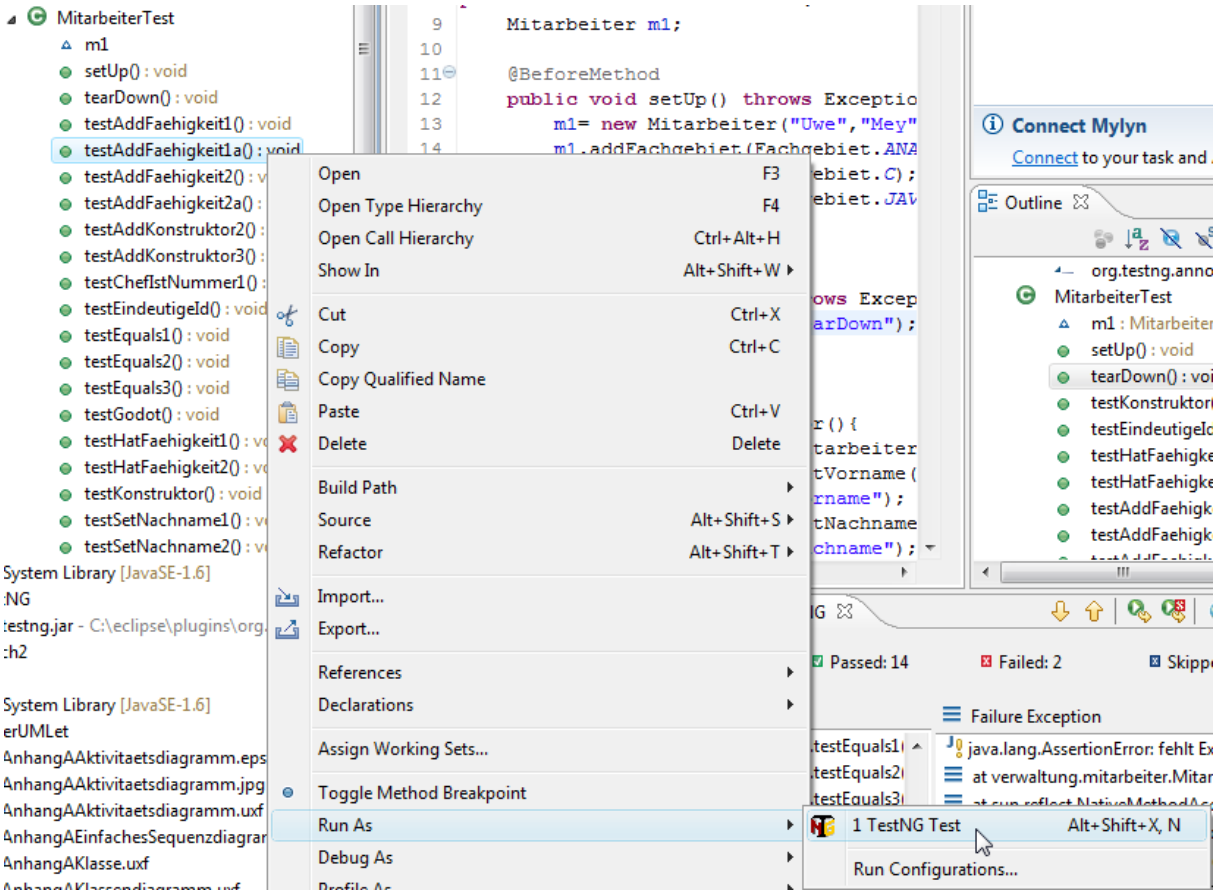
Die Ergebnisse stehen zum Einen in der Consolen-Ausgabe.



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.

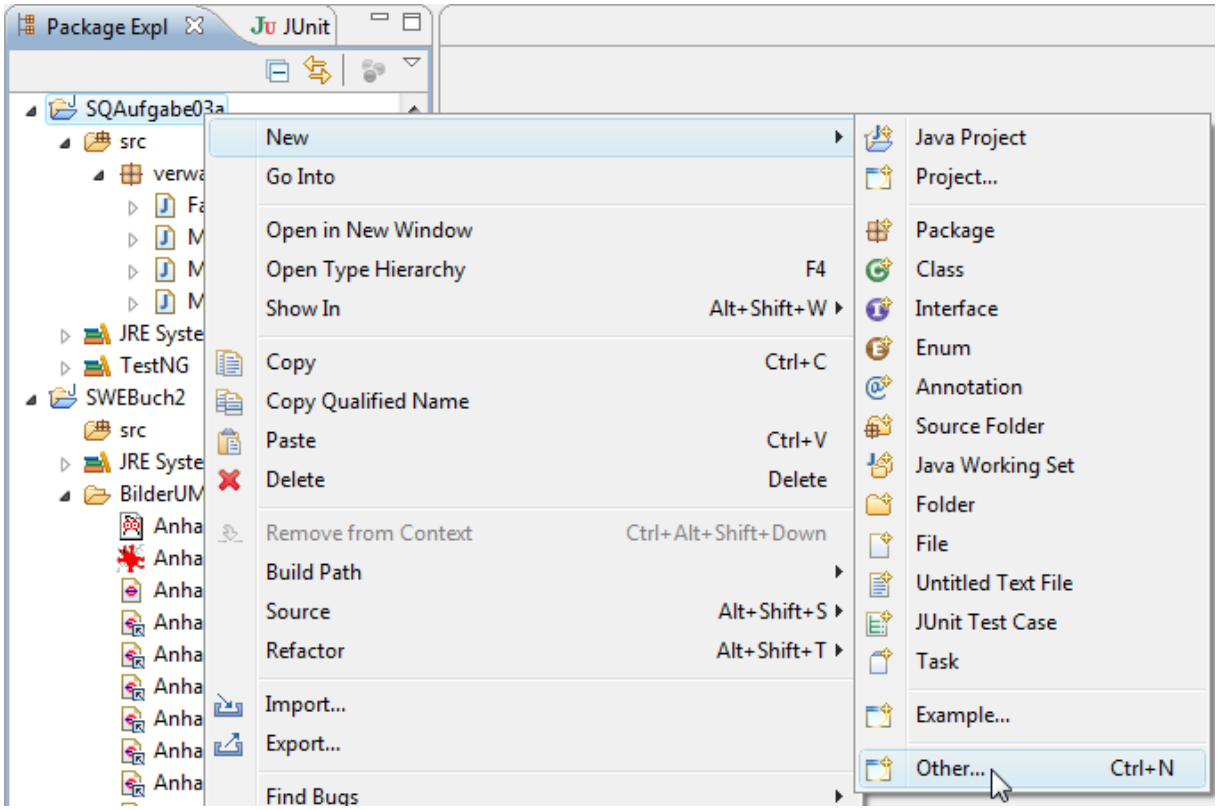


15.3 Erstellung von Start-Konfigurationen

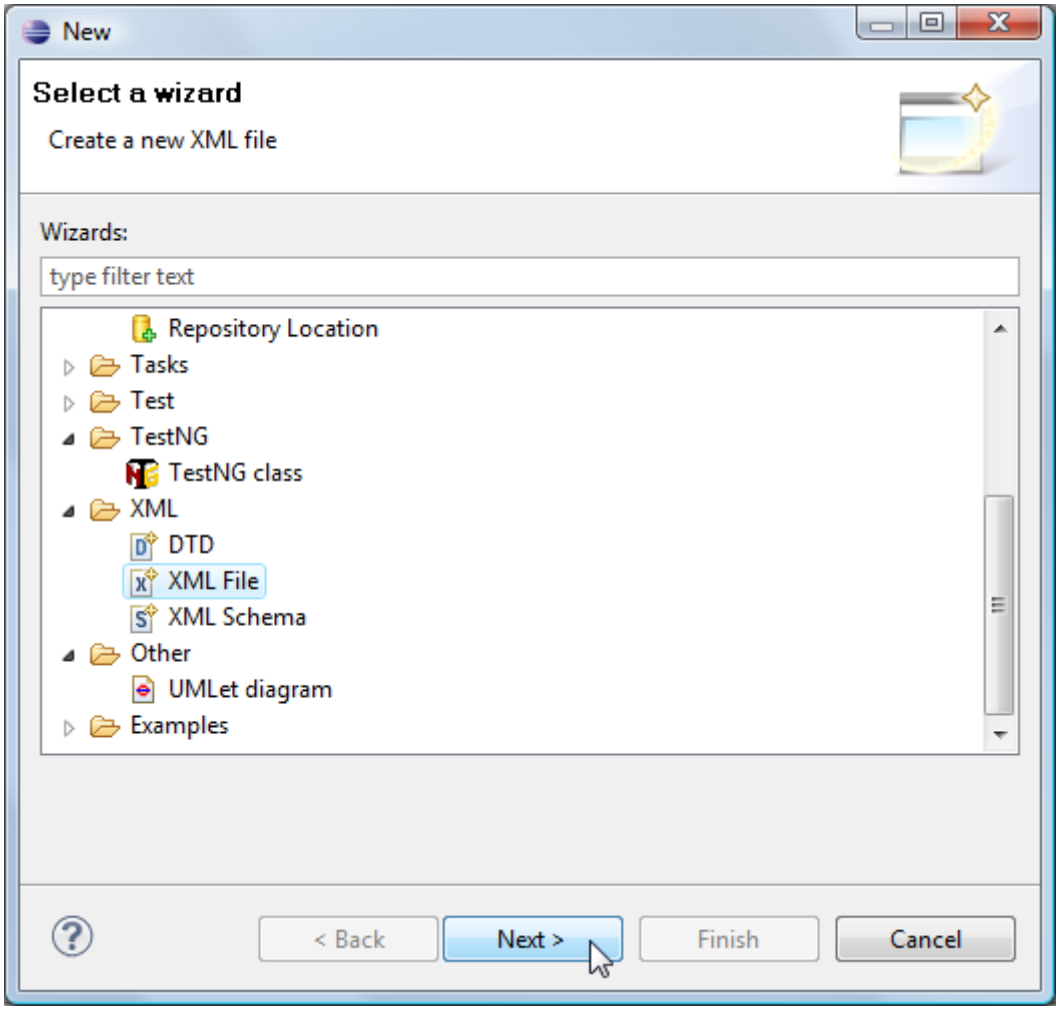
TestNG kann generell sehr flexibel gestartet werden. Interessant sind dabei die Möglichkeiten die Testausführung über die XML-Datei testng.xml zu steuern. Das folgende Beispiel zeigt als

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

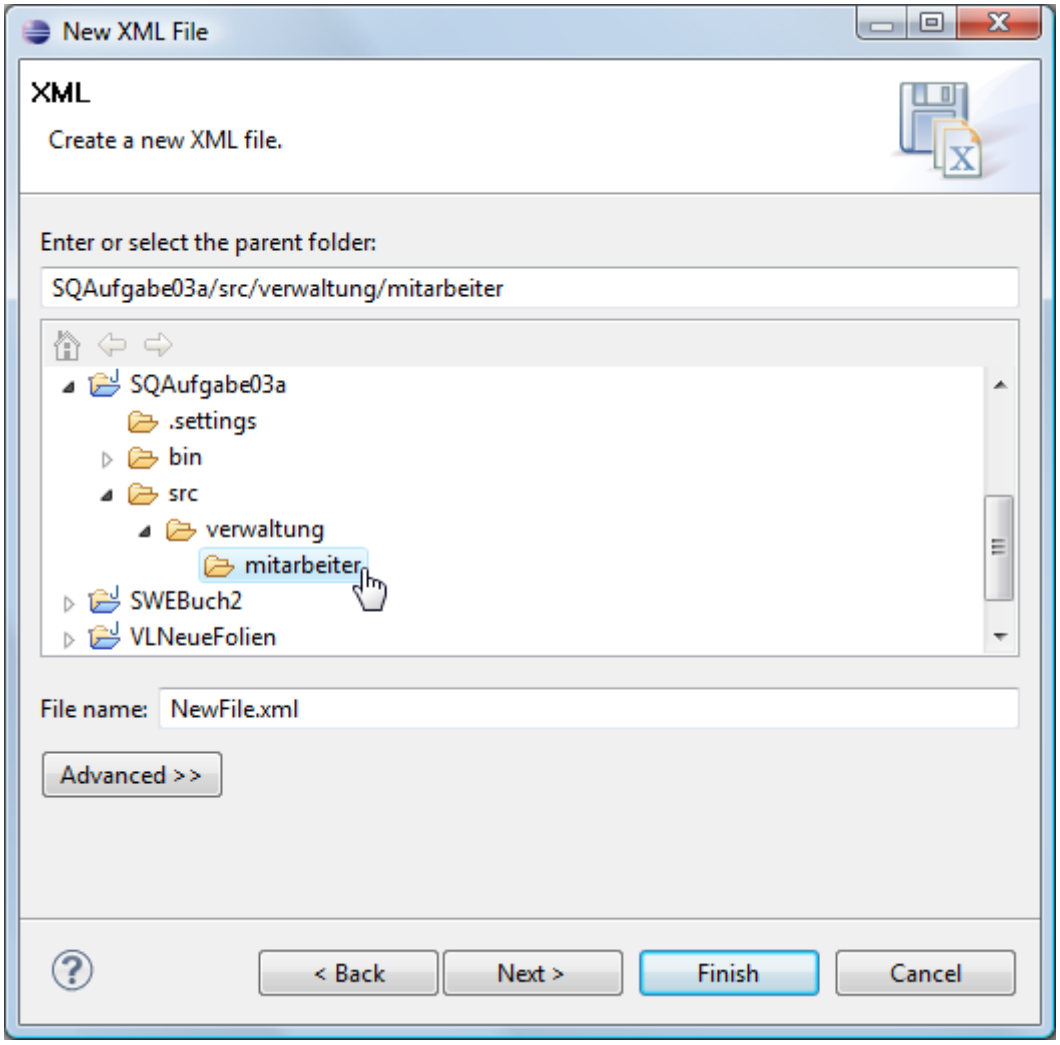
Abhängig von der Eclipse-Installation, kann man eine XML-Datei systematisch anlegen oder muss sie von Hand als Textdatei anlegen. Zur Erstellung einer XML-Datei kann man wie folgt vorgehen. Zunächst wird „New>Other...“ gewählt.



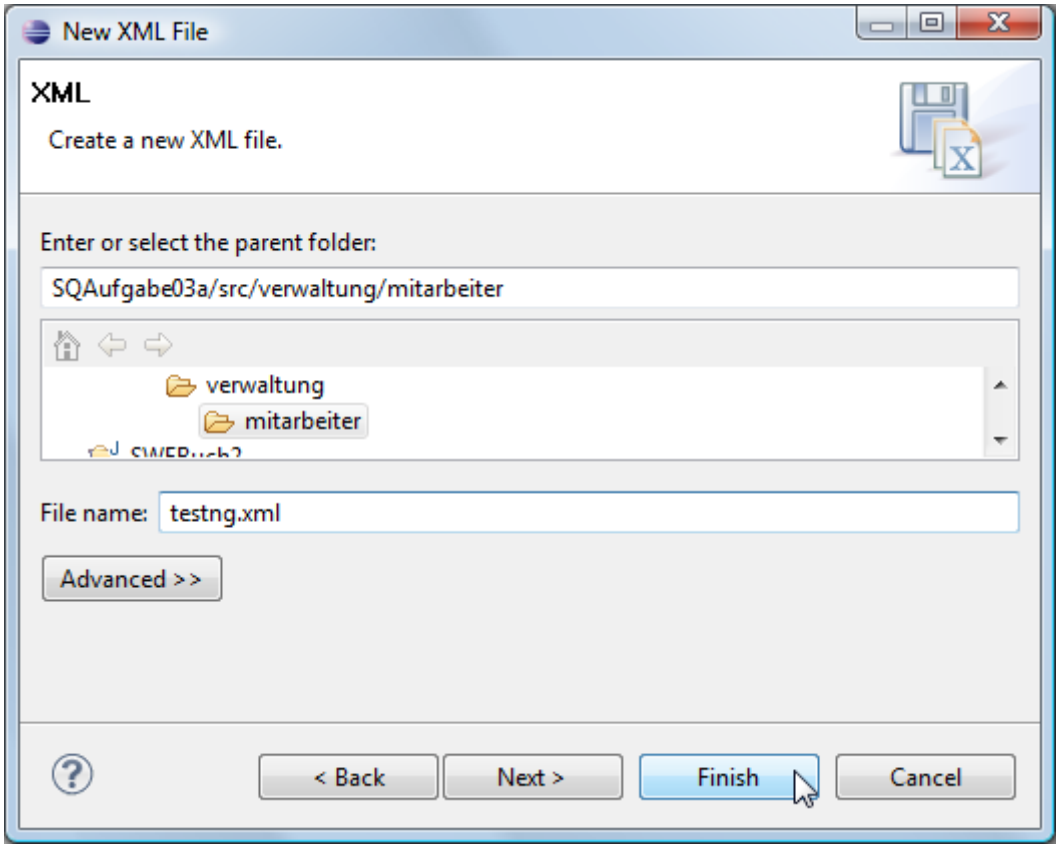
Hier wird der Punkt „XML“ aufgeklappt und „XML File“ gewählt und „Next>“ gedrückt.



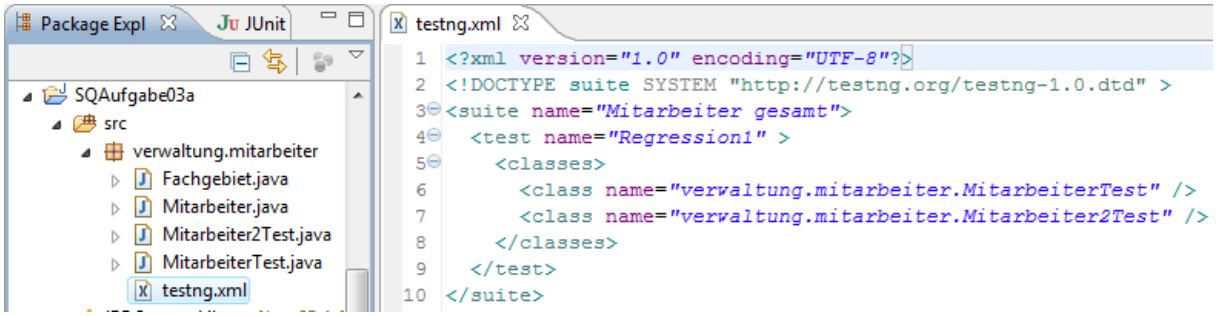
Zunächst wird im mittleren Browser ein passendes Verzeichnis, z. B. im src-Verzeichnis das Paket mit den zu testenden Klassen gewählt. Generell erlaubt es TestNG. dass mehrere XML-Dateien existieren, die dann zur Testausführung ausgewählt werden müssen.



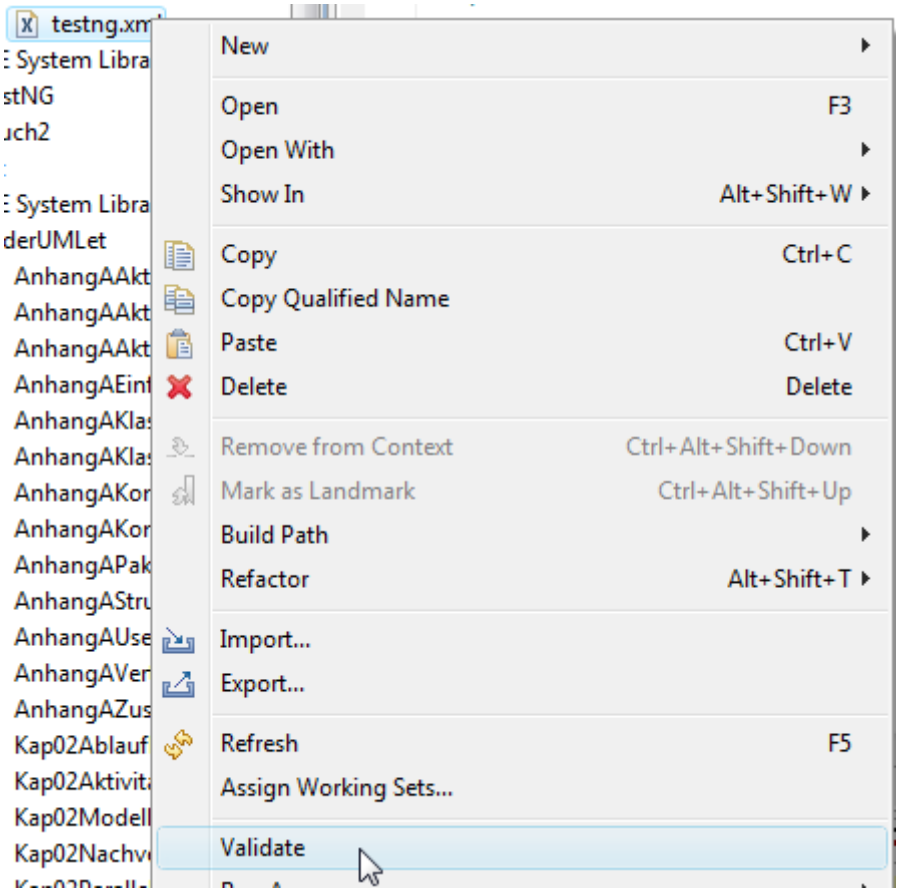
Als „File name“ wird testng.xml eingegeben und „Finish“ gedrückt.



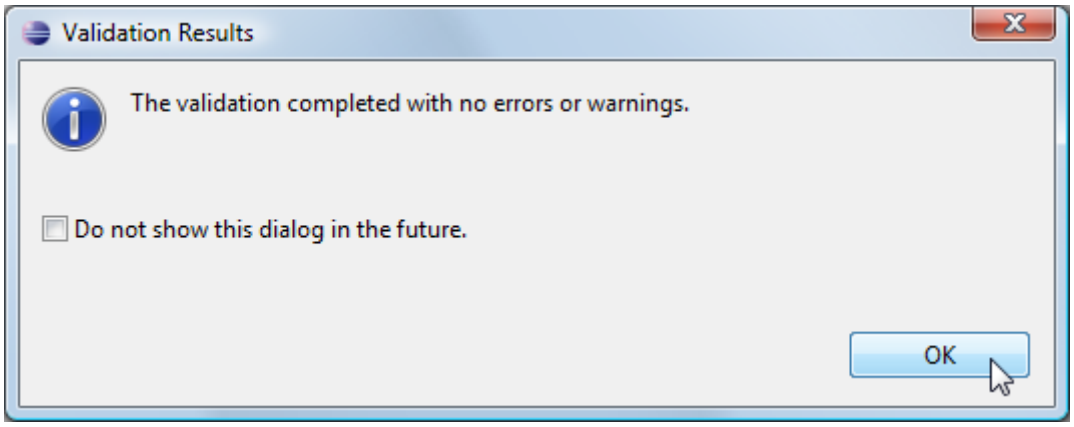
Nun kann das XML-File geschrieben werden.



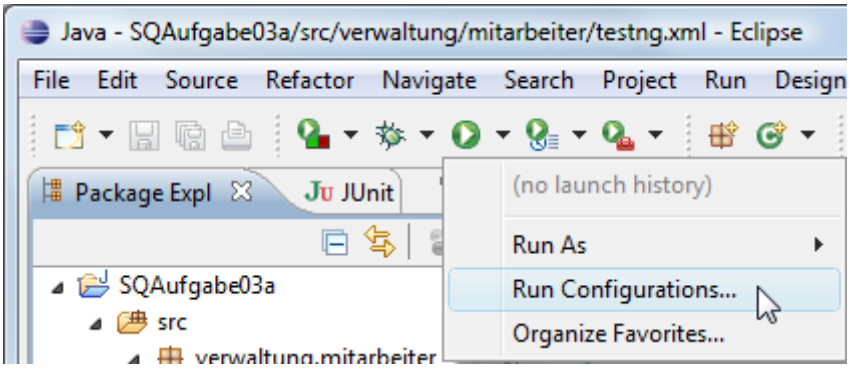
Mit einem Rechtsklick auf der Datei textng.xml kann man die Datei validieren und ihr ordnungsgemäße Form überprüfen.



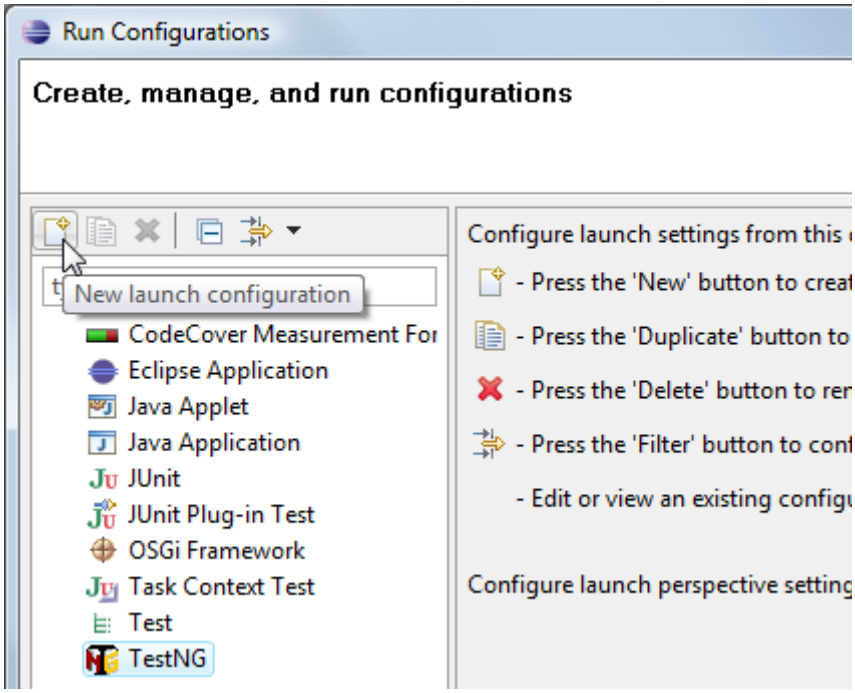
Man erhält entweder Fehlerinformationen, die auch im Editor markiert werden, oder die Nachricht, dass die Datei in Ordnung ist. Diese Meldung sollte man nicht abschalten, da man sonst keine klare Meldung bekommt, dass die Prüfung überhaupt stattgefunden hat.



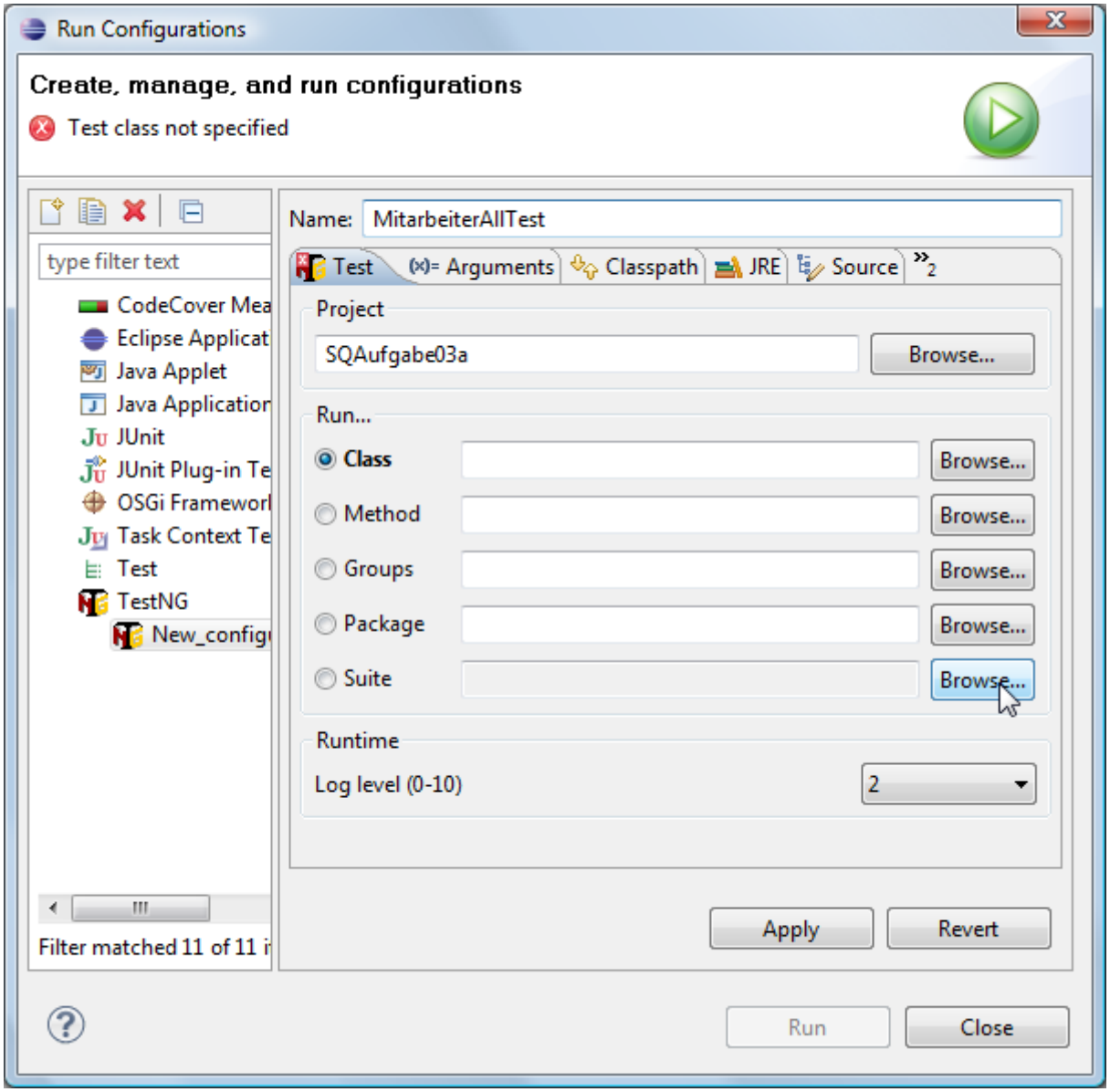
Um die zur XML-Datei gehörenden Tests auszuführen, gibt es wieder verschiedene Varianten. Wenn sich testng.xml im aktiven Editorfenster befindet, gibt es eine Möglichkeit durch das Drücken des Pfeils nach unten neben dem „Run“-Button  eine neue Startkonfigurationsdatei über „Run Configurations...“ zu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



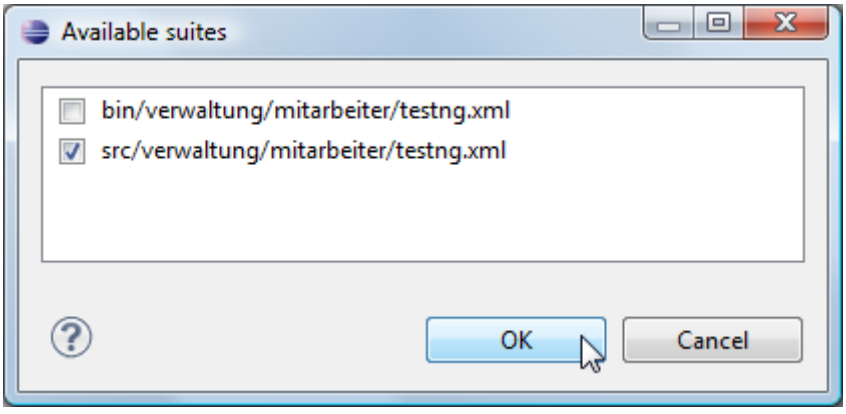
Hier wählt man zunächst rechts TestNG und drückt dann oben den Knopf „New launch Configuration“.



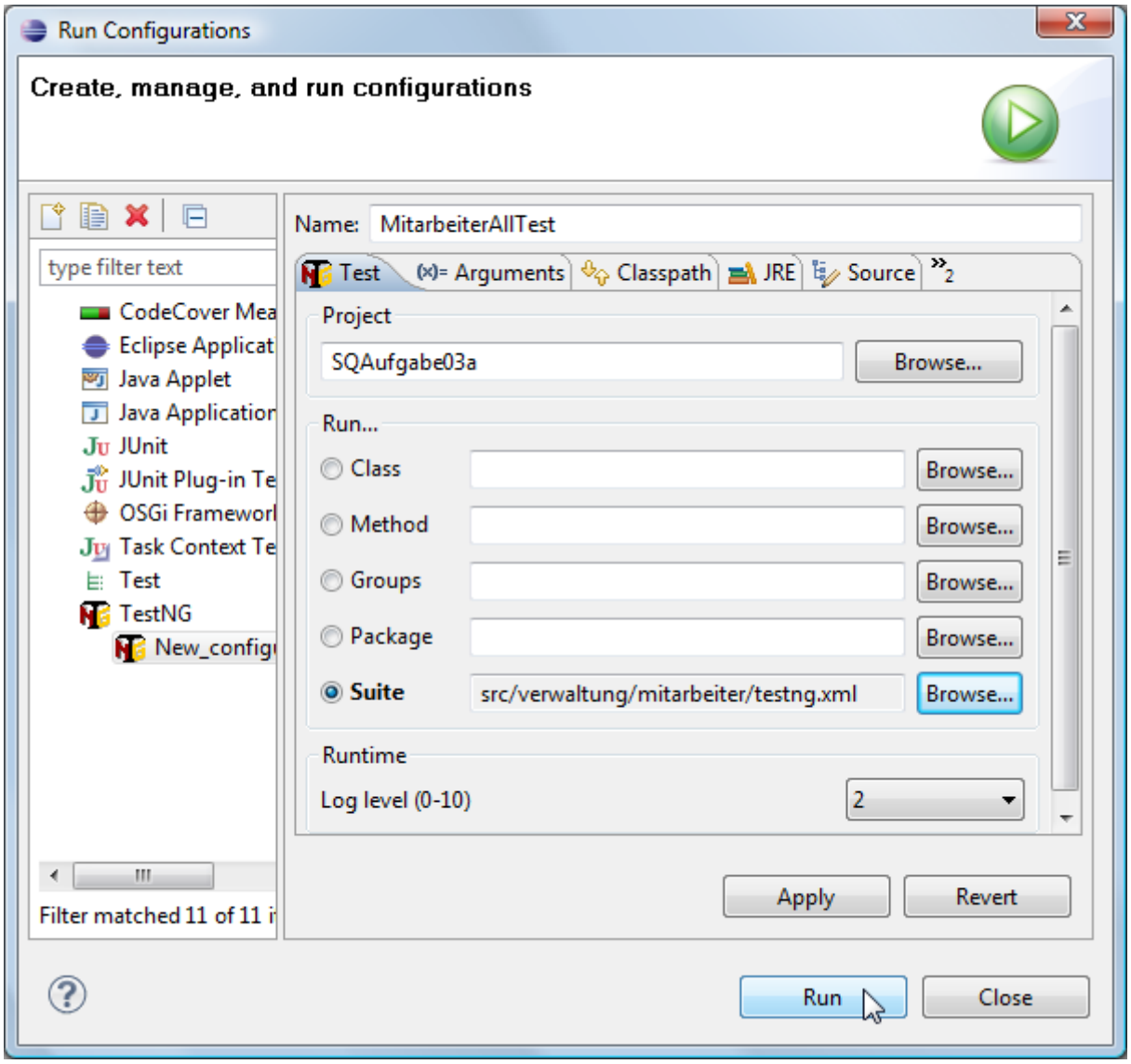
Man kann der Konfiguration unter „Name“ einen Namen geben“. Im mittleren Fenster unter „Run“ kann man dann zusammenstellen welche Tests laufen sollen. Man erkennt die sehr flexiblen Möglichkeiten hier Tests aus verschiedenen Quellen zusammen zu stellen. Um Elemente auszuwählen, wird rechts der Knopf „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



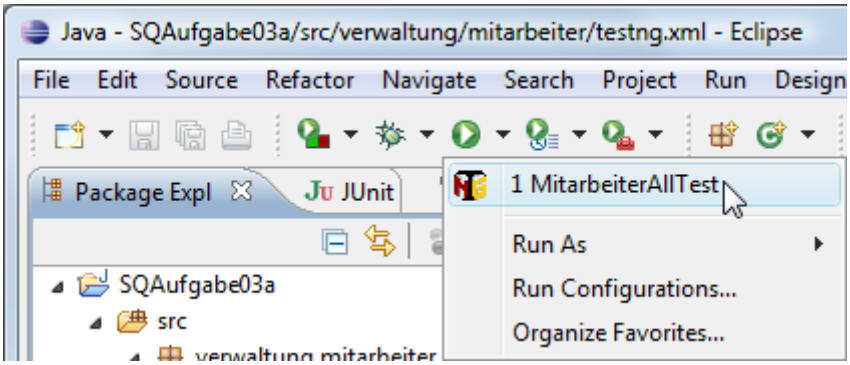
Man kann dann die interessanten testng.xml-Dateien mit einem Haken links auswählen. In den bin-Ordnern stecken Kopien der XML-Datei, deshalb müssen dies nicht angewählt werden. Die Auswahl wird dann mit „OK“ abgeschlossen.



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.

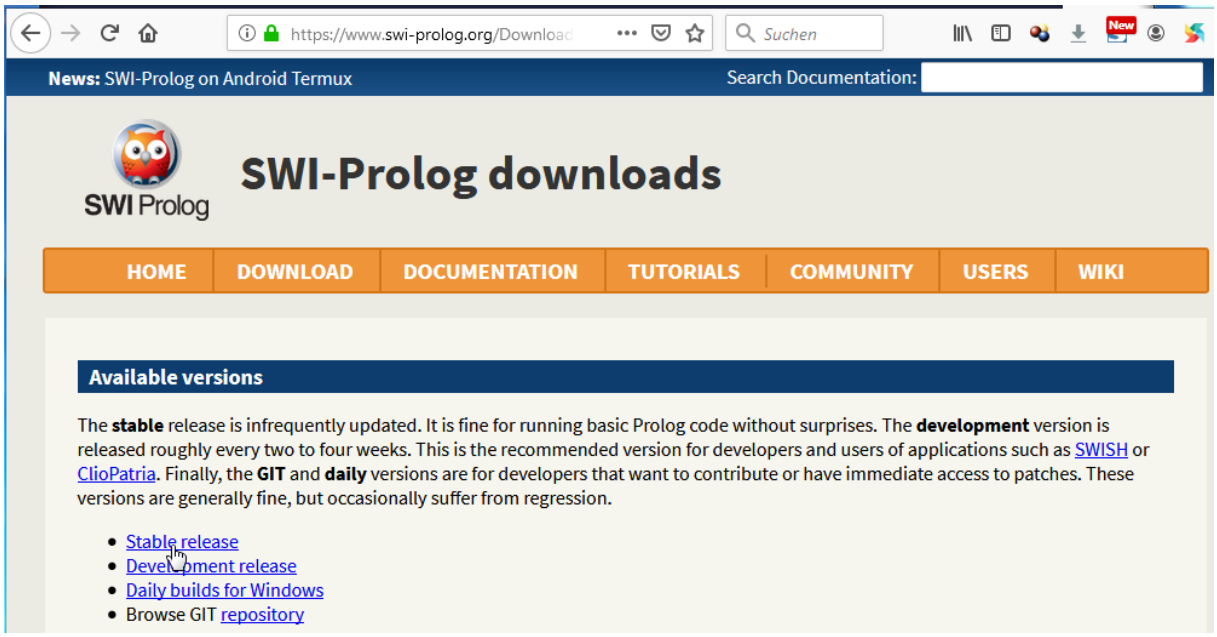


16 Prolog mit Eclipse (4.11.0)

Prolog ist eine logikbasierte Programmiersprache, die den deklarativen Stil nutzt. Dabei werden Fakten und Regeln deklariert, die dem Programm entsprechen. Die eigentlichen Aufgaben werden als Ziele definiert, deren mögliche Erfüllung vom Prolog-System anhand der Fakten und Regeln geprüft wird. SQL ist ebenfalls eine deklarative Sprache, mit der beschrieben wird, was gesucht wird. Der SQL-Interpreter sucht dann nach Lösungen.

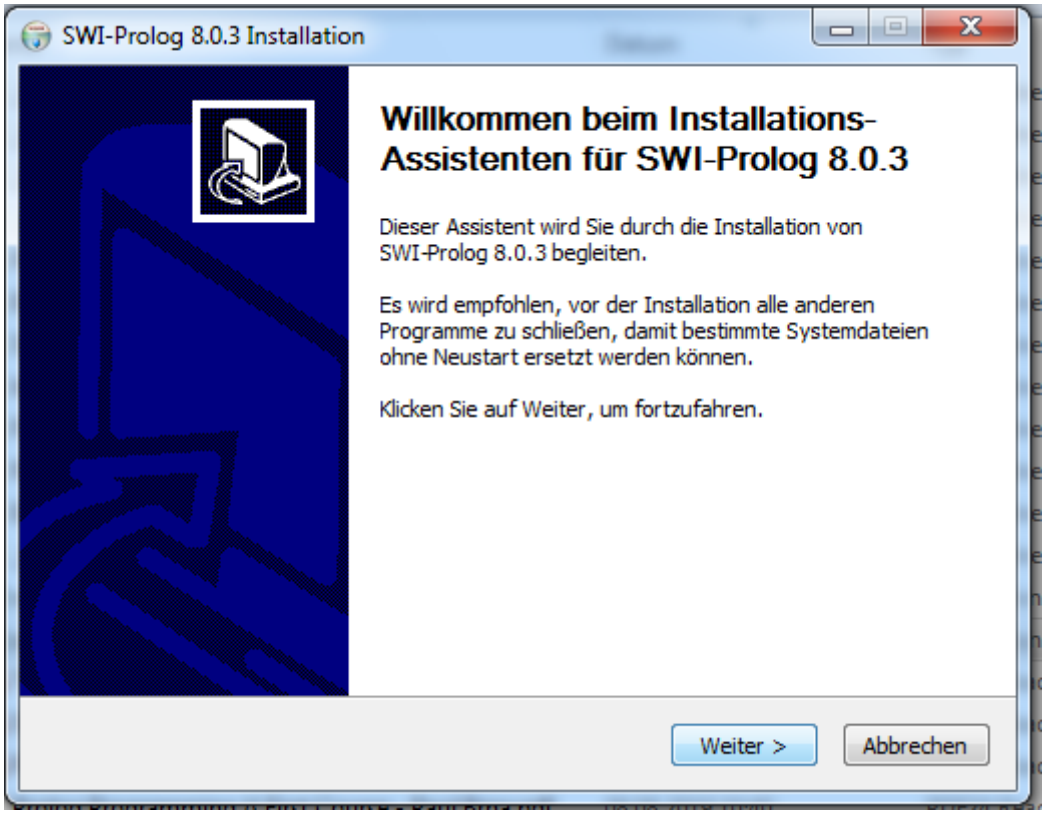
Hier wird gezeigt, wie SWI-Prolog installiert und Eclipse als Entwicklungsumgebung zu nutzen sind. Wieder werden nur erste Schritte gezeigt, die anderweitig zu vertiefen sind.

SWI-Prolog kann über die Seite https://www.swi-prolog.org/Download.html geladen werden. Es wird auf „Stable release“ geklickt.

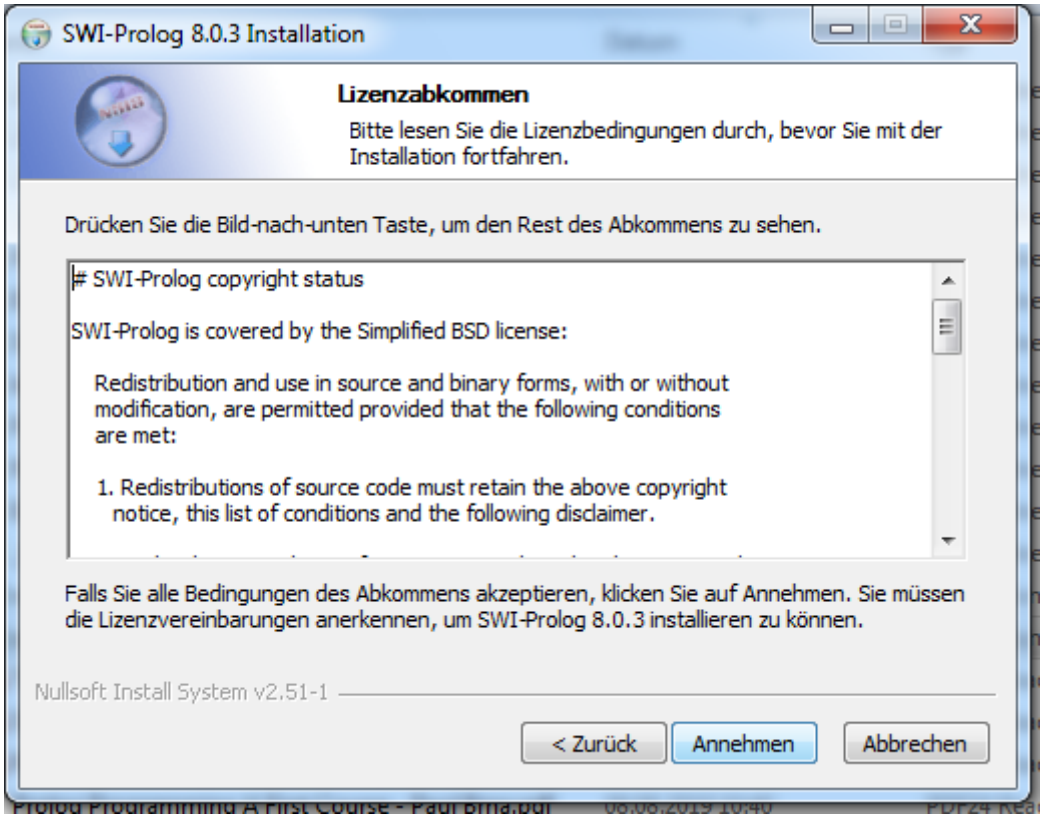


Es wird auf den zum eigenen Betriebssystem passenden Download geklickt, der dann automatisch startet.

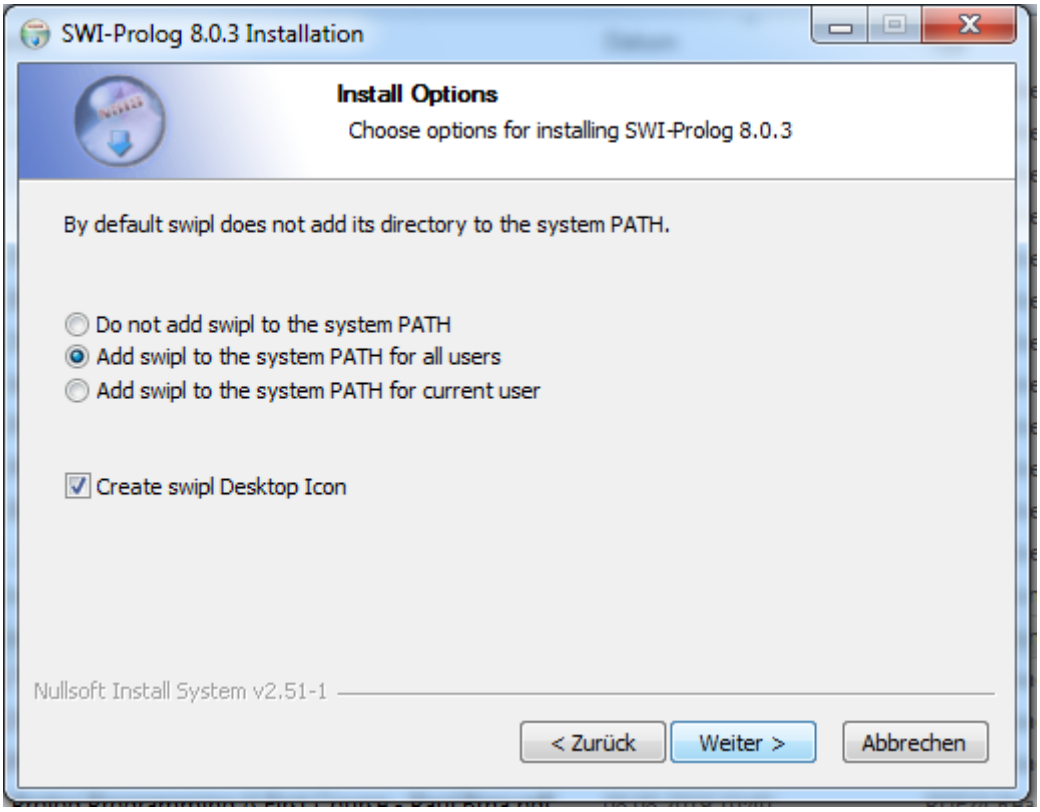
Die Installation startet mit einem Doppelklick auf der Datei und dann auf „Weiter>“



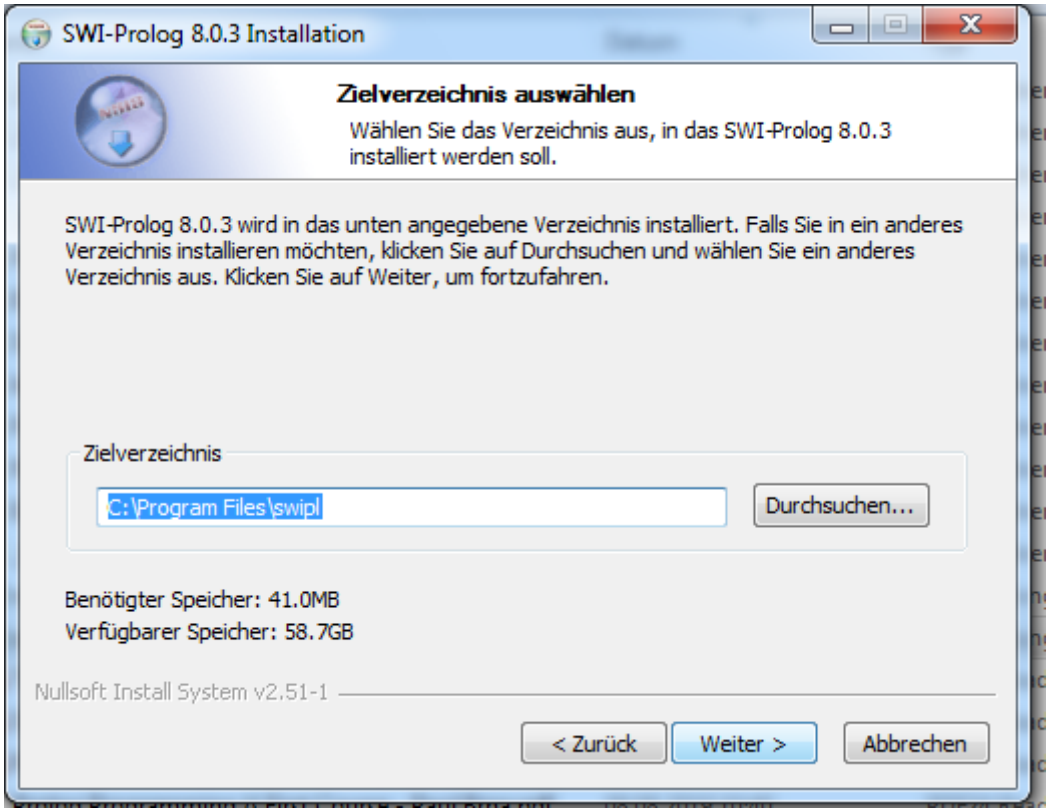
Dier Lizenz muss mit „Annehmen“ zugestimmt werden.



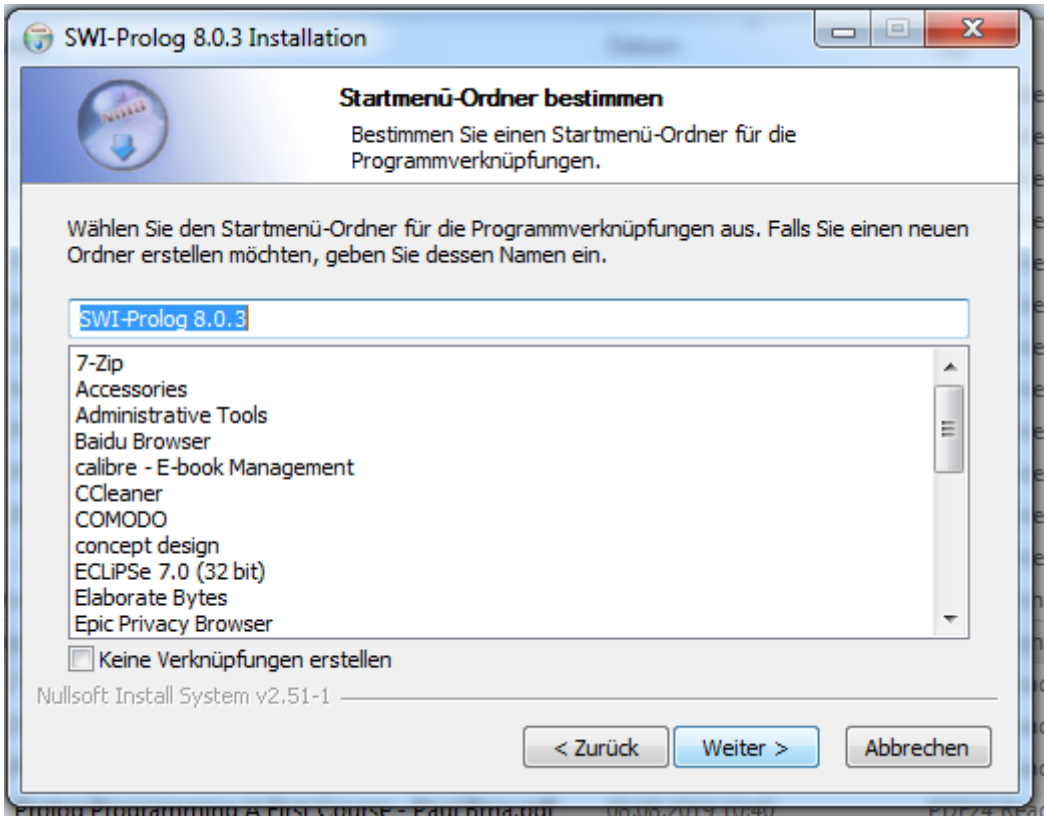
Der PATH-Eintraf für swipl kann jetzt hier gesetzt werden oder muss später manuell erfolgen. Es wird danach „Weiter>“ geklickt.



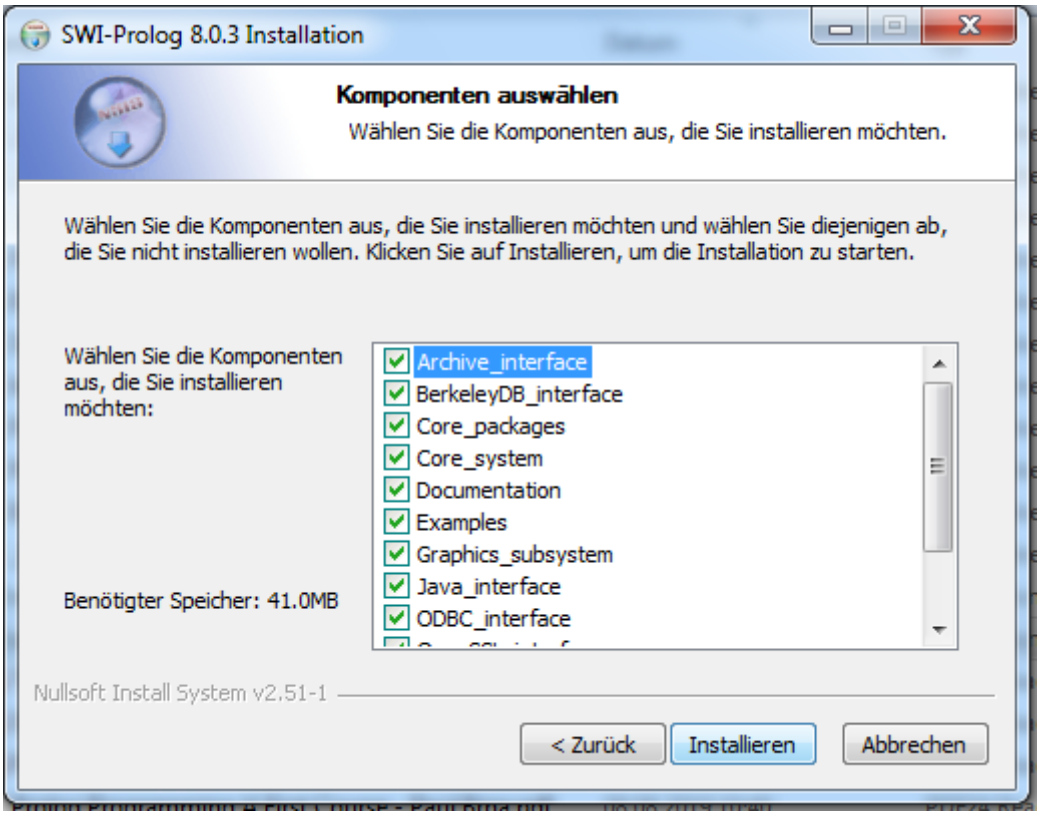
Das Installationsverzeichnis ist frei wählbar, es wird auf „Weiter >“ geklickt.



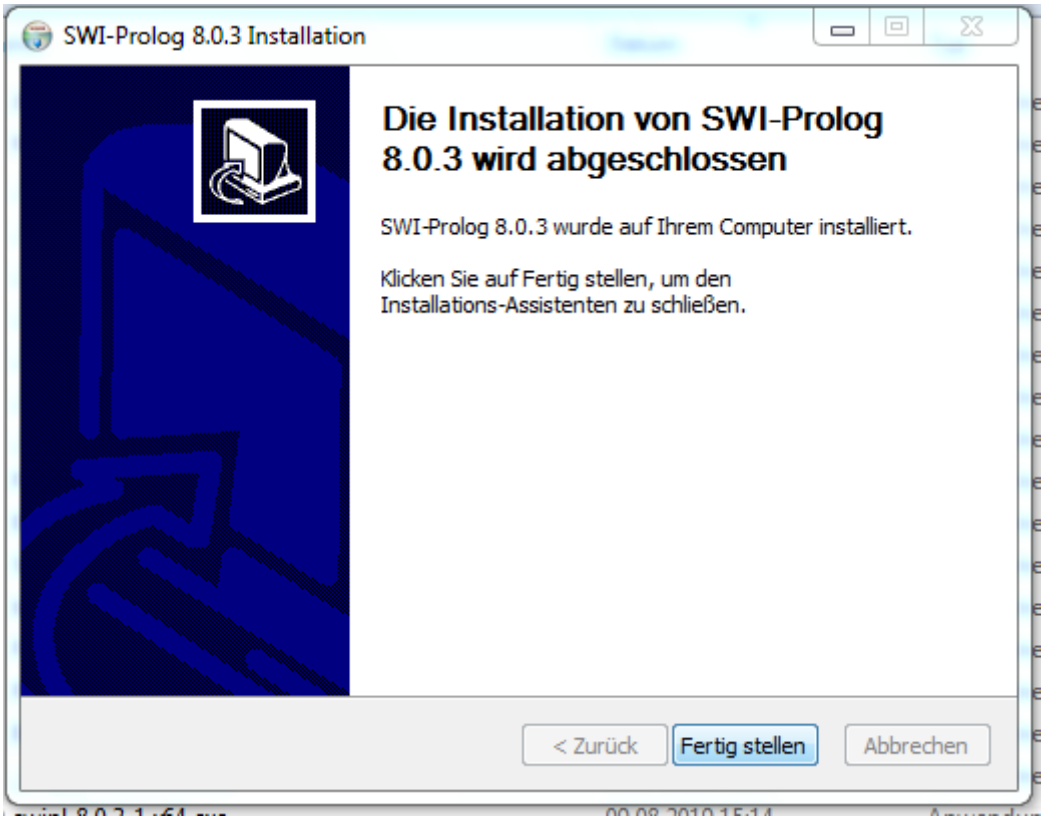
Es wird wieder „Weiter >“ geklickt.



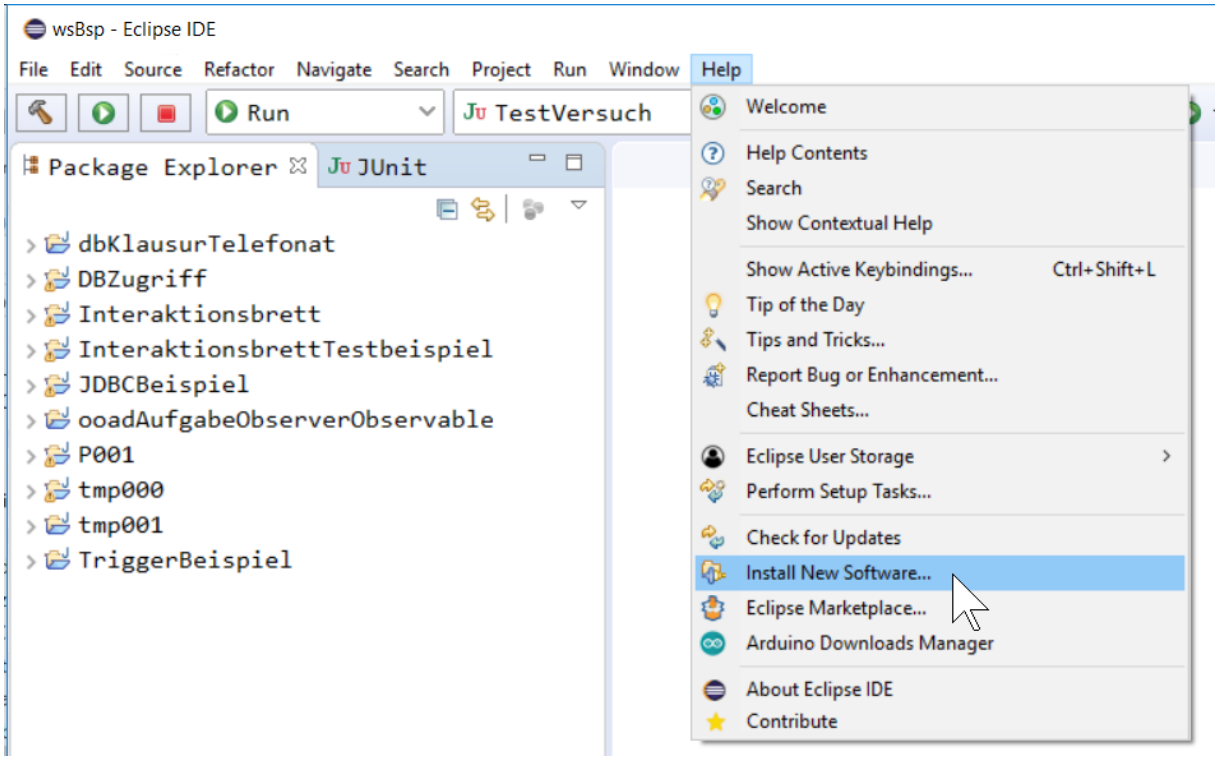
Die Installation wird mit „Installieren“ abgeschlossen.



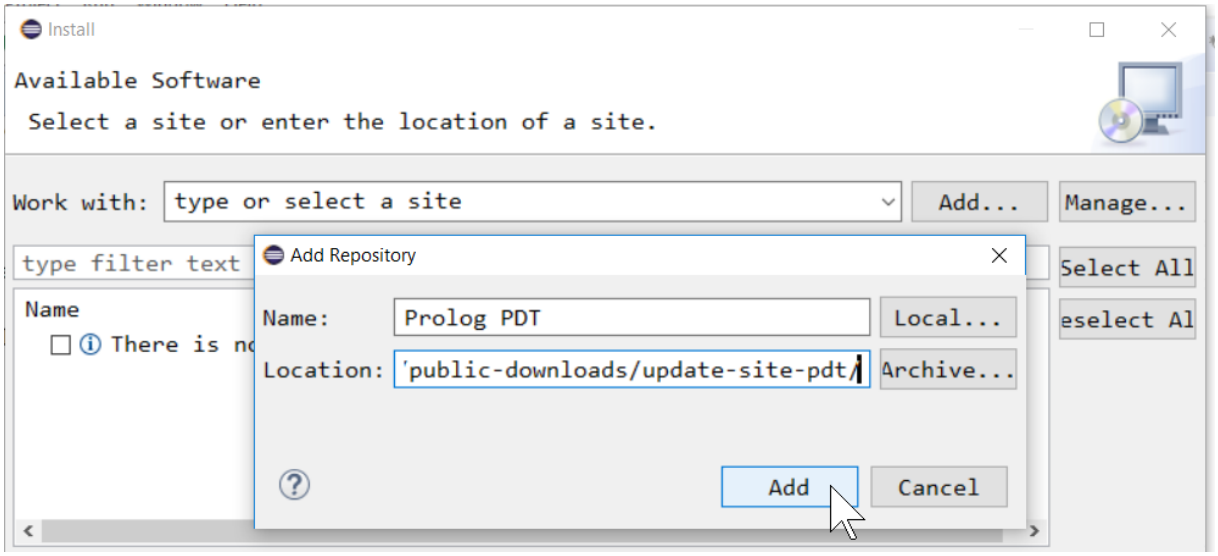
Die erfolgreiche Installation wird mit „Fertig stellen“ abgeschlossen.



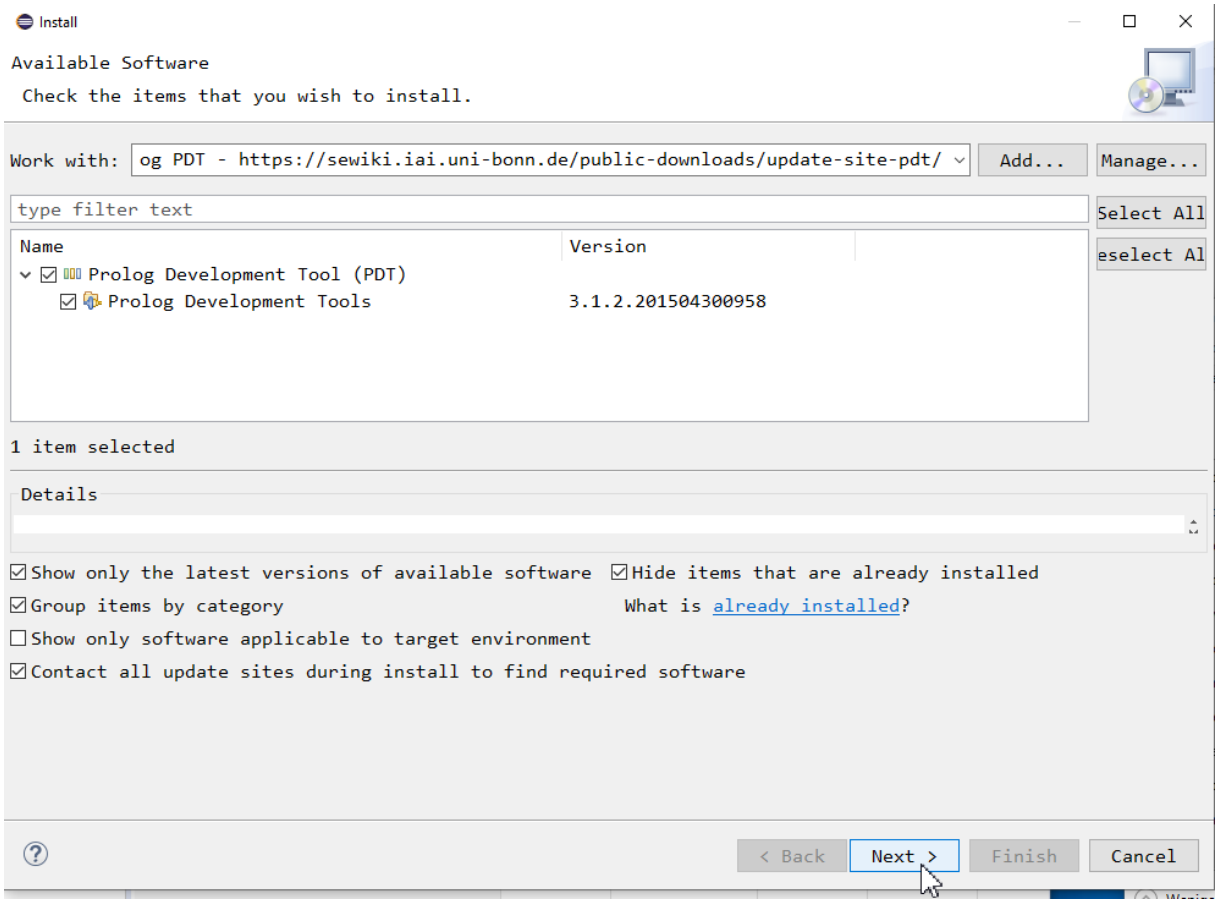
Zur Installation der Eclipse.Erweiterung für Prolog wird „Help > Install New Software…“ ausgewählt.



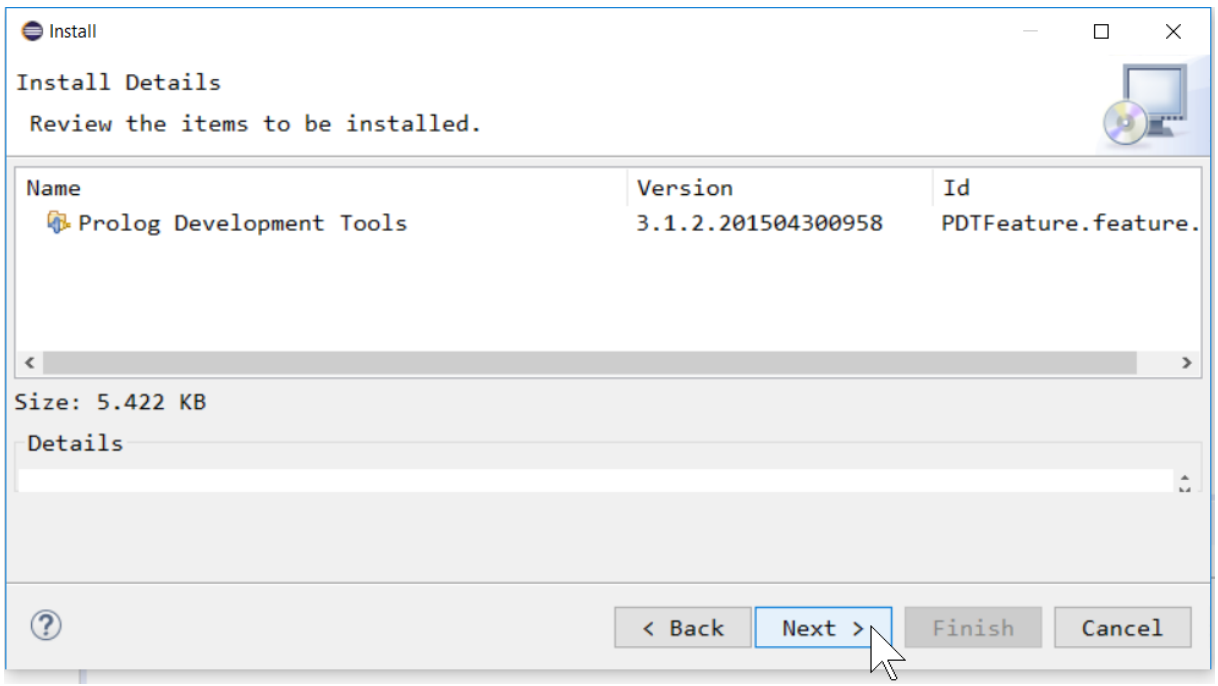
Unter „Add…“ wird als Name PDT und der Link https://github.com/pdt-git/public/raw/updatesite/pdt.updatesite/target/repository oder alternativ https://sewiki.iai.uni-bonn.de/public-downloads/update-site-pdt/ eingetragen.



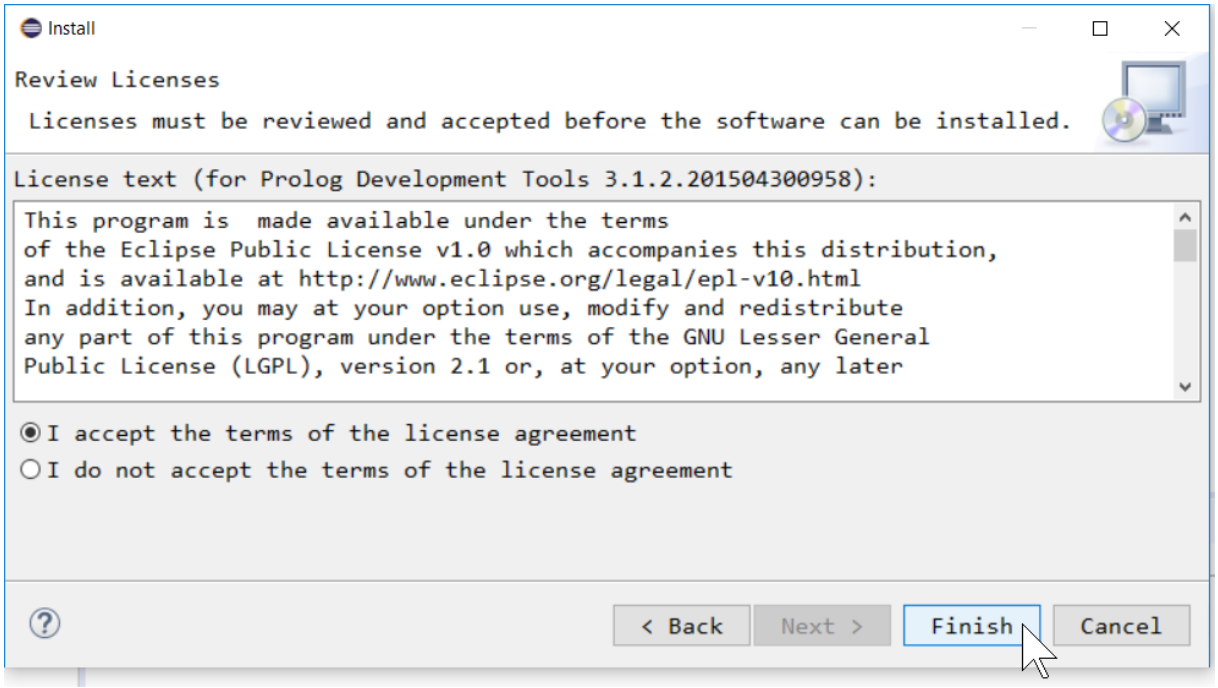
Es wird rechts „Select All“ und dann „Next >“ angeklickt.



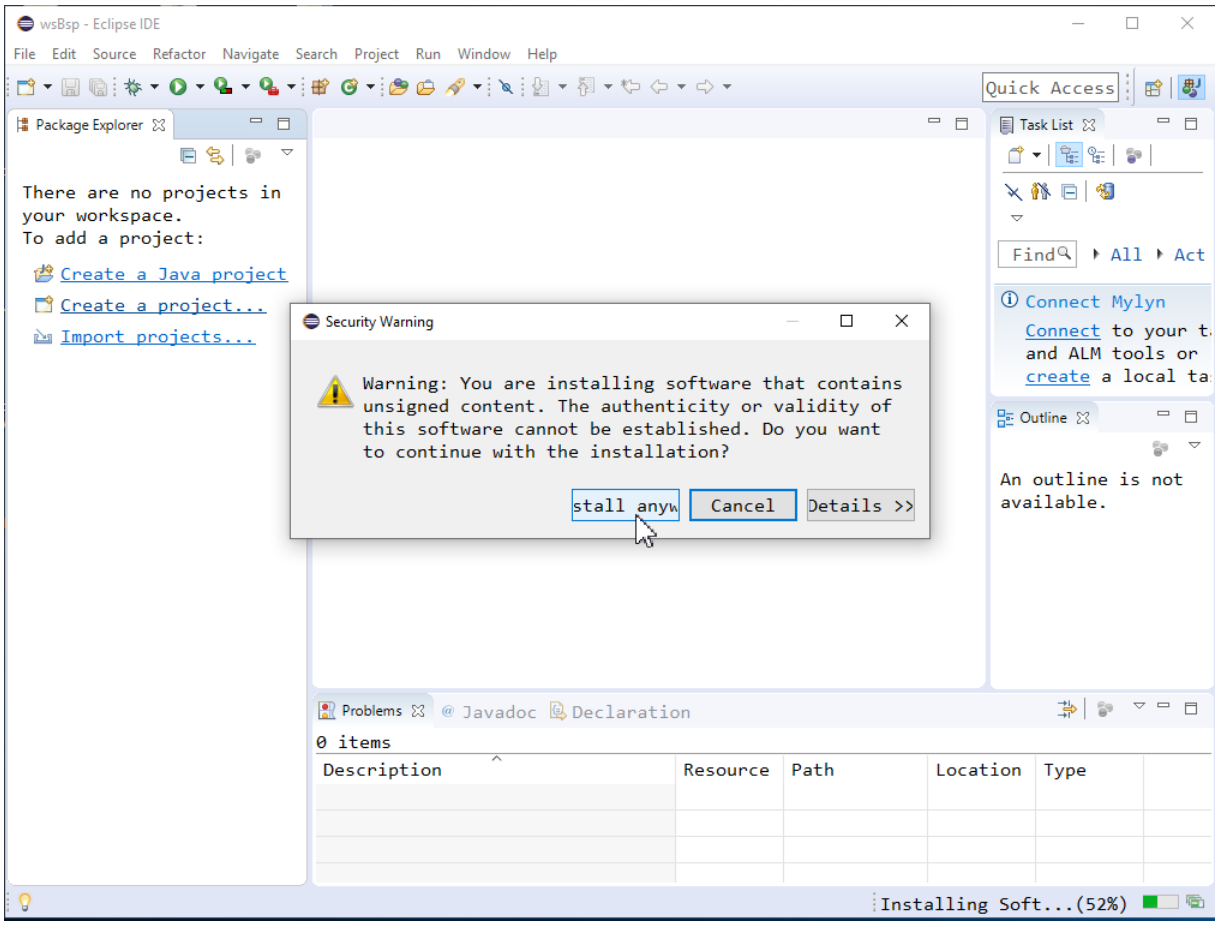
Es wird wieder „Next >“ angeklickt.



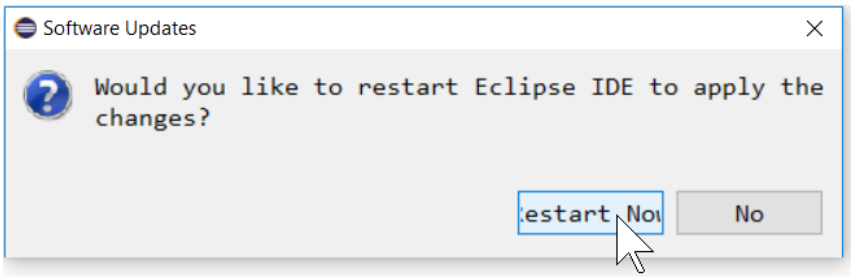
Die Lizenz wird gelesen, mit „I accept“ bestätigt und „Finish“ geklickt.



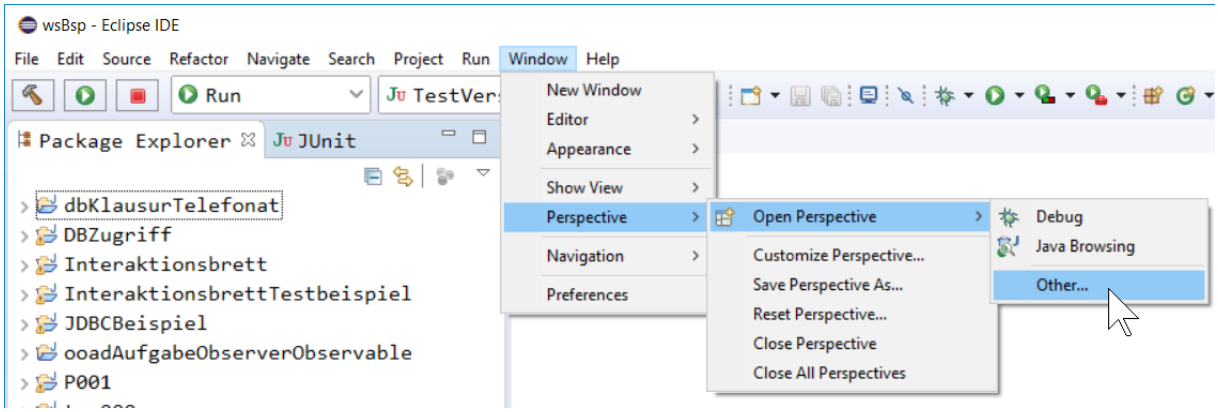
Die Installation wird mit „Install anyway“ fortgesetzt.



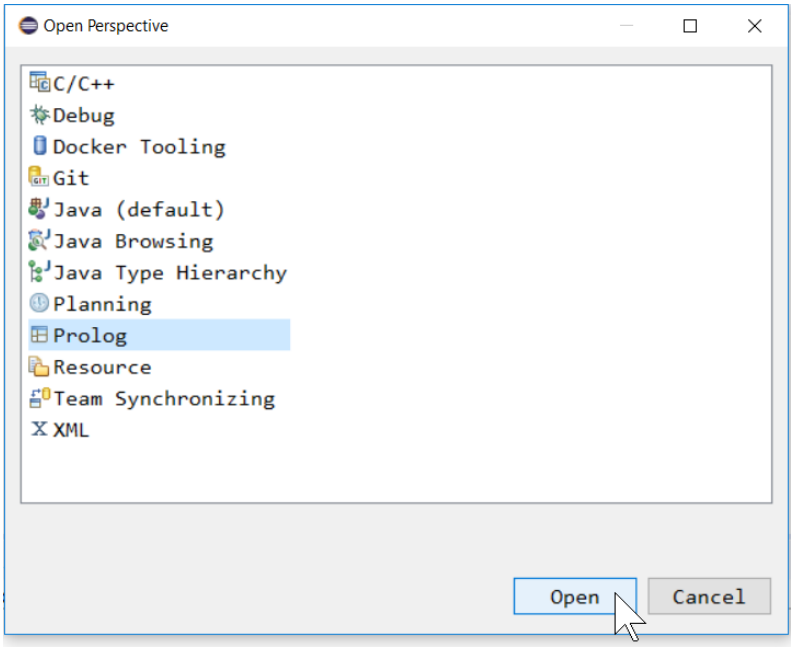
Der vorgeschlagene Neustart wird mit einem Klick auf „Restart Now“ durchgeführt.



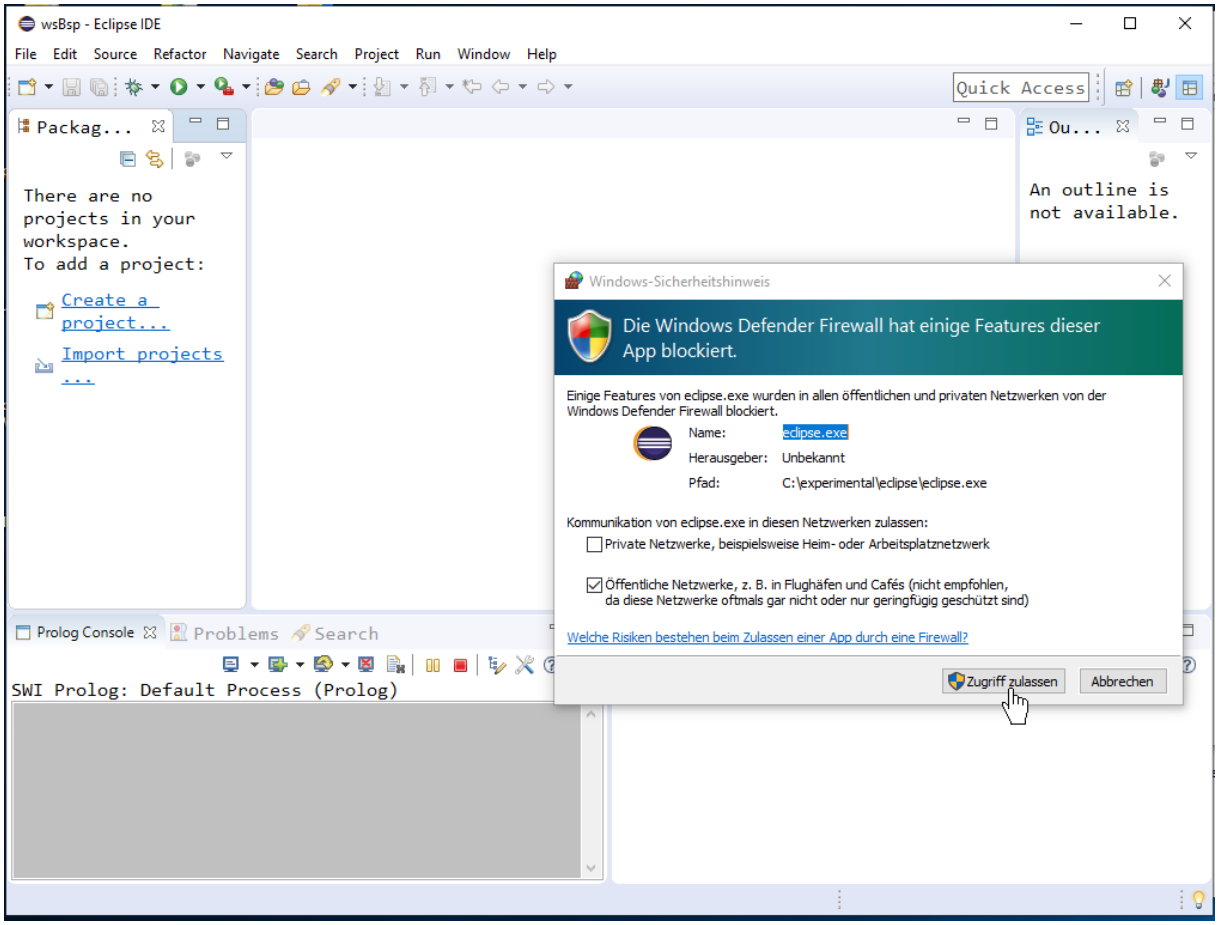
Nach dem Neustart wird „Window > Perspective > Open Perspective > Other“ ausgewählt.



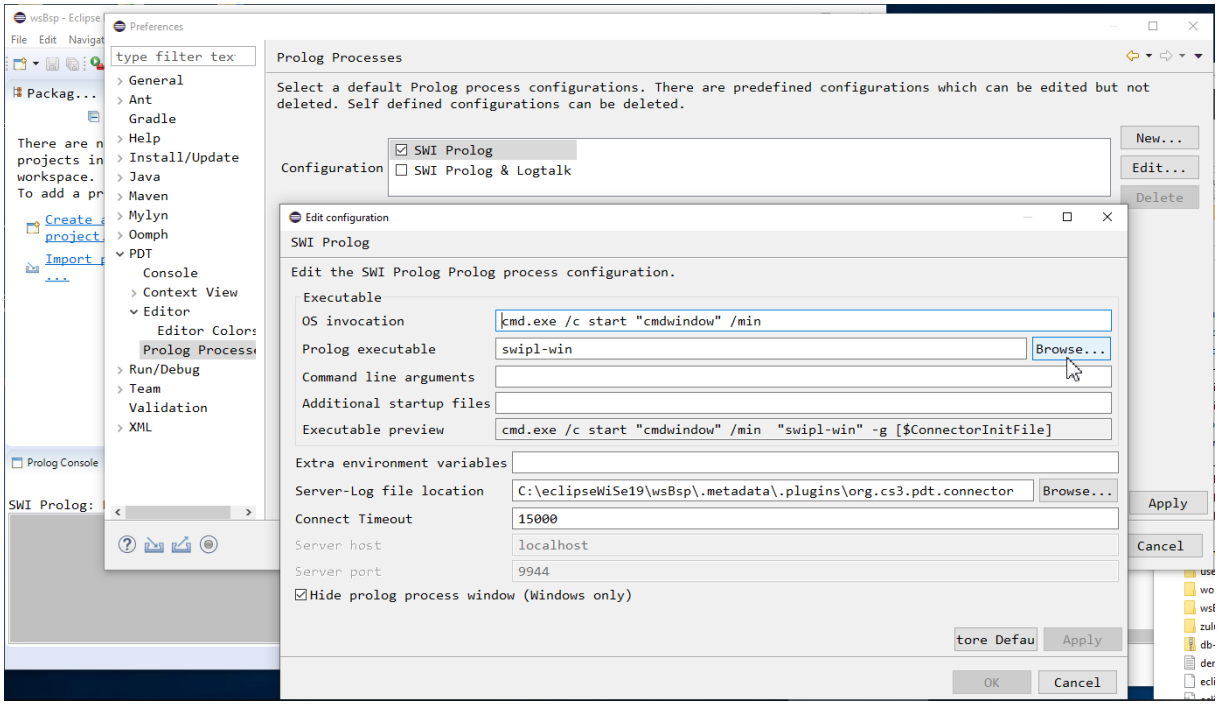
Es wird „Prolog“ ausgewählt und „Open“ geklickt.



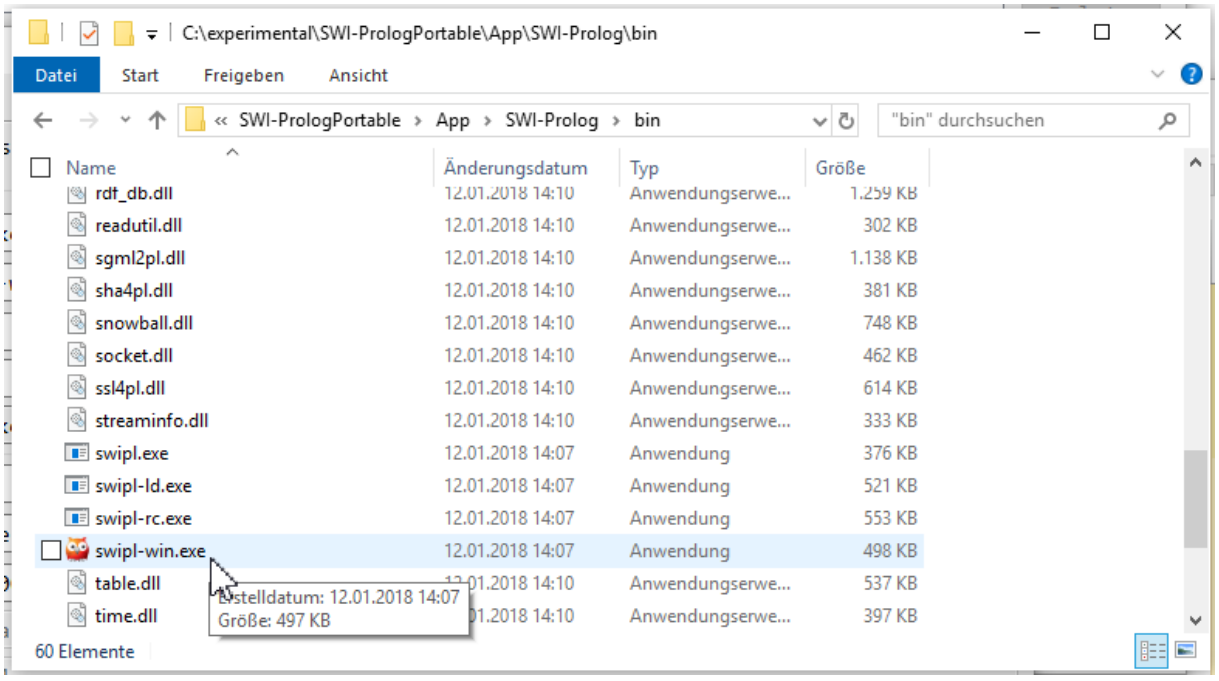
Eventuell muss eine Sicherheitsabfrage bestätigt werden.



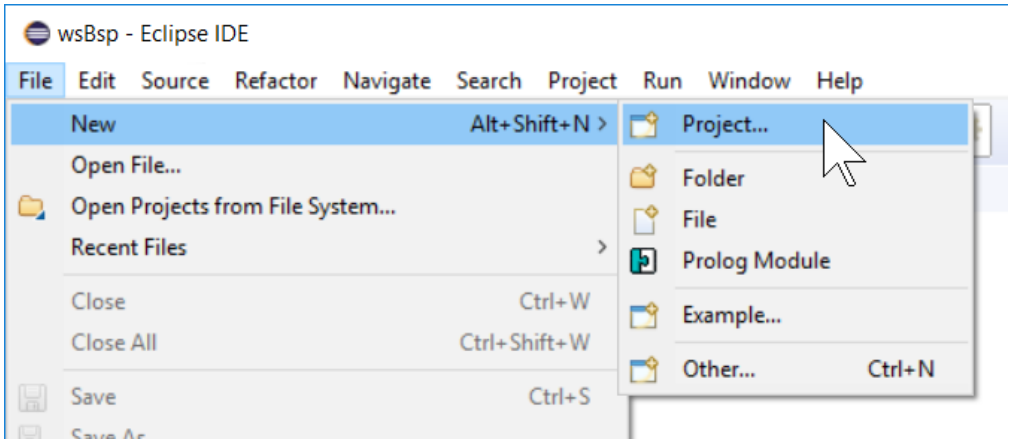
Unter „Window > Preferences“ wird links „PDT > Prolog Processes“ ausgewählt, dann für die Auswahl „SWI Prolog“ auf „Edit…“ geklickt, um dann das „Prolog-Executable“ über „Browse“ einzustellen.



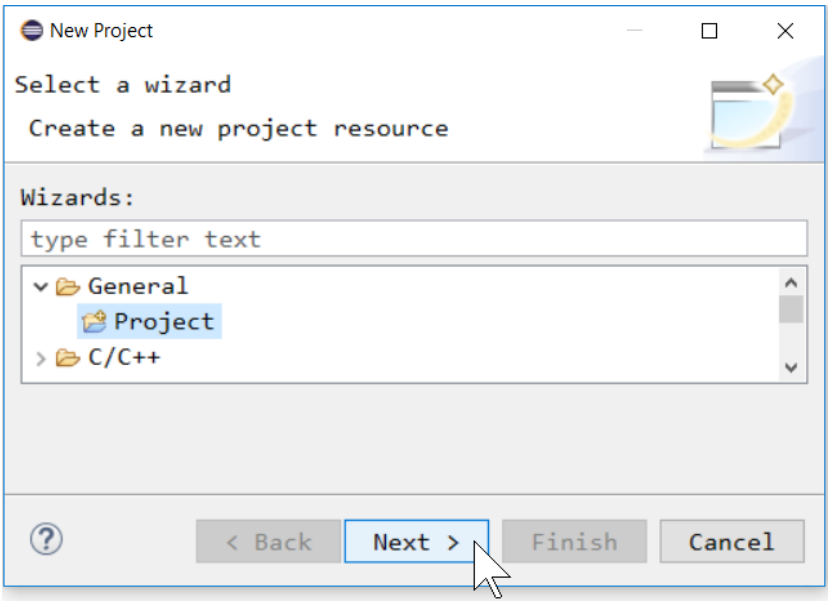
Das Executable swipl-win.exe befindet sich im bin-Verzeichnis der Prolog-Installation.



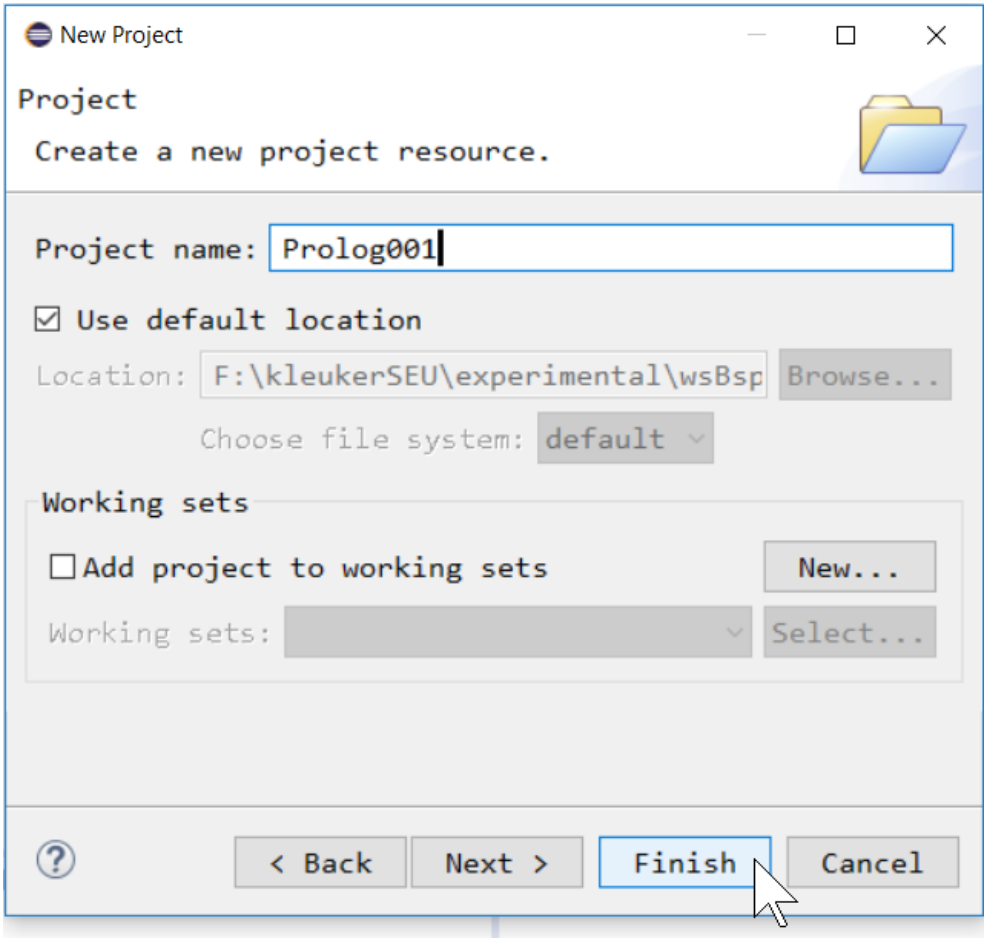
Ein neues Prolog-Projekt wird als normales Eclipse-Projekt über „File > New > Project“ erzeugt.



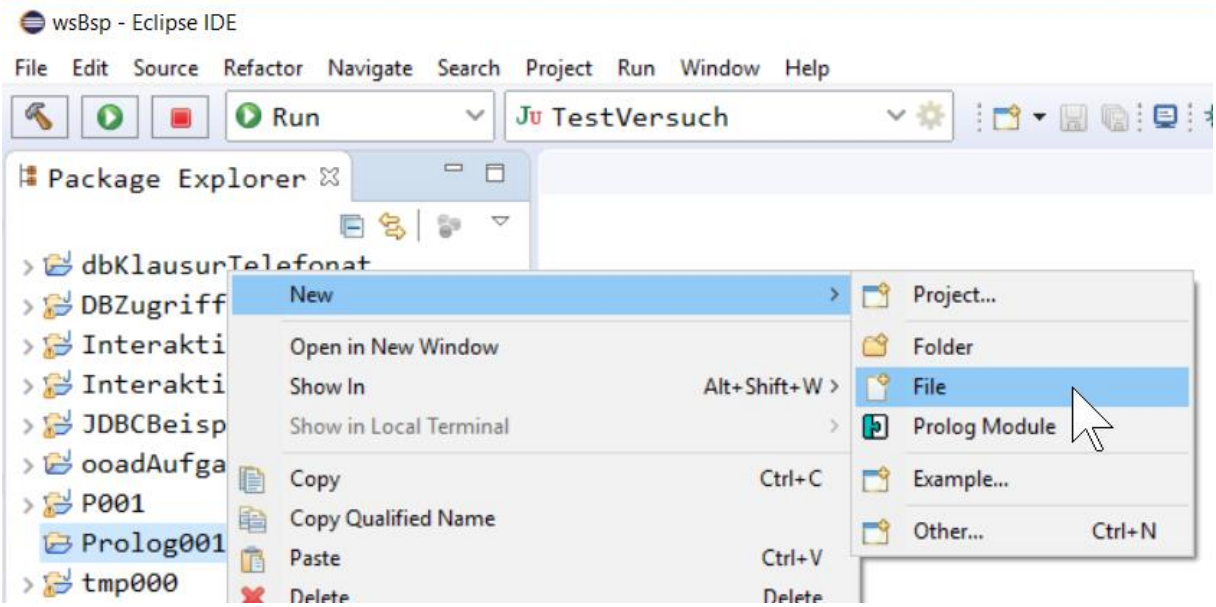
Es wird „General > Project“ gewählt und „Next >“ geklickt.



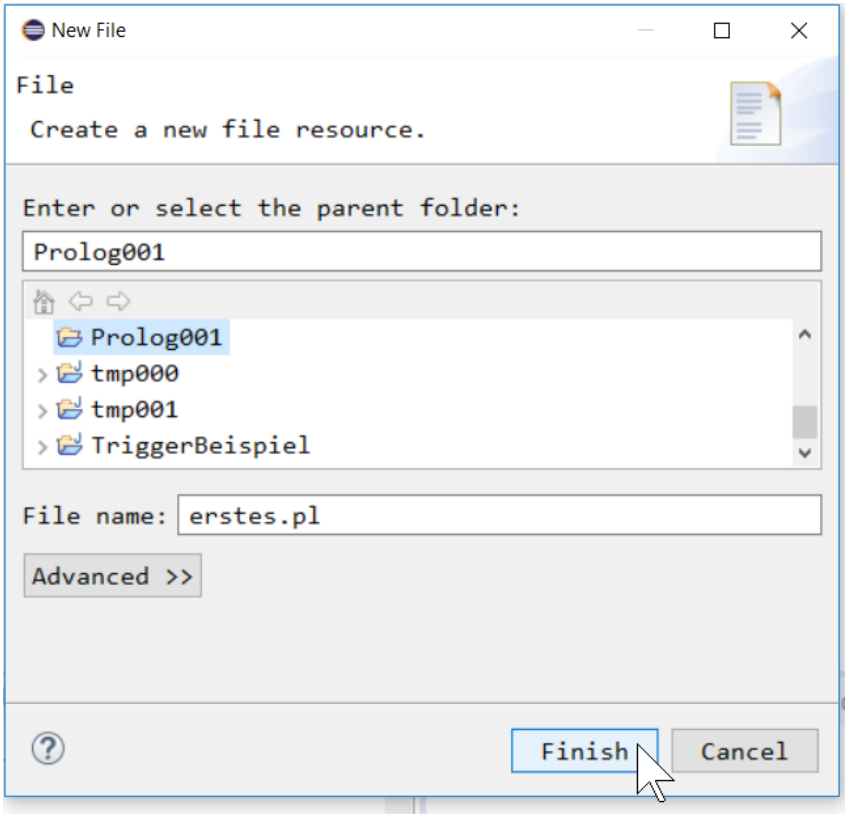
Es wird ein Projektname angegeben und „Finish“ geklickt



Es wird ein Rechtsklick auf dem Projekt gemacht und „New > File“ ausgewählt.

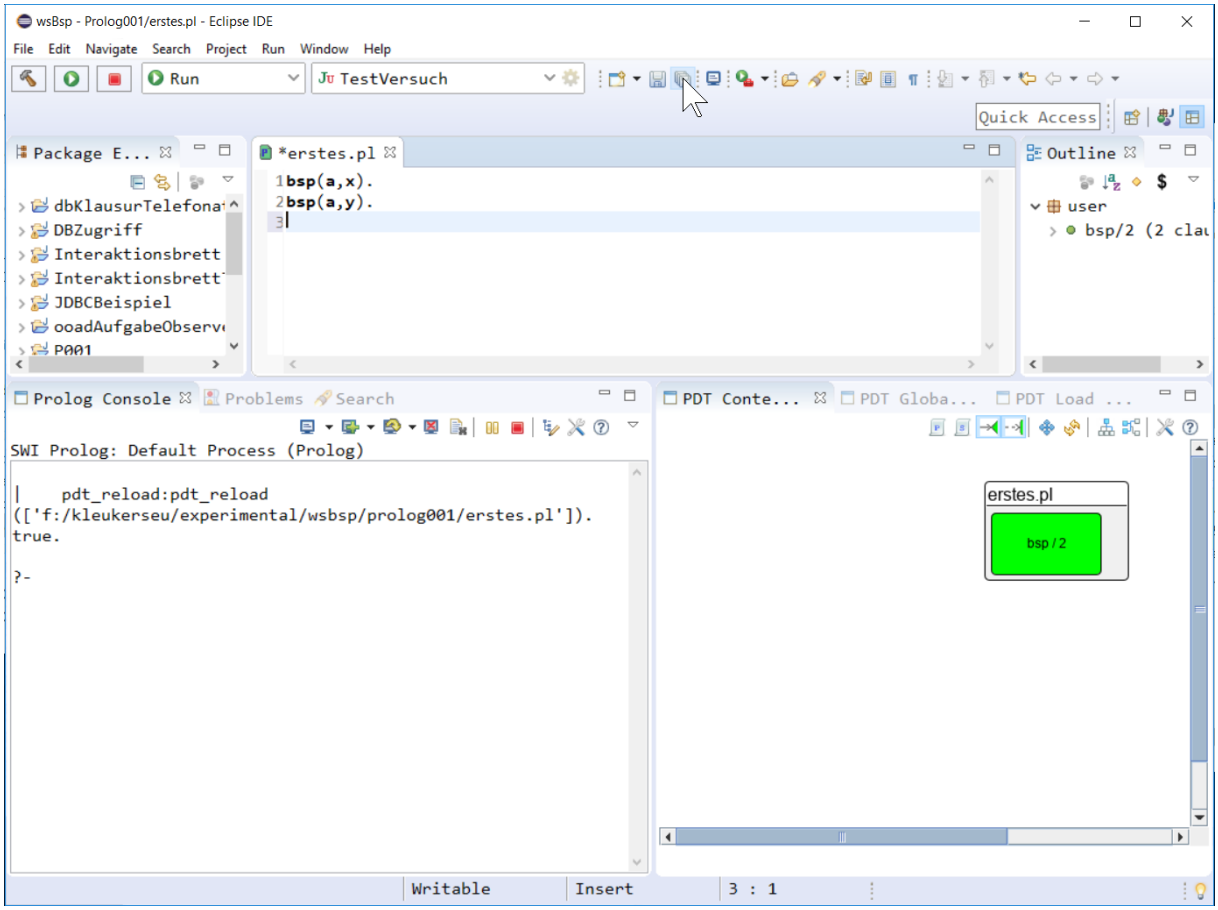


Der Filename endet mit „.pl“.



Werden erste Fakten eingegeben und auf „Speichern“ in der Kopfzeile geklickt, wird die Datei links-unten automatisch in das Prolog-System geladen.

```
bsp(a,x).
bsp(a,y).
```



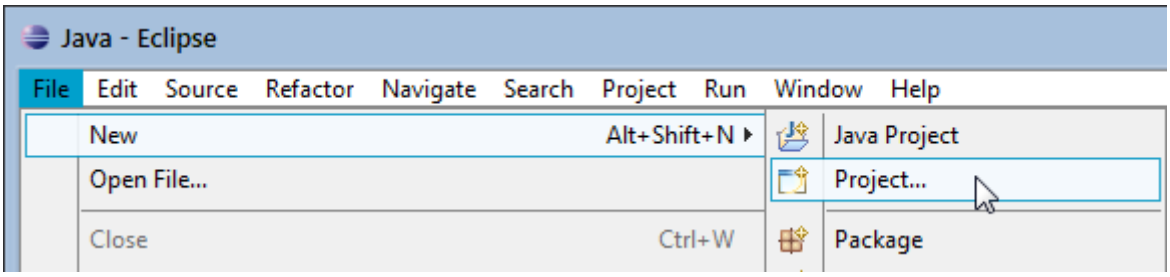
In dem Fenster links-unten können dann Anfragen gestellt werden, bei mehreren Ergebnissen muss zwischenzeitlich ein Semikolon „ ; “ (oder) eingegeben werden.

```
pdt_reload:pdt_reload
(['f:/kleukerseu/experimental/wsbsp/prolog001/erstes.pl']).
|    % f:/kleukerseu/experimental/wsbsp/prolog001/erstes
compiled 0.02 sec, 0 clauses
true.

?- bsp(a,X).
X = x ;
X = y.

?-
```

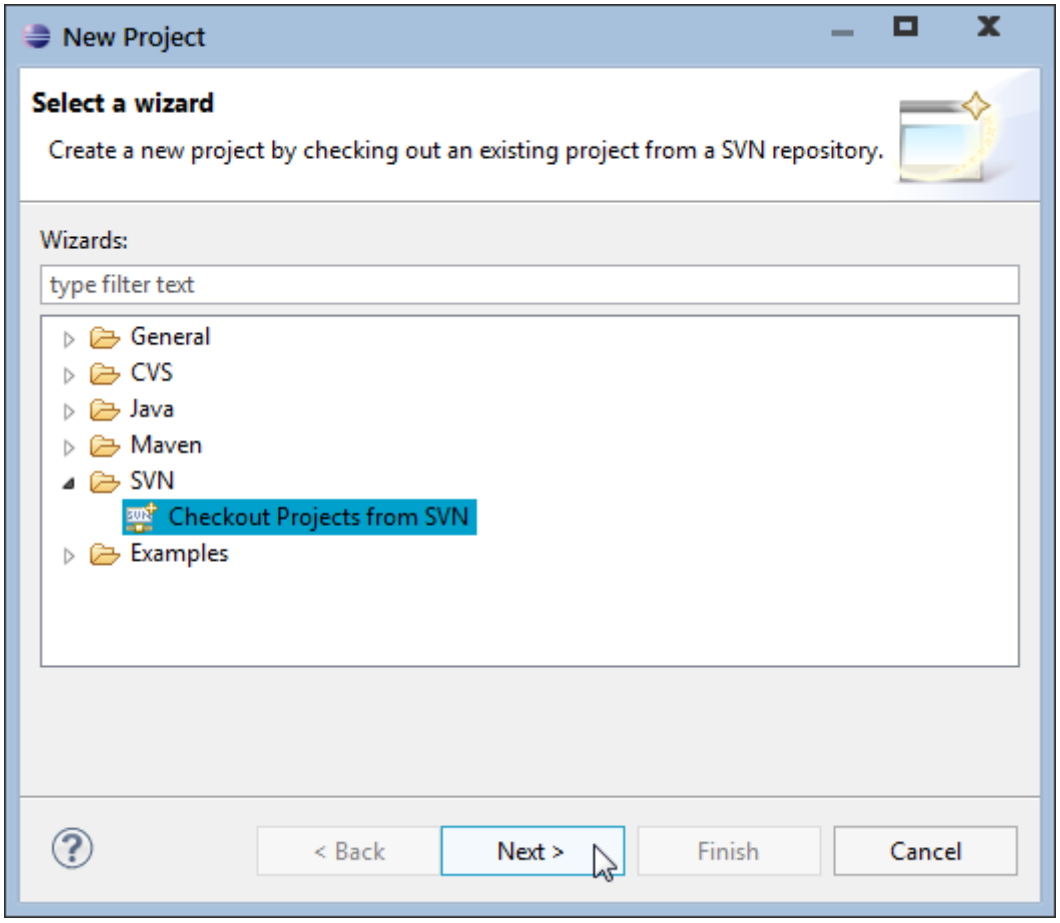
17 Versionsmanagement mit Subclipse (4.3.1)

Mit Subclipse kann man auf Subversion-Verzeichnisse zugreifen. Die Installation von Subclipse wird im Kapitel „8 Installation von Plugins über Marketplace“ beschrieben. In diesem Abschnitt wird die Einrichtung einer Verbindung und damit die Nutzung eines existierenden Projekts beschrieben.

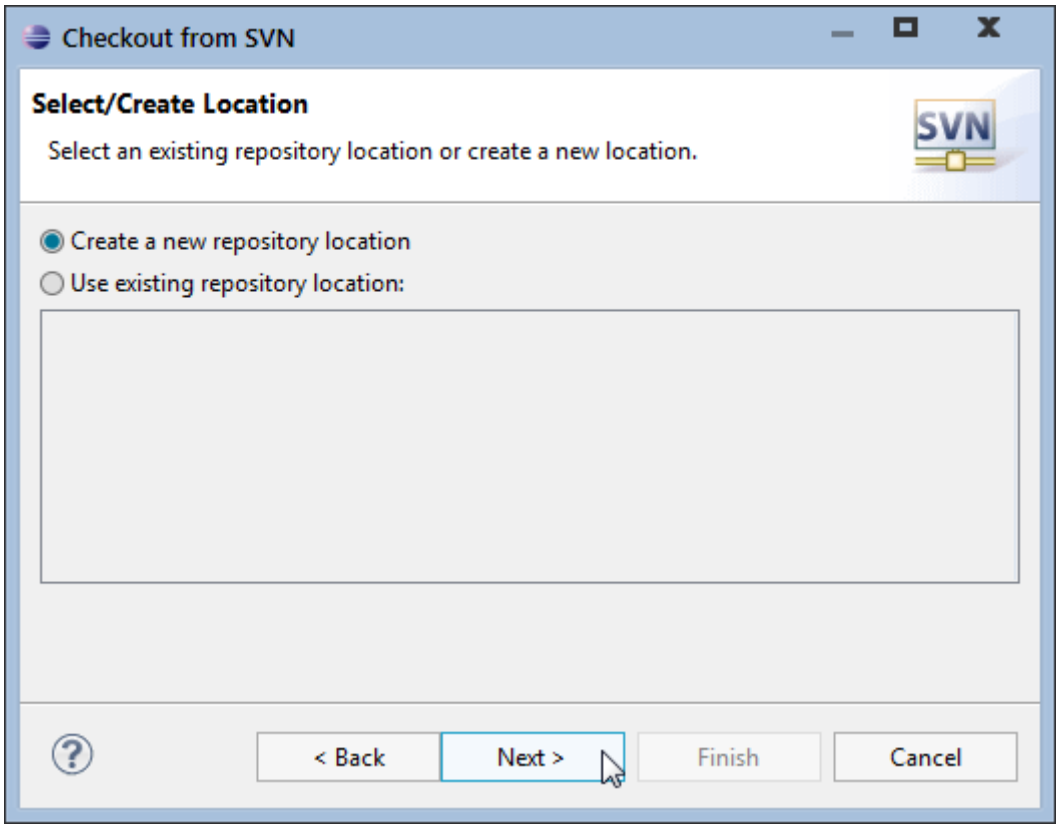
Es wird ein neues Projekt angelegt, das allerdings von einem SVN-Verzeichnis eingelesen werden soll.



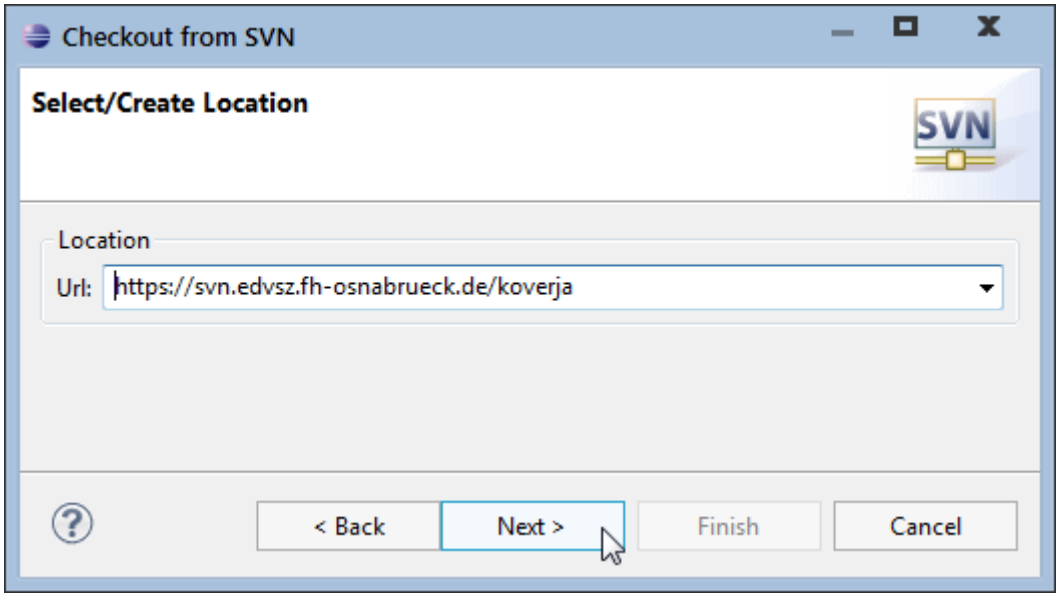
Es wird dann SVN aufgeklappt und „Checkout Projects from SVN“ ausgewählt und „Next>“ gedrückt.



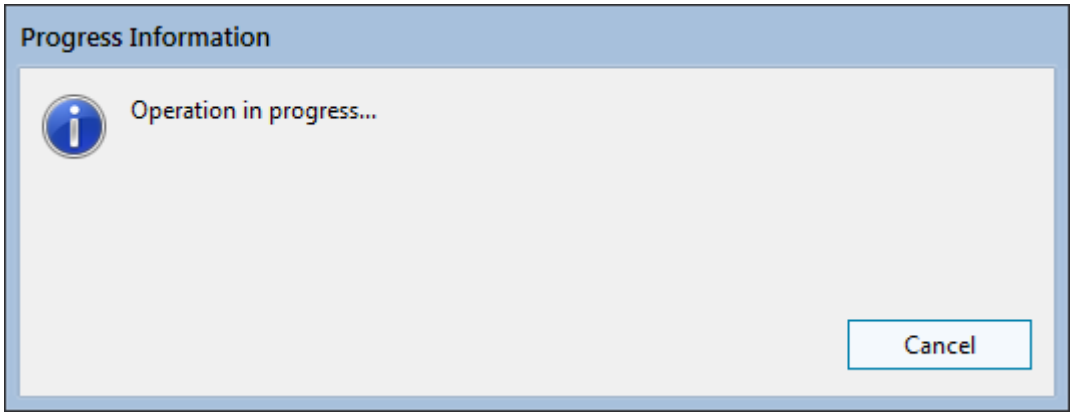
Abhängig davon, ob schon Subversion-Verzeichnisse genutzt werden, muss man sich für ein „Location“ entscheiden. Hier soll ein Verzeichnis genutzt werden, das in Eclipse noch nicht bekannt ist.



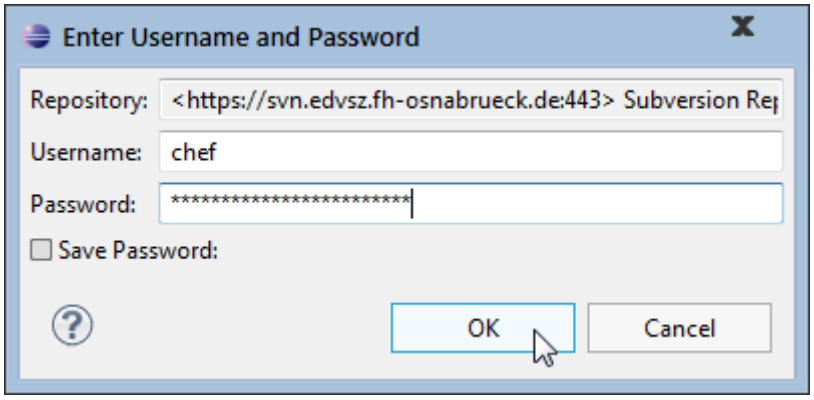
Hier muss der Pfad zum Subversion-Verzeichnis angegeben werden.



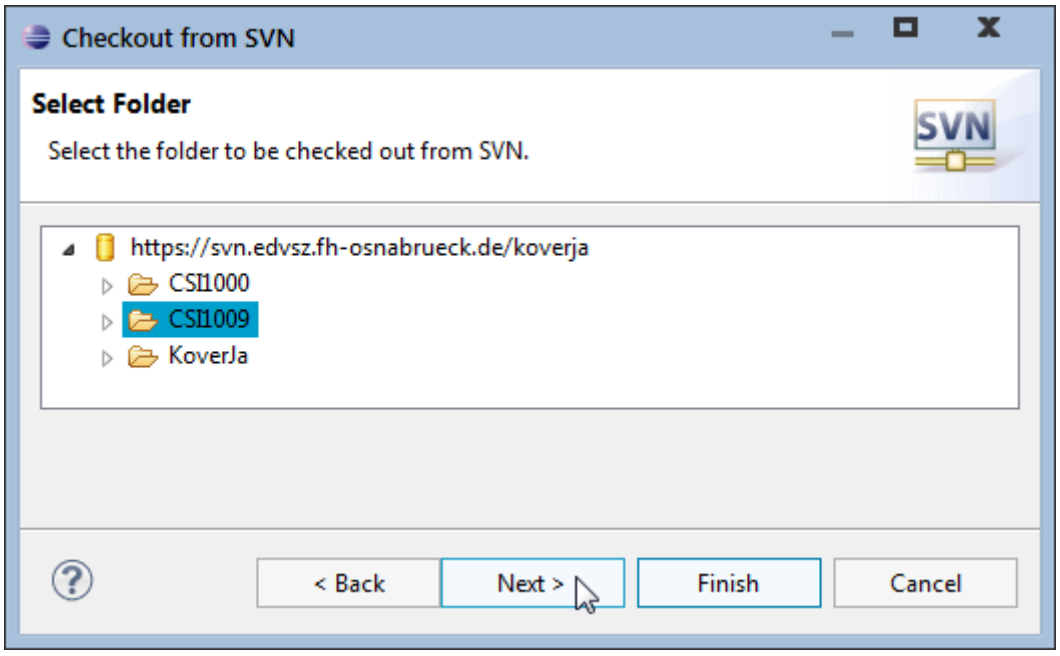
Der Verbindungsaufbau kann etwas dauern.



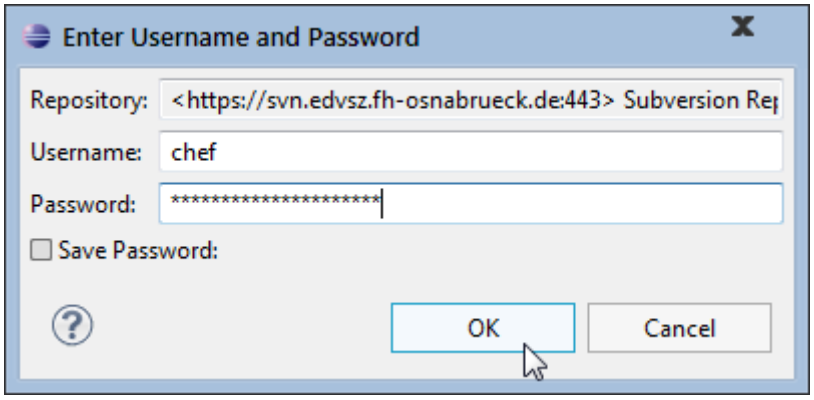
Danach muss die Nutzerkennung eingegeben werden und man muss entscheiden, ob diese Daten gespeichert werden sollen.



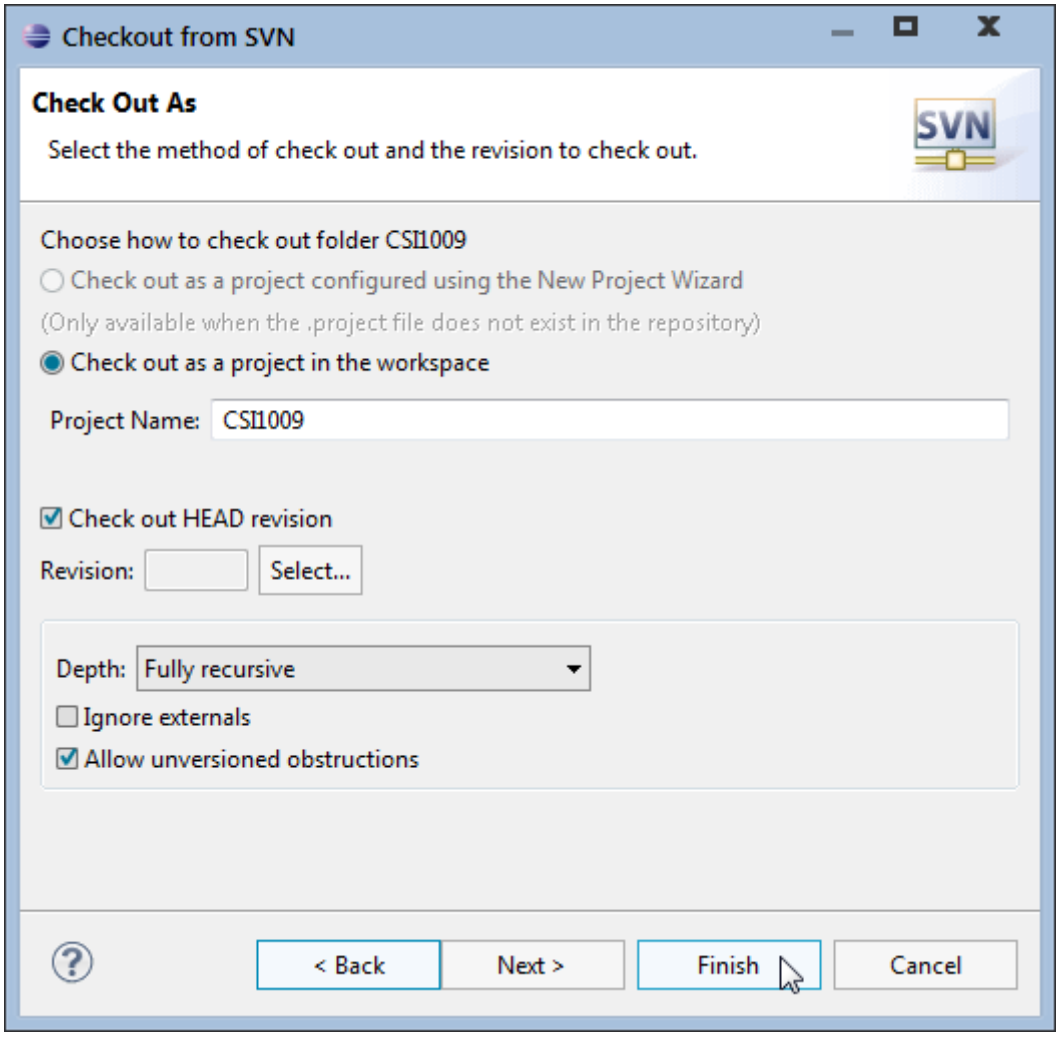
Danach kann der zu nutzende SVN-Ordner ausgewählt werden, der typischerweise einem Projekt entspricht, danach wird „Next>“ gedrückt.



Eventuell muss man sich erneut anmelden.

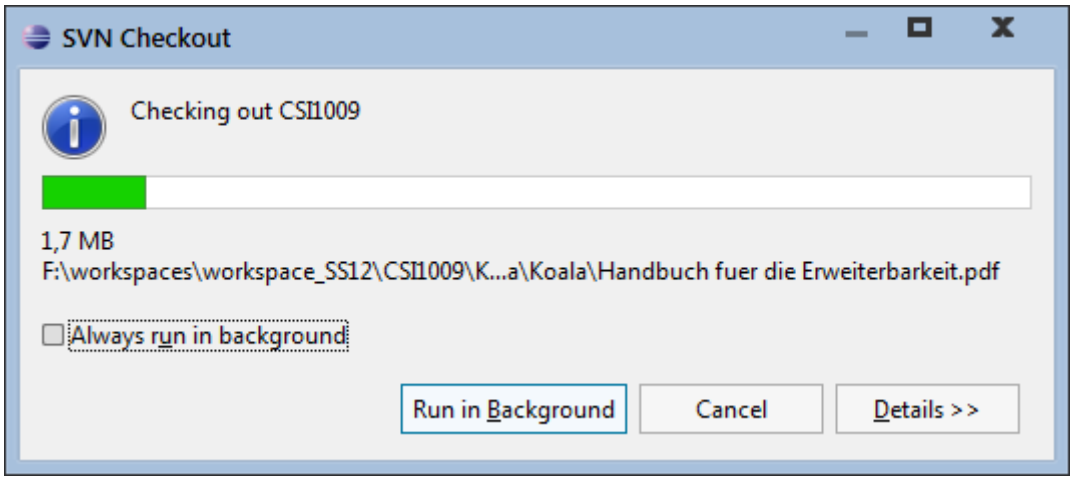


Danach kann genau angegeben werden, was ausgecheckt werden und wie der zugehörige Projektname lauten soll. Ändert man die Einstellungen nicht, wird der Ordnername als Projektname genutzt und die aktuellste Version ausgecheckt. Ist das gewünscht, wird „Finish“ gedrückt.

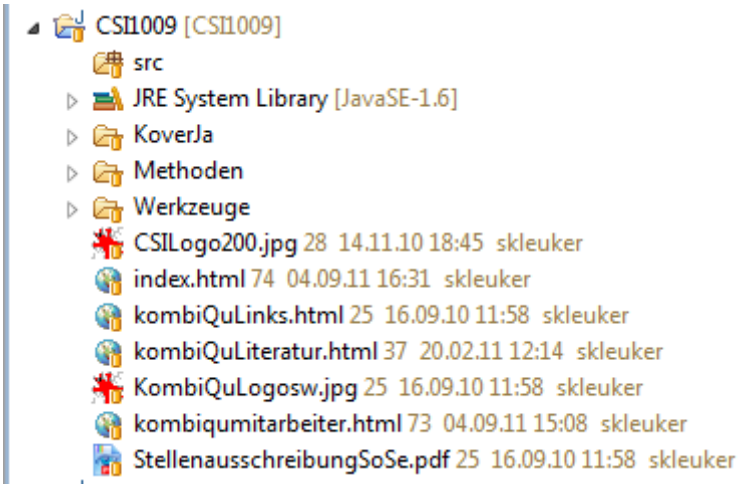


Gegebenenfalls muss man sich nochmals anmelden.

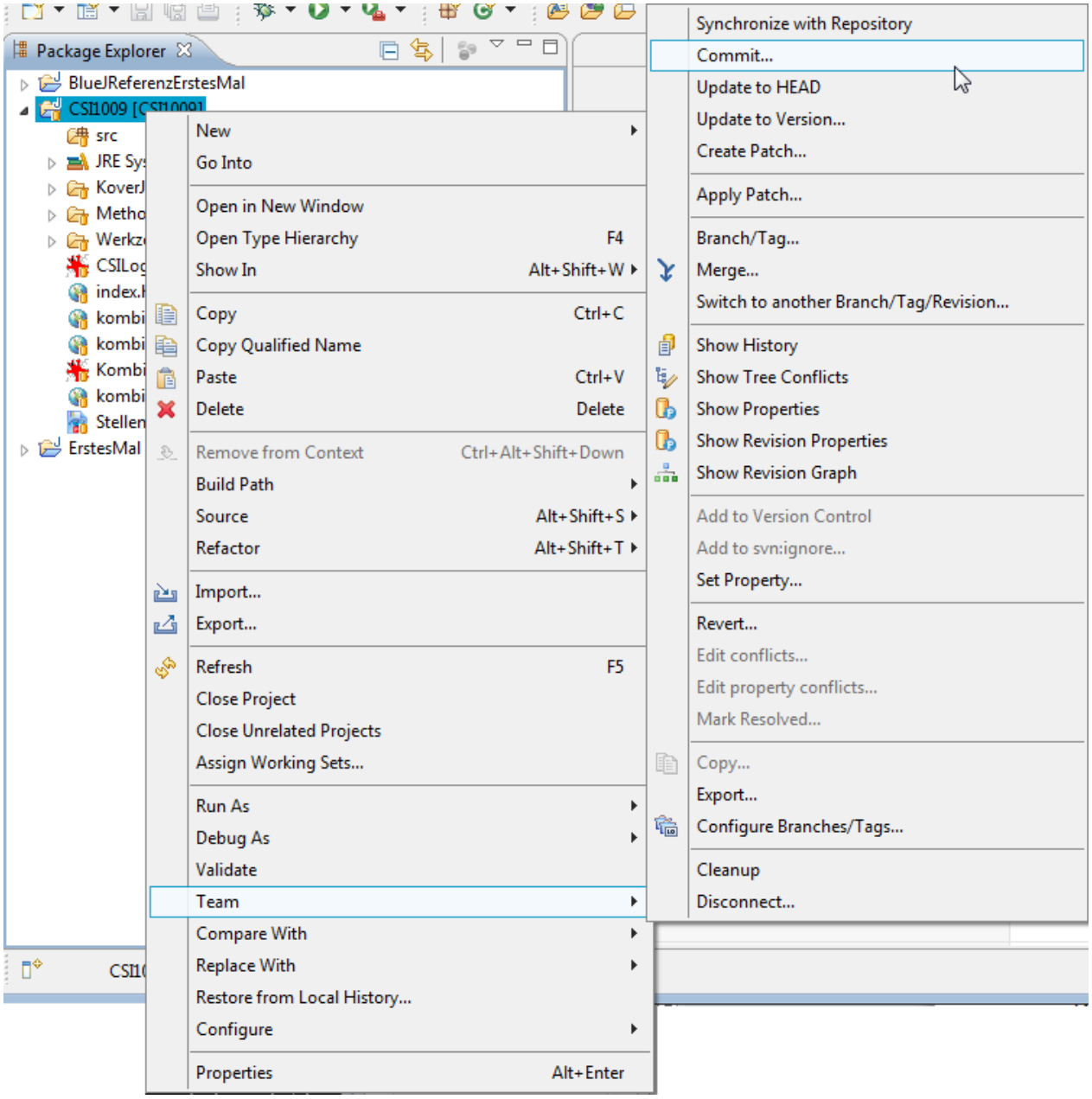
Das Auschecken kann einige Zeit dauern.



Danach liegt das Projekt in Eclipse vor, man erkennt die kleinen Tonnensymbole und kann gearbeitet werden.



Die SVN-Nutzung befindet sich wiedermit einem Rechtsklick auf dem Projekt unter „Team“.


Package Explorer
BlueJReferenzErstesMal
CSI1009
src
ErstesMal
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Validate
Team
Compare With
Replace With
Restore from Local History...
Configure
Properties Alt+Enter
Synchronize with Repository
Commit...
Update to HEAD
Update to Version...
Create Patch...
Apply Patch...
Branch/Tag...
Merge...
Switch to another Branch/Tag/Revision...
Show History
Show Tree Conflicts
Show Properties
Show Revision Properties
Show Revision Graph
Add to Version Control
Add to svn:ignore...
Set Property...
Revert...
Edit conflicts...
Edit property conflicts...
Mark Resolved...
Copy...
Export...
Configure Branches/Tags...
Cleanup
Disconnect...

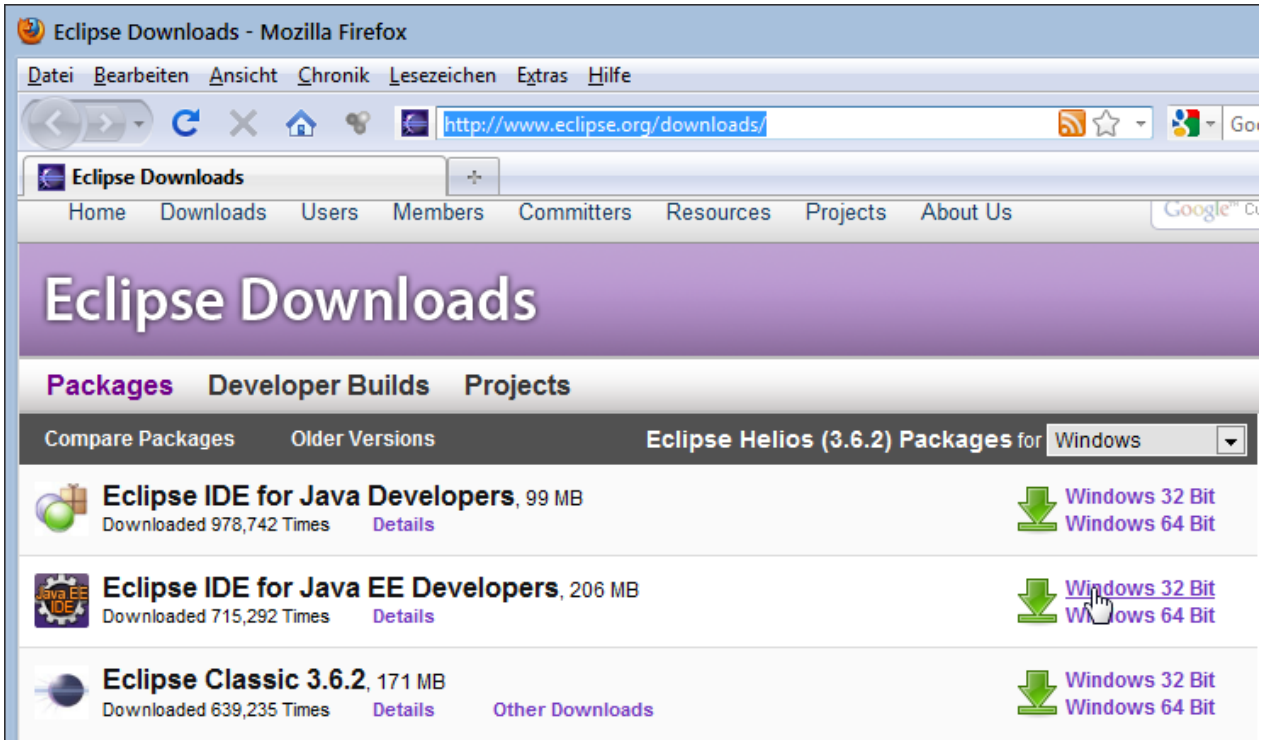
18 Eclipse JEE und GlassFish-Installation (4.3.1)

Zur Entwicklung von Web-Applikationen, aber auch zur JPA-Nutzung, gibt es eine vorkonfigurierte Eclipse-Variante, in der die wesentlichen Erweiterungen von Eclipse zur Entwicklung solcher Applikationen enthalten sind. Weitere Plugins können natürlich nachinstalliert werden.

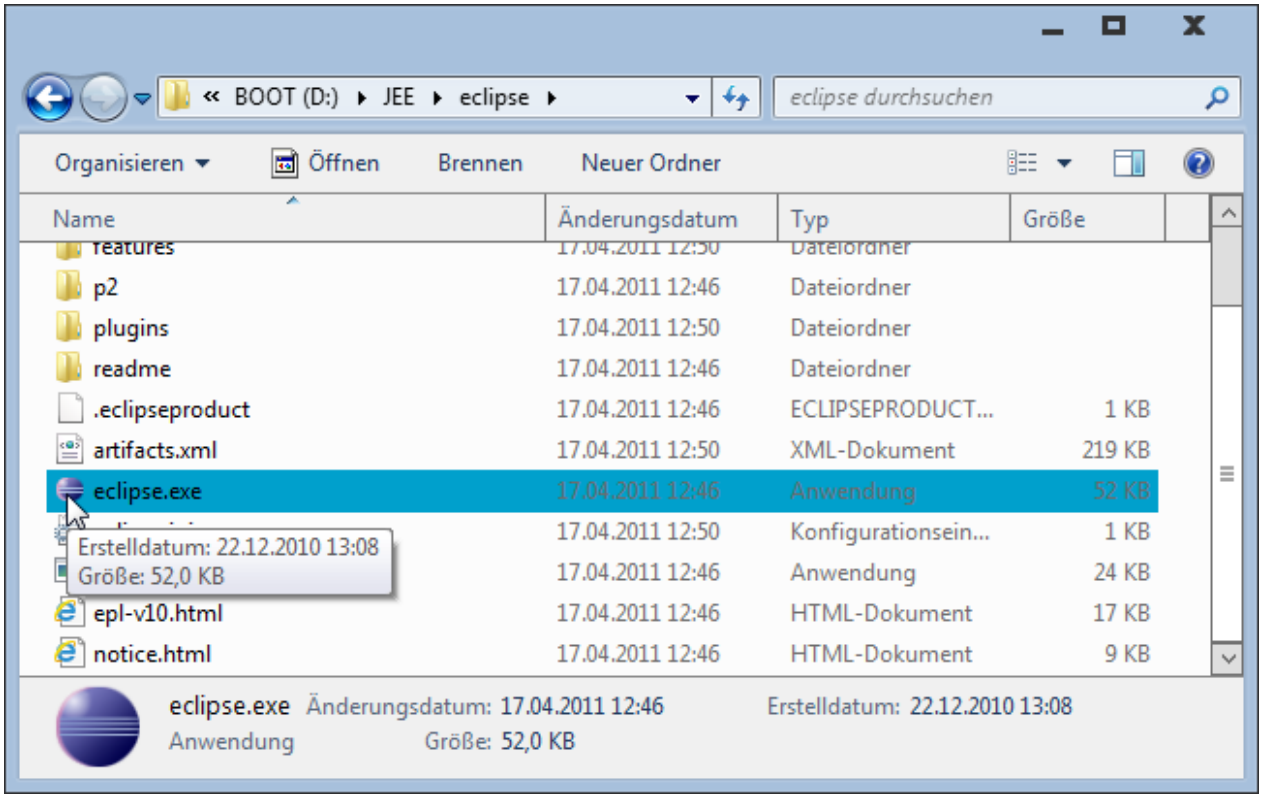
Um Web-Applikationen zu entwickeln, wird weiterhin ein Applikationsserver benötigt, der einfach lokal gestartet und terminiert werden kann. Zu diesem Zweck wird ein GlassFish-Server aufgesetzt.

18.1 Installation von Eclipse JEE

Eclipse JEE benötigt eine halbwegs aktuelle Java-Version (ab Version 6, bzw. 1.6) zur Entwicklung von Software mit gesetzter JAVA_HOME-Variable. Danach wird Eclipse JEE einfach von der Eclipse-Seite geladen und durch Auspacken installiert. Den Download findet man unter http://www.eclipse.org/downloads/. Man muss die zum eigenen Betriebssystem passende Version zu laden.



Für die folgenden Beispiele wird Eclipse im Ordner D:\JEE\ im Unterordner Eclipse ausgepackt. Diese Eclipse-version wird genau wie die anderen mit einem Doppelklick auf eclipse.exe gestartet.

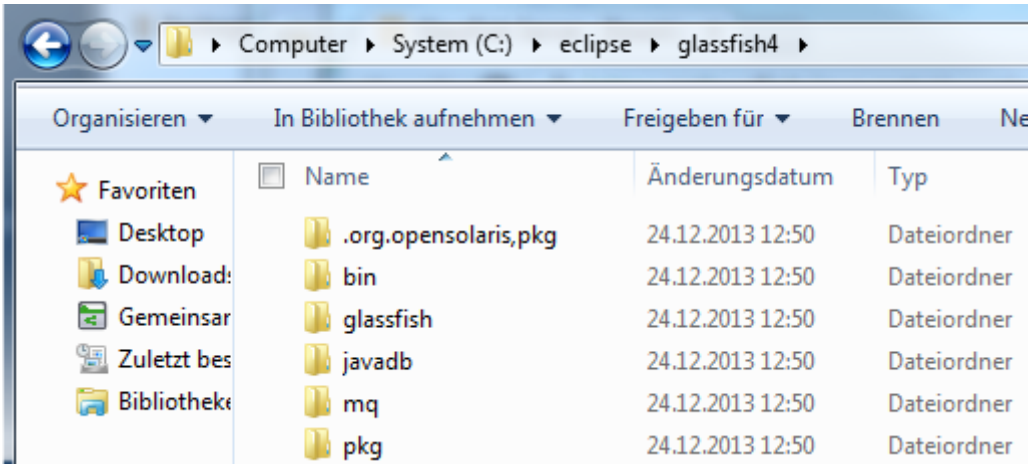


18.2 Installation von GlassFish (4.0)

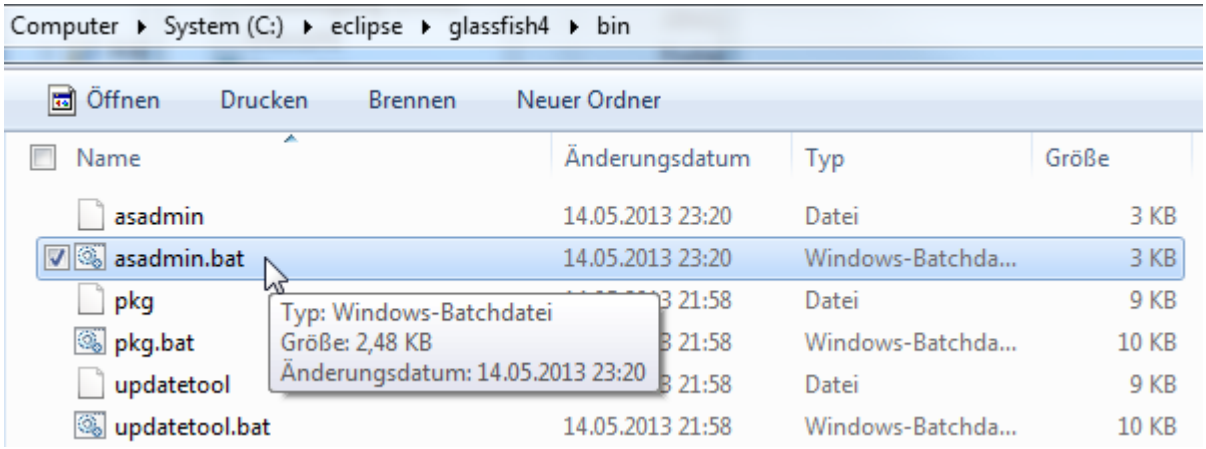
GlassFish kann direkt von Oracle geladen werden, wobei es auch eine Installationsanleitung für die jeweiligen Betriebssysteme gibt. Die Adresse für die Open Source-Edition lautet http://glassfish.java.net/public/downloadsindex.html.

In diesem Beispiel wird die multilinguale Version mit einem Windows-Installer genutzt und heruntergeladen.

Die Zip-Datei wird in einem passenden Ordner ausgepackt. Im konkreten Fall wird der untypische Ordner C:\eclipse genutzt. Beim Entpacken ist zu beachten, dass sich im Zip-Verzeichnis bereits ein Unterordner glassfish4 befindet.

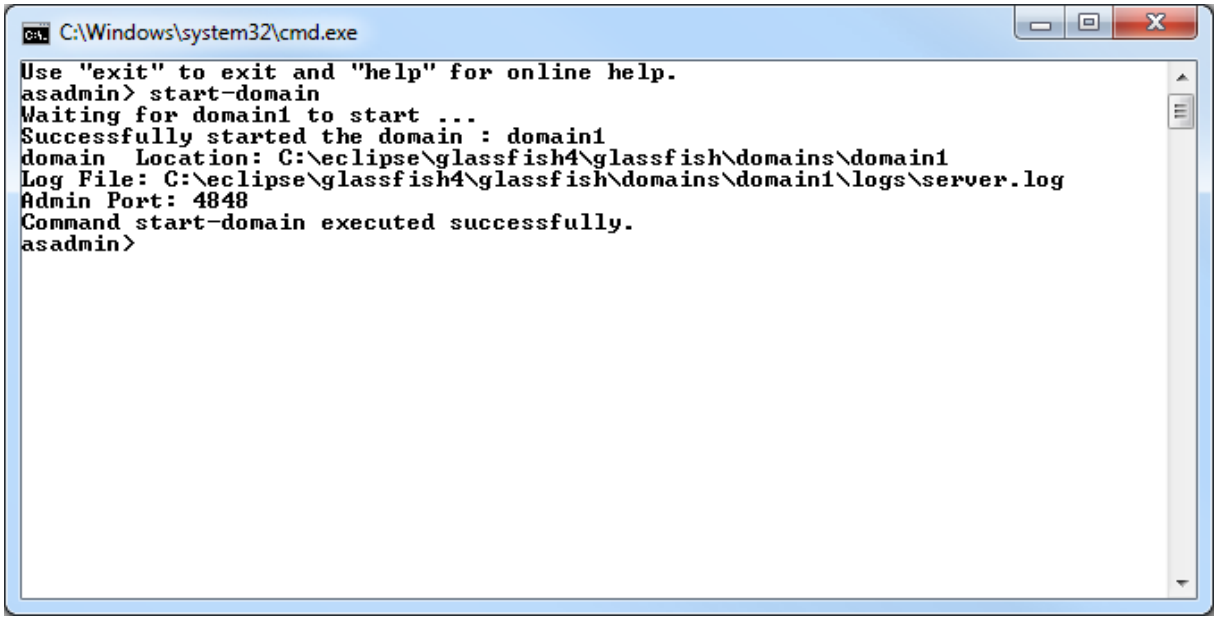


Zur Prüfung, ob der GlassFish überhaupt funktioniert, wird er jetzt gestartet. Dazu wird im Unterverzeichnis bin dann asadmin aufgerufen (oder in einer Konsole asadmin start-domain)

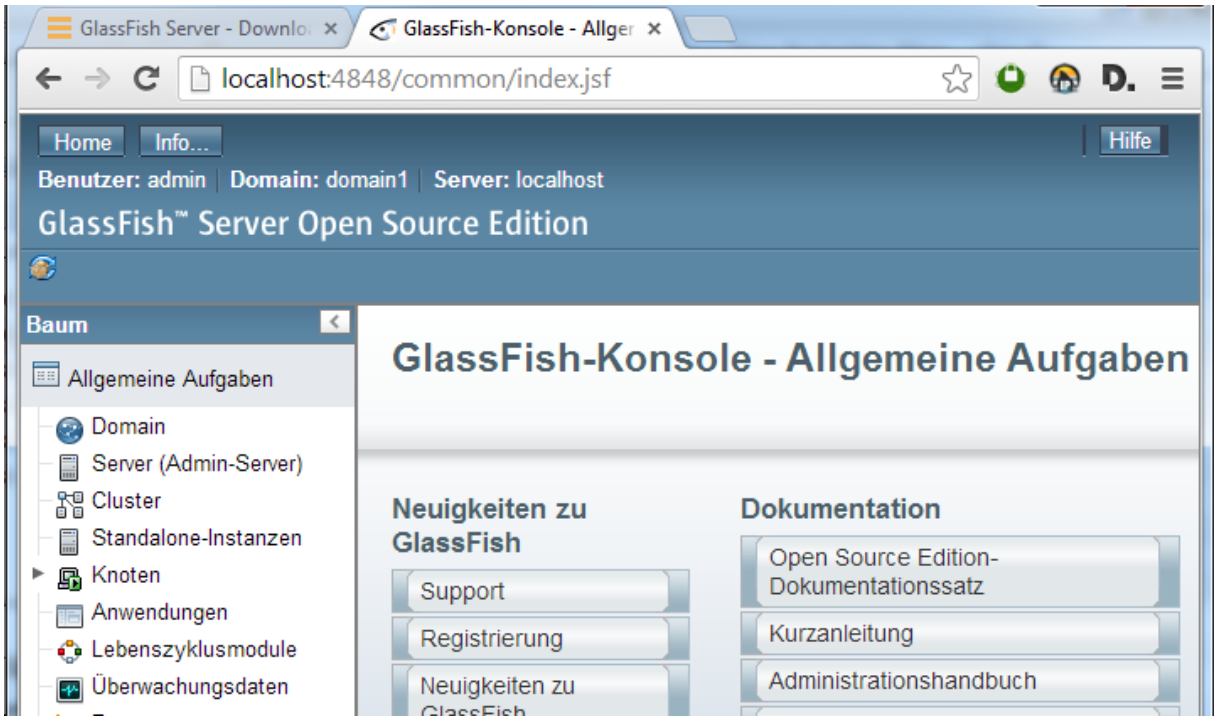


Abhängig von der Firewall und anderen Sicherheitseinstellungen, müssen einige Berechtigungen erteilt werden.

Es öffner sich ein Konsolenfenster, das zentral zur Steuerung des Servers genutzt wird. Im konkreten Fall wird nur der Befehl start-domain eingegeben. Das Konsolenfenster darf nicht geschlossen werden, da sonst auch der Server beendet wird. Wenn man dies möchte, sollte vorher der Befehl stop-domain eingegeben werden.



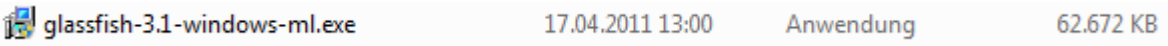
In einem Browser kann dann unter der Adresse http://localhost:4848 die Administration des Servers über das Web-Interface bearbeitet werden.



Das nachfolgende Unterkapitel kann übersprungen werden.

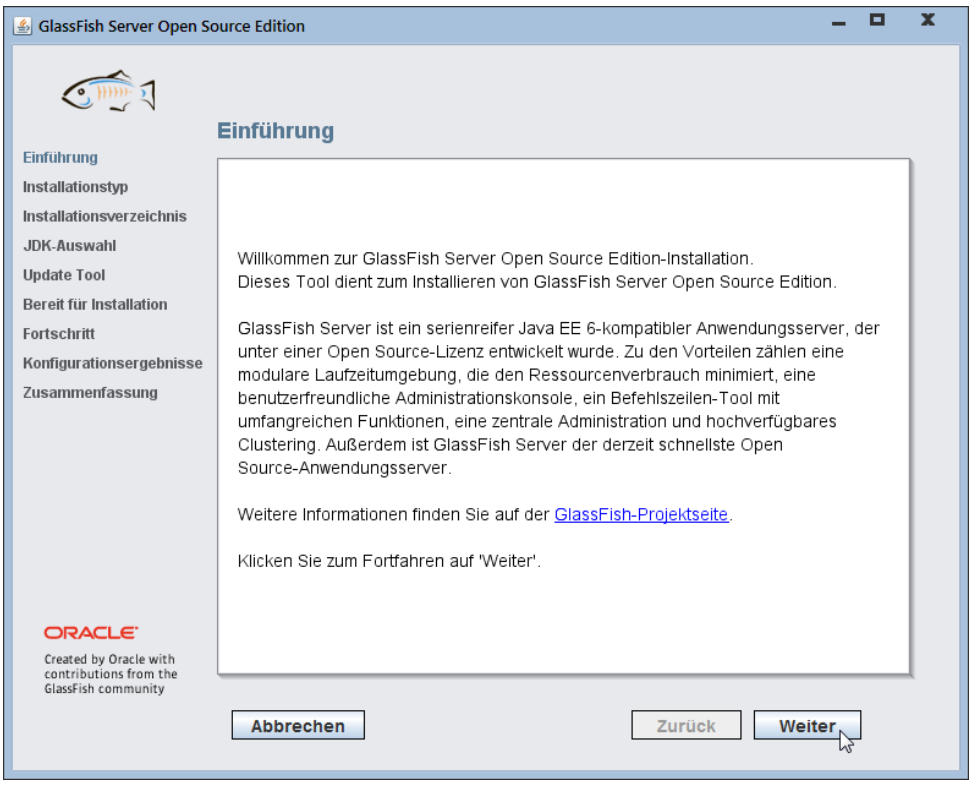
18.3 Installation von GlassFish mit dem Native Installer

Lädt man sich statt einer Zip-Datei den Native Installer herunter, können bei der Installation einige Einstellungen vorgemmen werden, die hier für die GlassFish-Cerion 3.1 gezeigt werden. Die Installation wird durch einen Doppelklick gestartet.

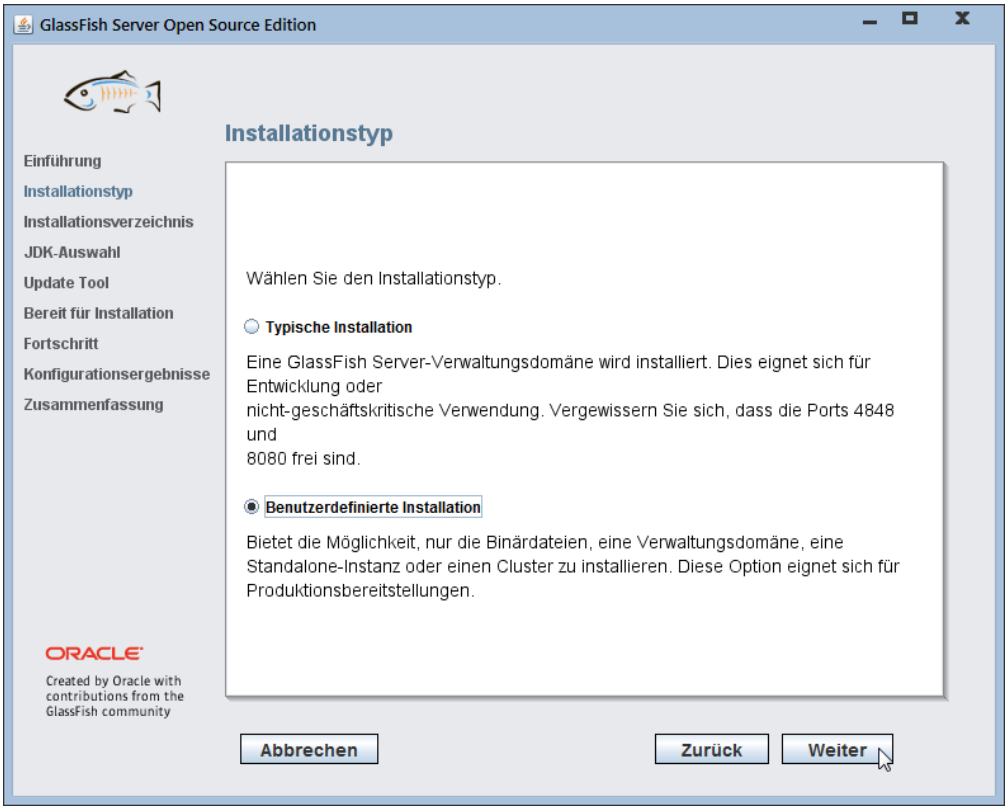


Bei der Installation ist generell zu beachten, dass ein Web-Server installiert wird und abhängig von der weiteren Konfiguration des Rechners und des genutzten Netzwerks dadurch ein Zugriff von außen auf den eigenen Rechner über die installierte Software möglich ist. Hier sollte eine sinnvoll konfigurierte Firewall allerdings die Probleme lösen.

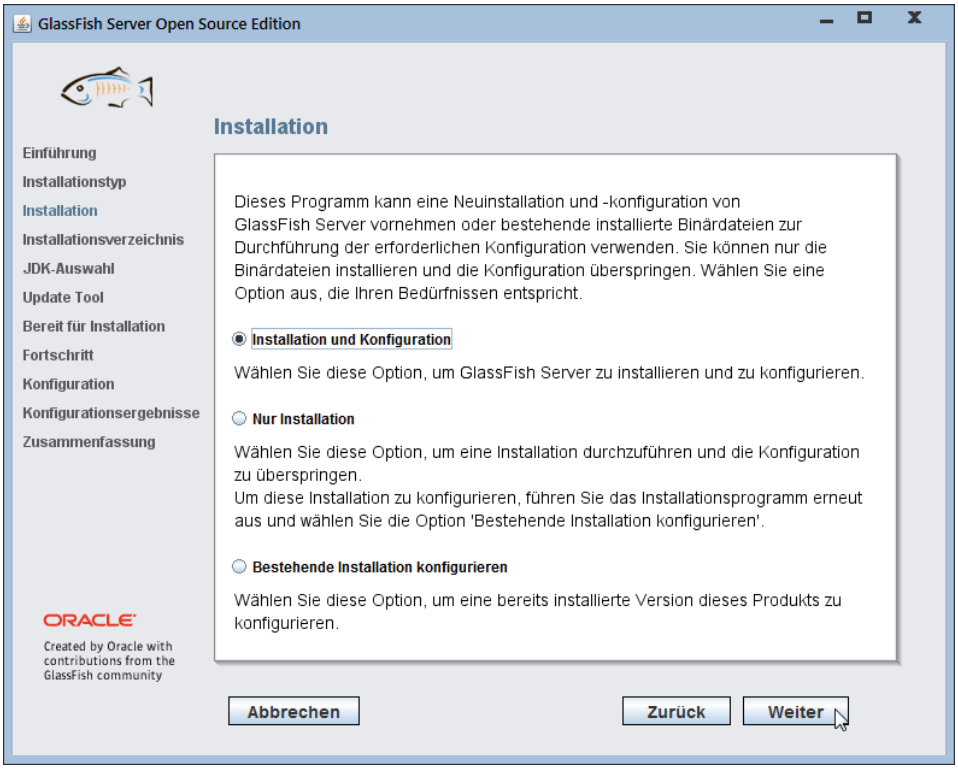
Das erste Installationsfenster wird gelesen und „Weiter“ gedrückt.



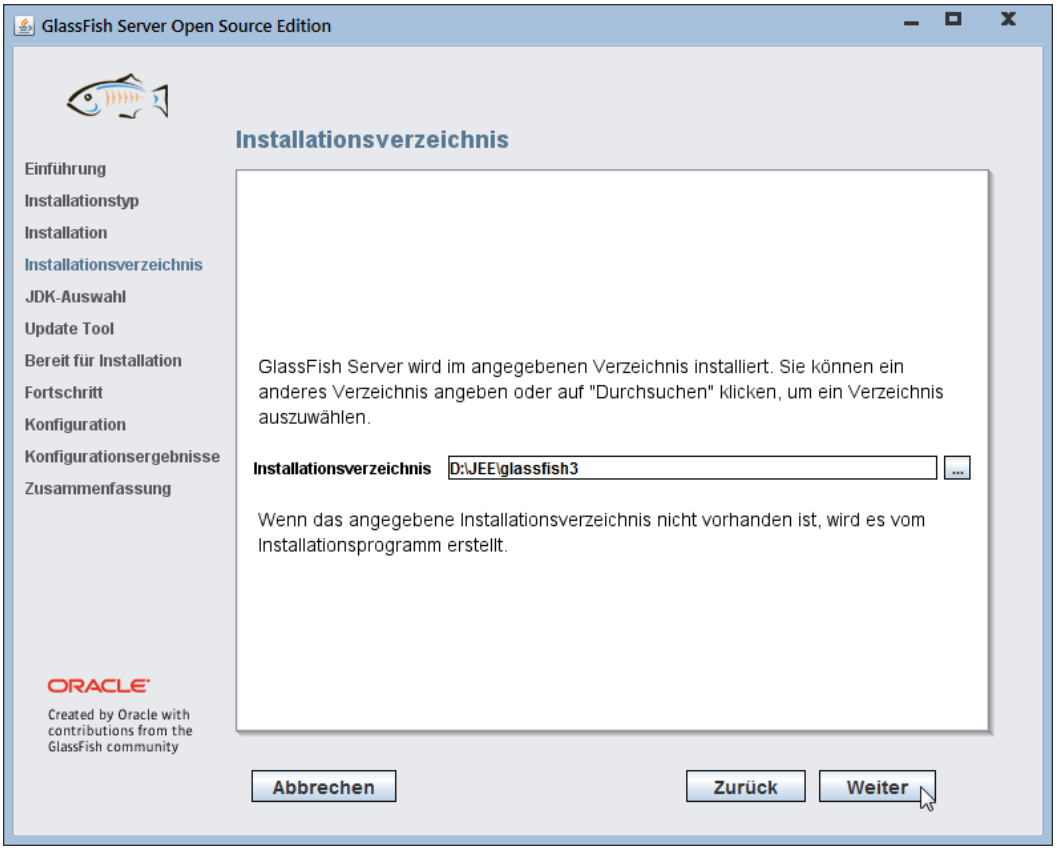
Im nächsten Schritt kann eine typische Installation gewählt werden, da diese Einstellungen für Entwicklungsaufgaben ausreichen. Man beachte die Forderung nach den freien Ports 4848 (Admin-Konsole) und 8080 (Verbindungsaufbau). Hier wird die benutzerdefinierte Installation verfolgt.



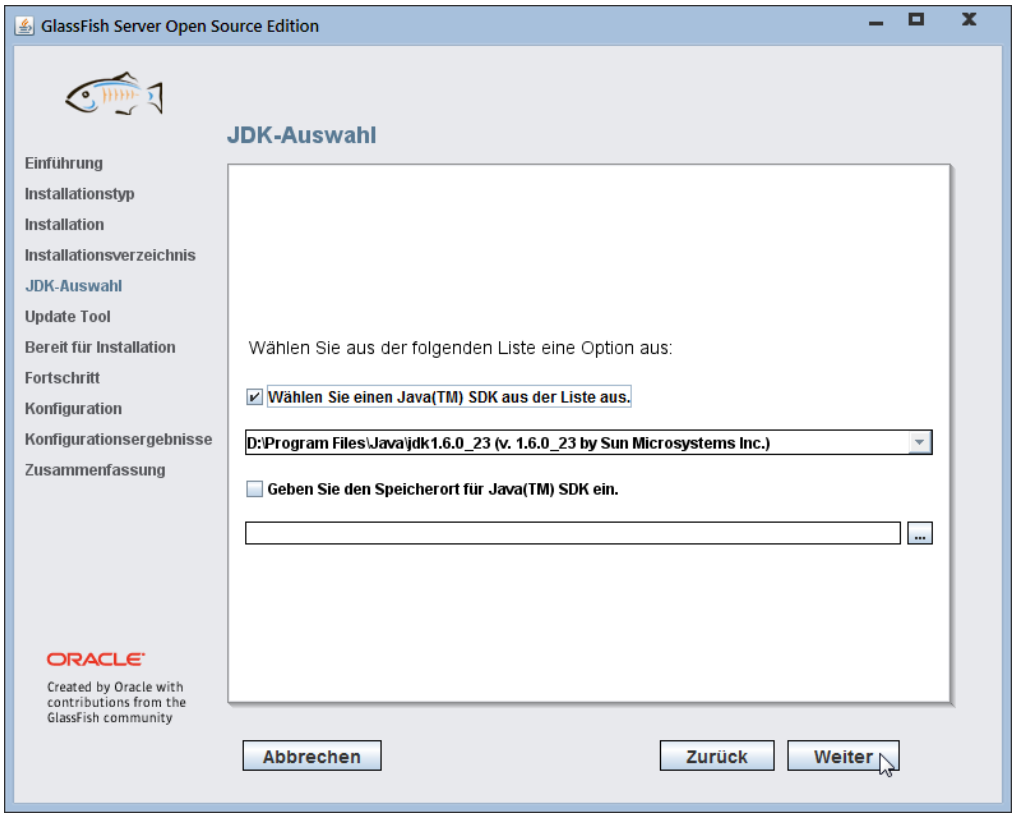
Es wird „Installation und Konfiguration“ gewählt.



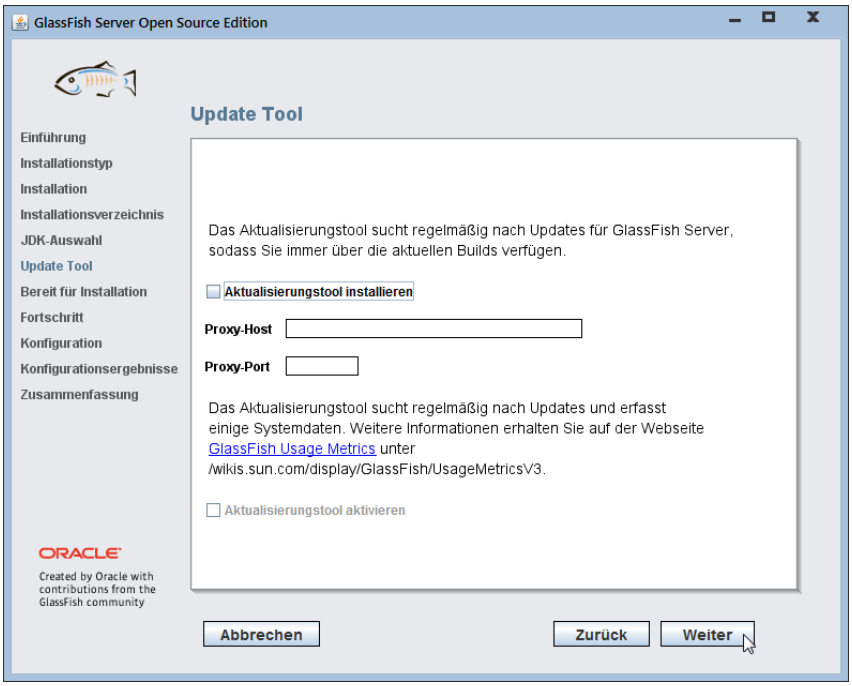
Das Installationsverzeichnis kann individuell, hier D:\JEE\glassfish3, gewählt werden.



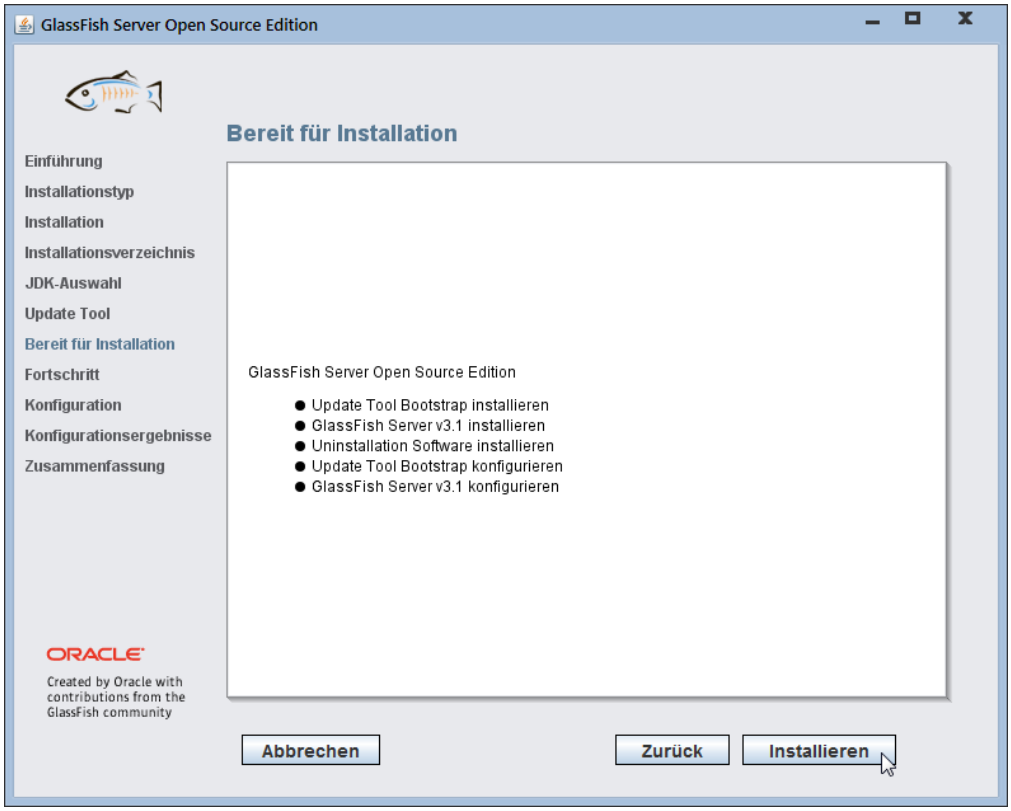
Nun sollte das installierte JDK gefunden werden, bei mehreren JDK-Installationen kann man eine auswählen. Wird keine Installation gefunden, muss ein zu einem JDK gehörender Pfad in der unteren Auswahl eingetragen werden.



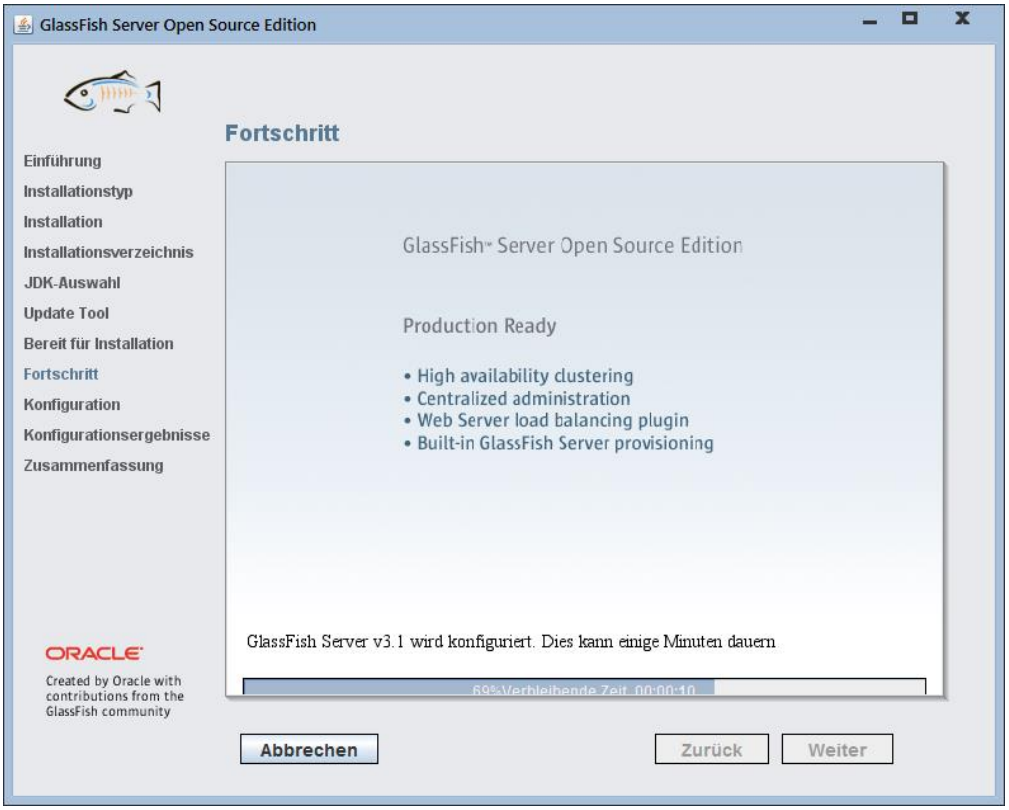
Es wird ein Aktualisierungstool angeboten, dass man auch wie im Beispiel ablehnen kann, damit Änderungen nachvollziehbar bleiben.



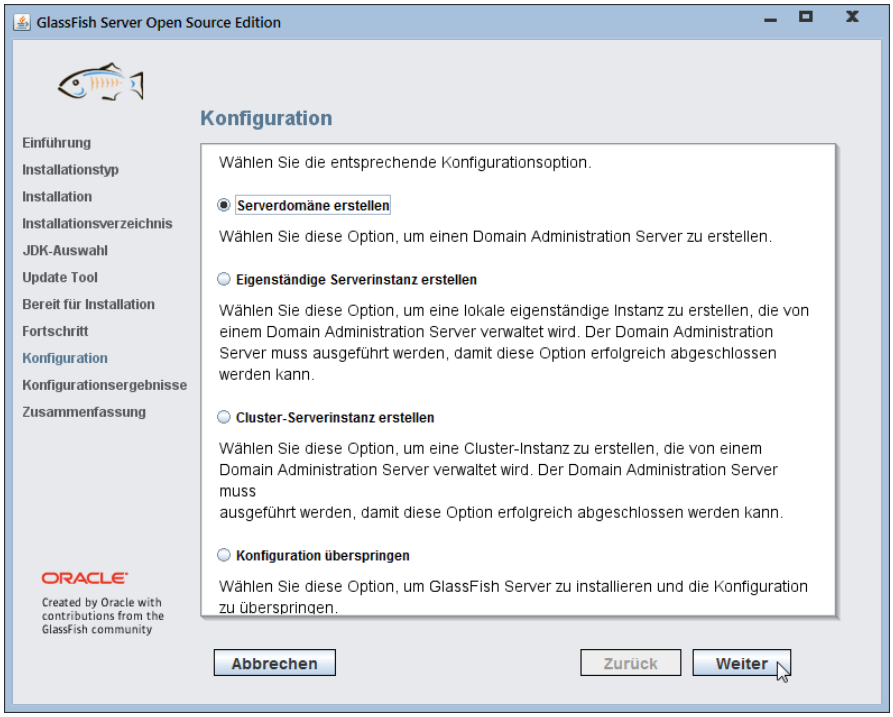
Nun kann die Software installiert werden.



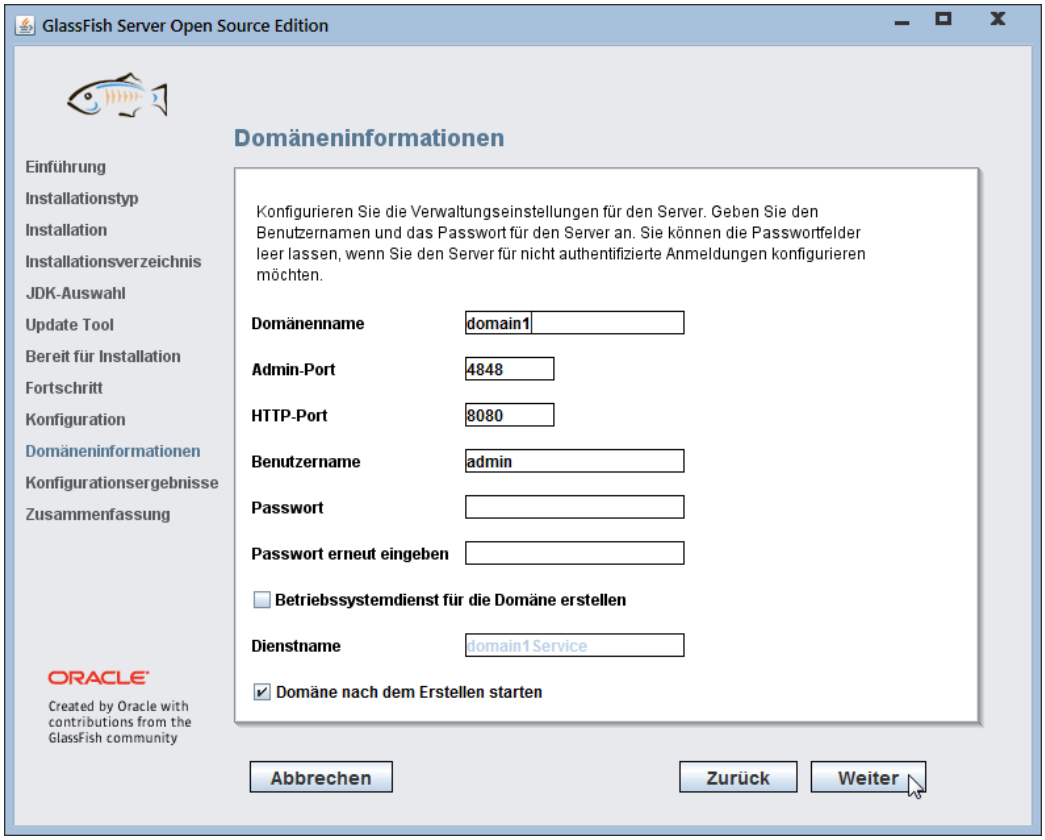
Die Installation kann einige Zeit dauern.



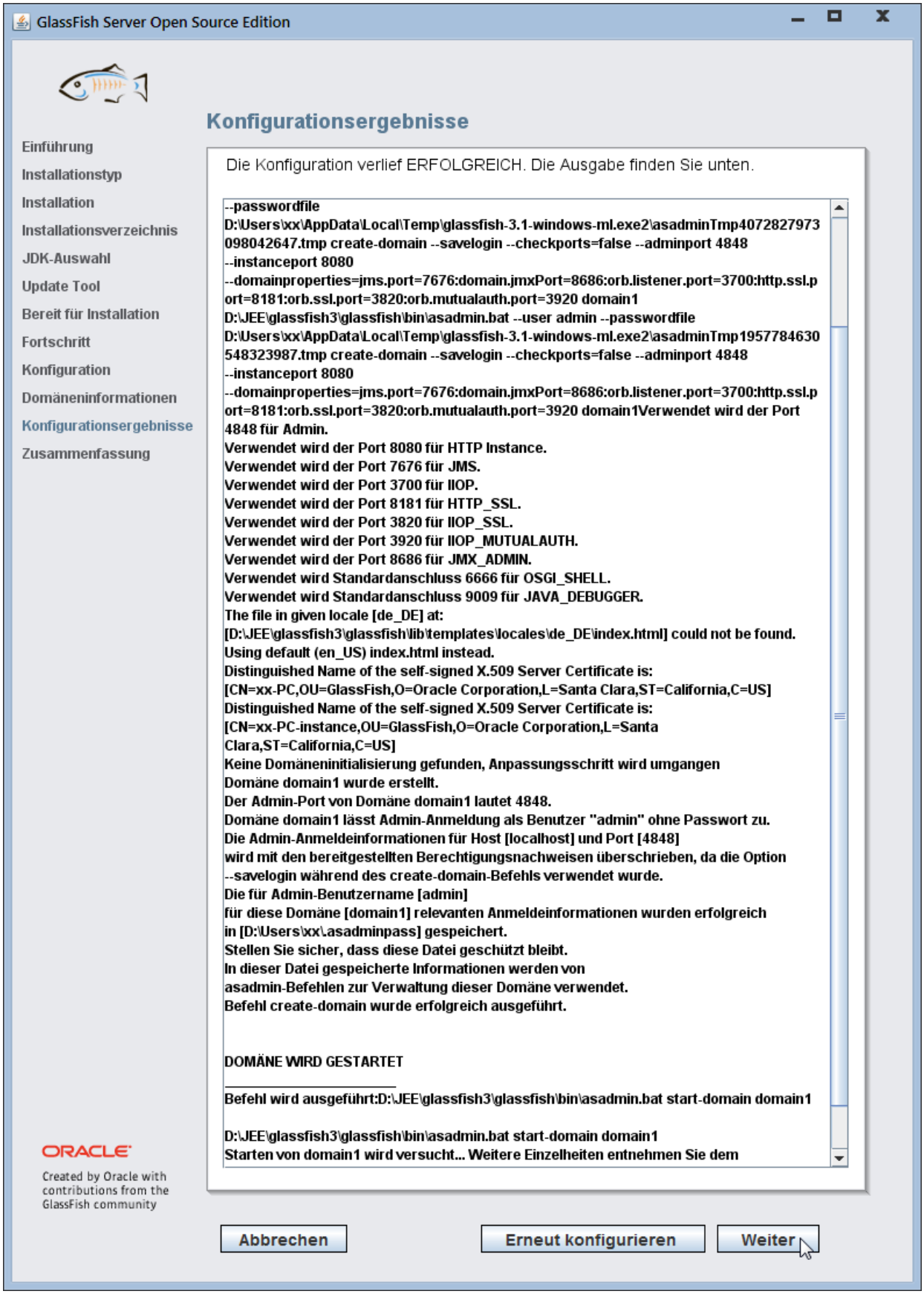
Im nächsten Fenster wird „Serverdomäne erstellen“ gewählt.



Die vorgegeben Einstellungen können übernommen werden, der Admin-Port ist damit 4848, man beachte, dass der Admin kein Passwort hat, was für die Entwicklung sinnvoll ist, aber im laufenden Betrieb natürlich geändert werden muss.

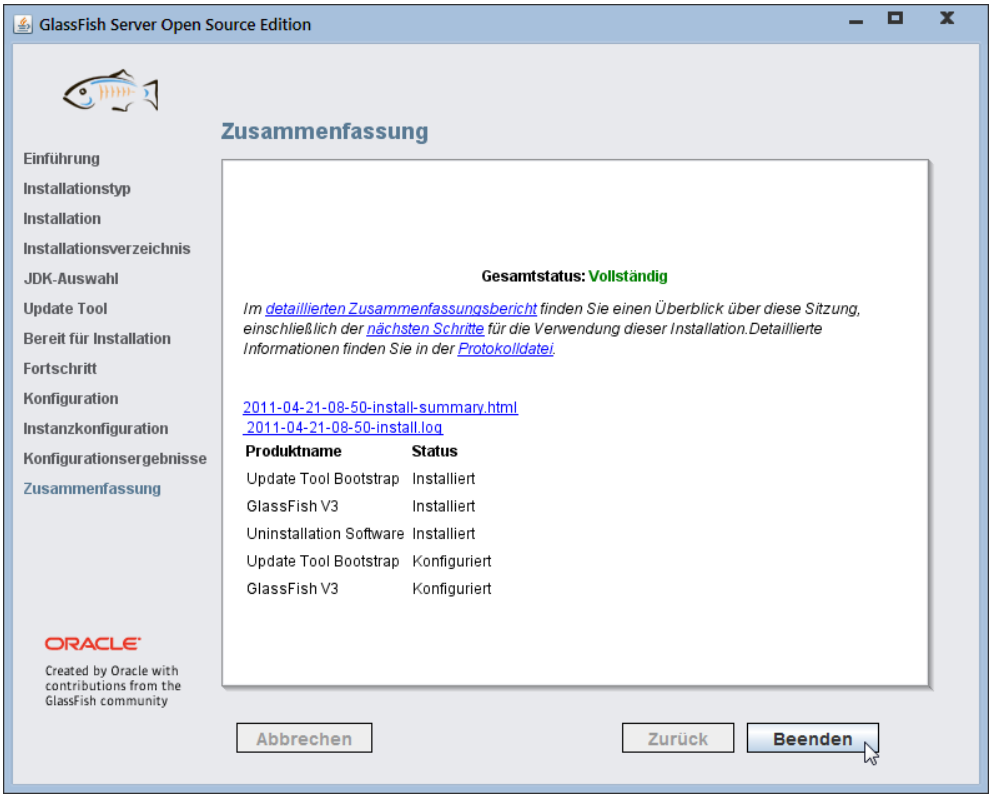


Nun wird der Server gestartet, was einige Zeit dauern kann.

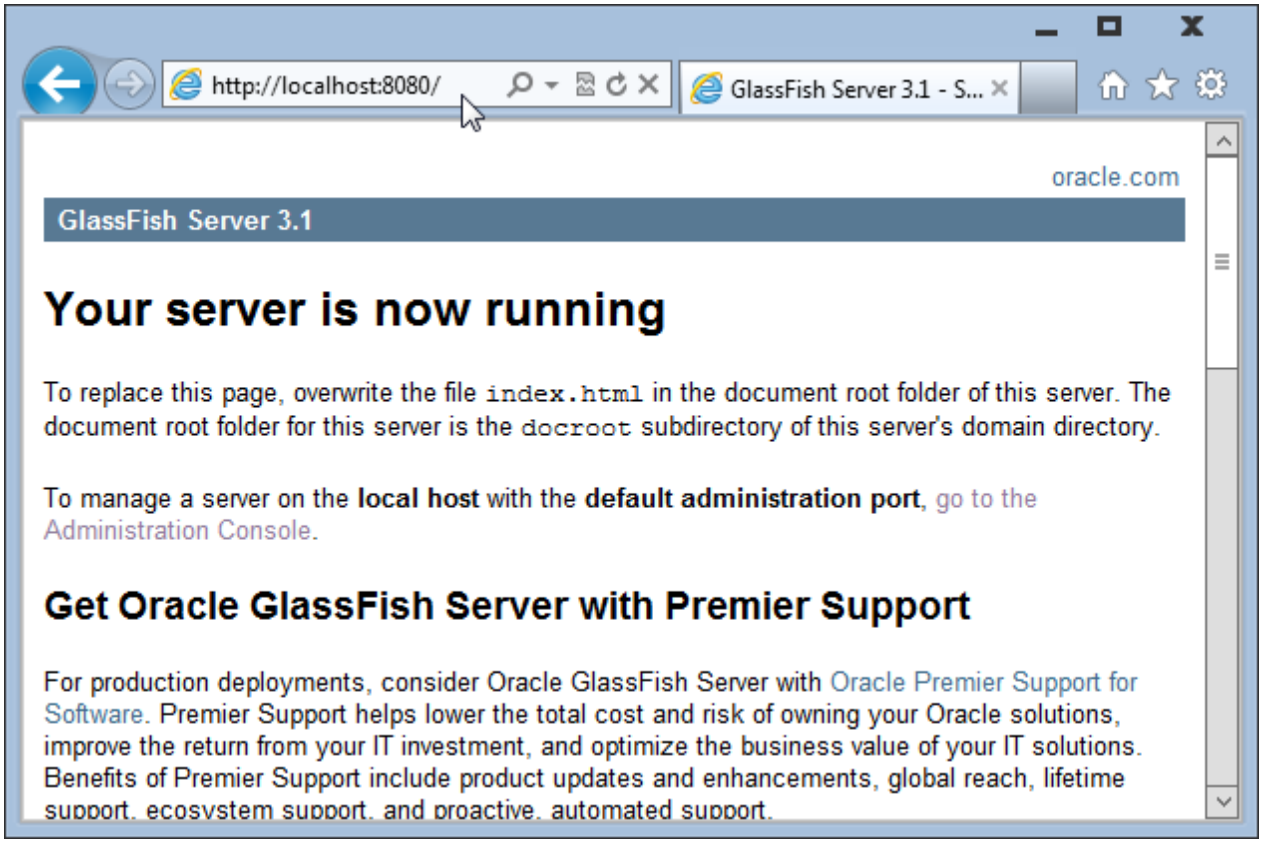


Beim Start kann sich eine installierte Firewall melden, mit der der Zugriff erlaubt wird.

Danach ist die Installation abgeschlossen.

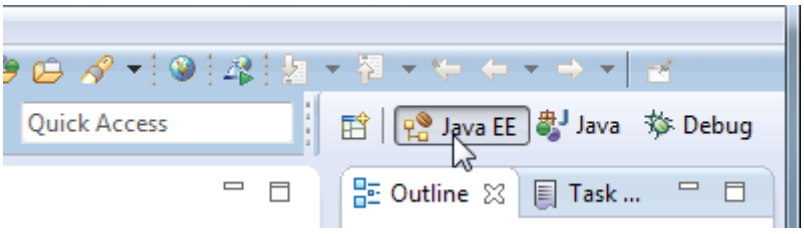


Da der Server bereits gestartet wurde, führt ein Aufruf der lokalen Web-Seite http://localhost:8080/ zur Startnachricht des Servers.

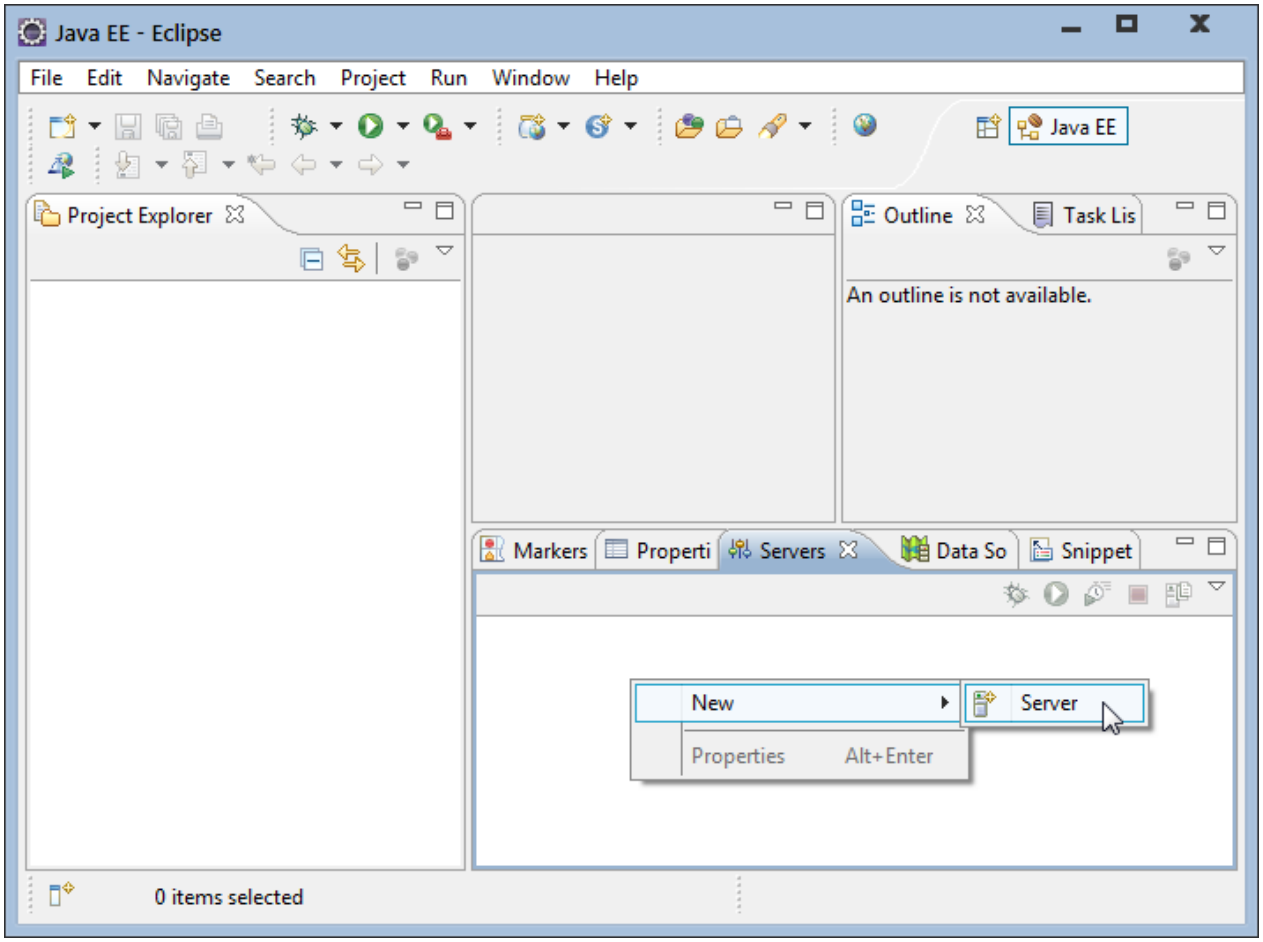


18.4 Integration von GlassFish in Eclipse JEE

Nachdem Eclipse JEE und GlassFish installiert sind, soll im nächsten Schritt die Möglichkeit geschaffen werden, GlassFish von Eclipse aus zu steuern. Durch die Integration des Servers ist es weiterhin möglich, dass entwickelte Software automatisch auf dem Server installiert (deployed) wird, was viele recht monotone Schritte, wie das Packen und Installieren, automatisiert. Zunächst wird Eclipse gestartet. Weiterhin muss in die „JavaEE-Sicht gewechselt werden, falls dies nicht aktiv ist.

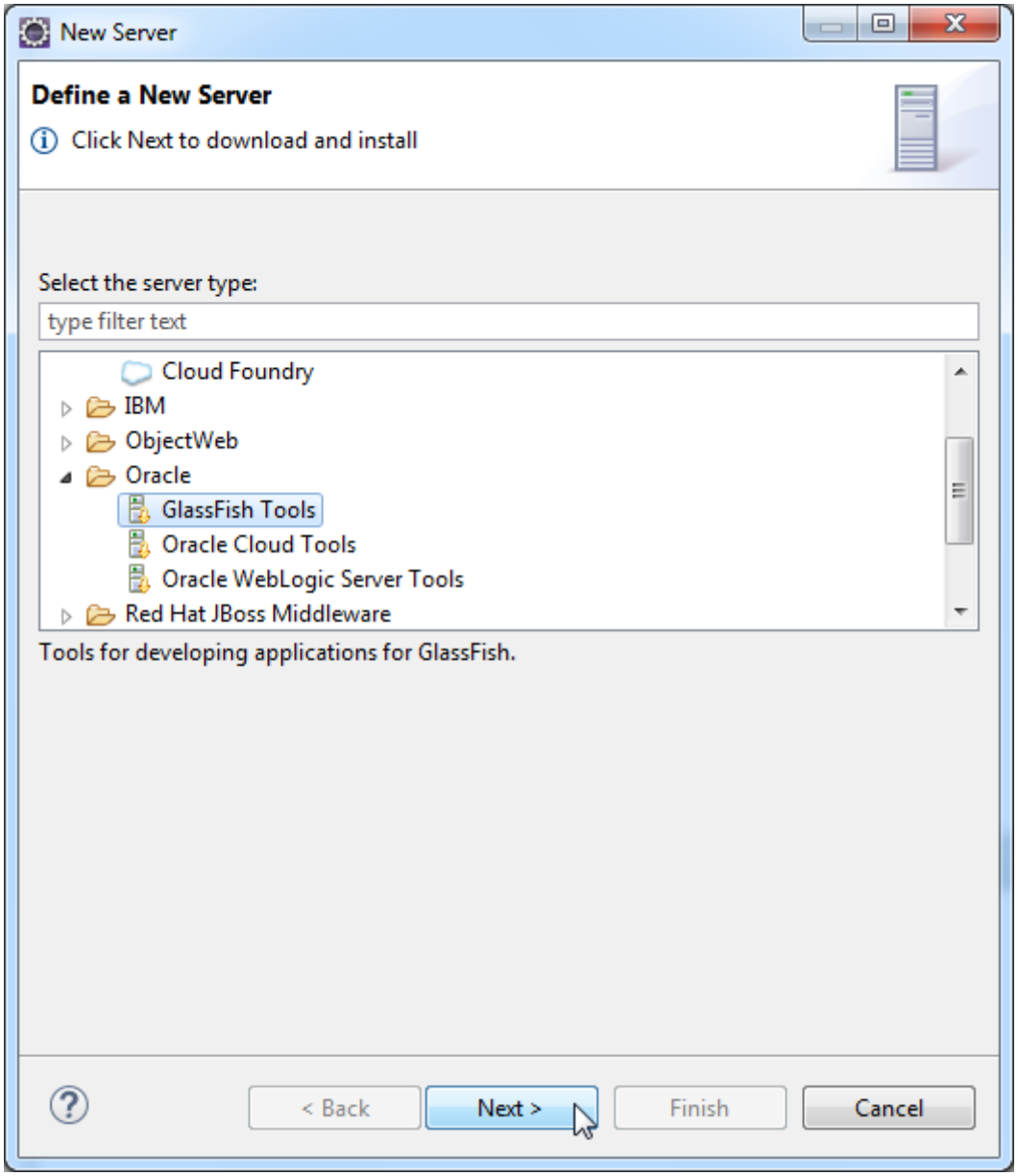


Rechts unten befindet sich ein Reiter „Servers“, bei dem ein Rechtsklick im leeren weißen Feld ausgeführt und „New“ mit „Server“ ausgewählt wird.

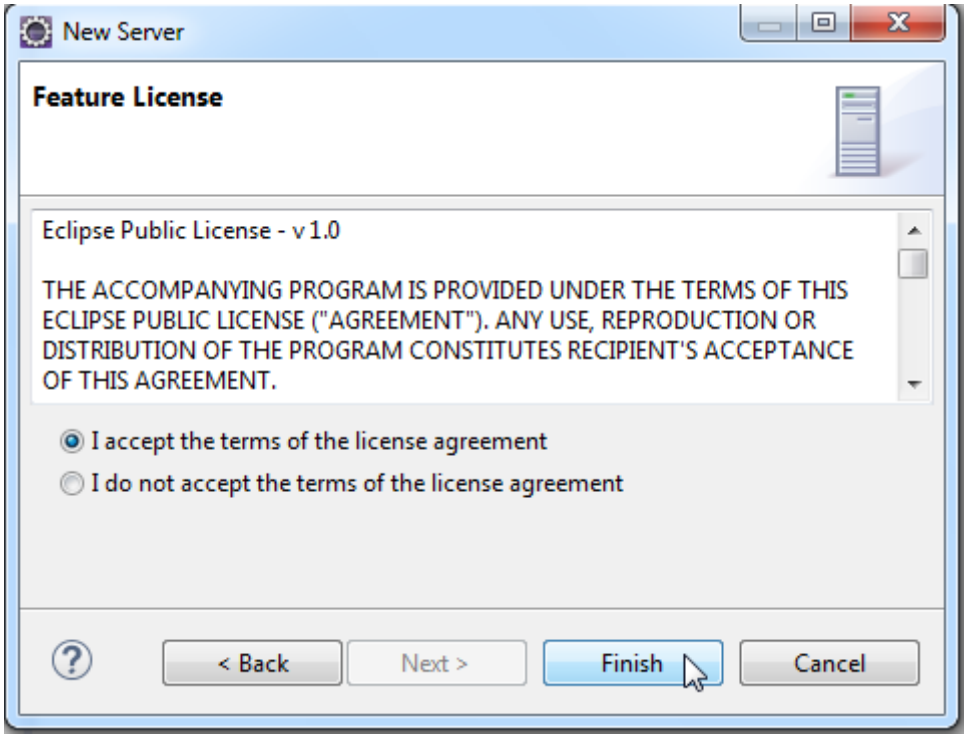


Sollte der Reiter „Oracle“ nicht vorhanden sein, kann die Anleitung bis zur erfolgreichen direkten Installation übersprungen werden, der alternative Weg wird dann gezeigt.

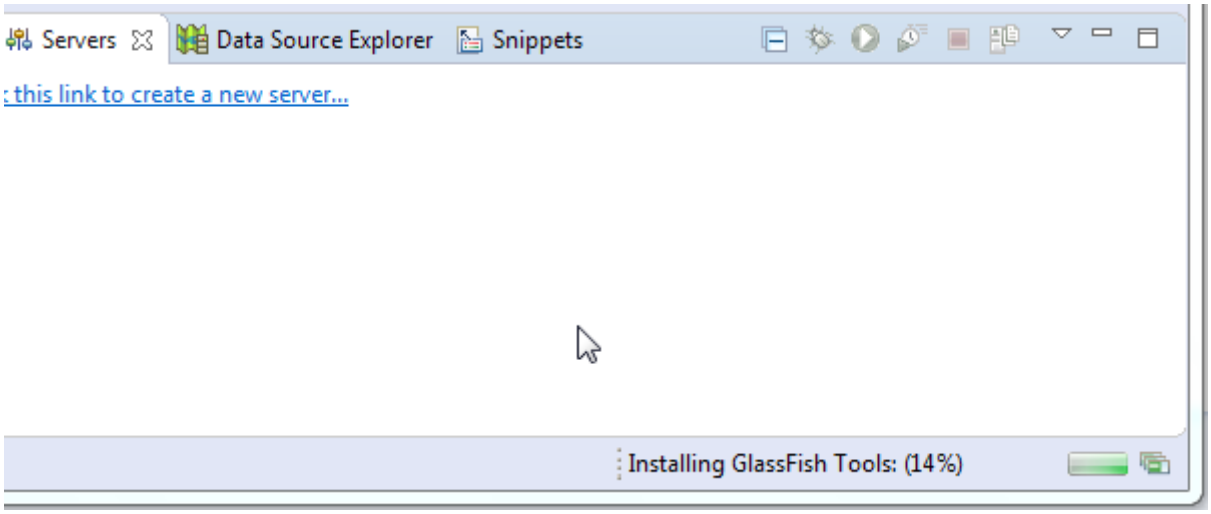
Es wird der Tab „Oracle“ aufgeklappt, „GlassFish Tools“ ausgewählt und unten „Next>“ geklickt. Es finden einige Downloads statt.



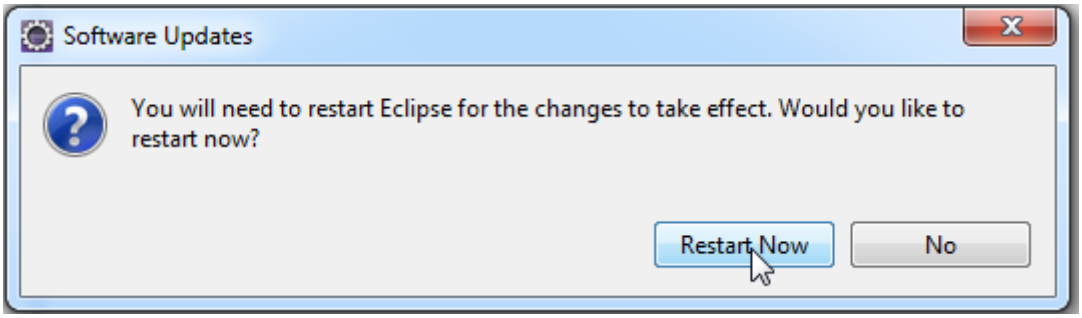
Die Lizenz muss gelesen und akzeptiert und „Finish“ geklickt werden.



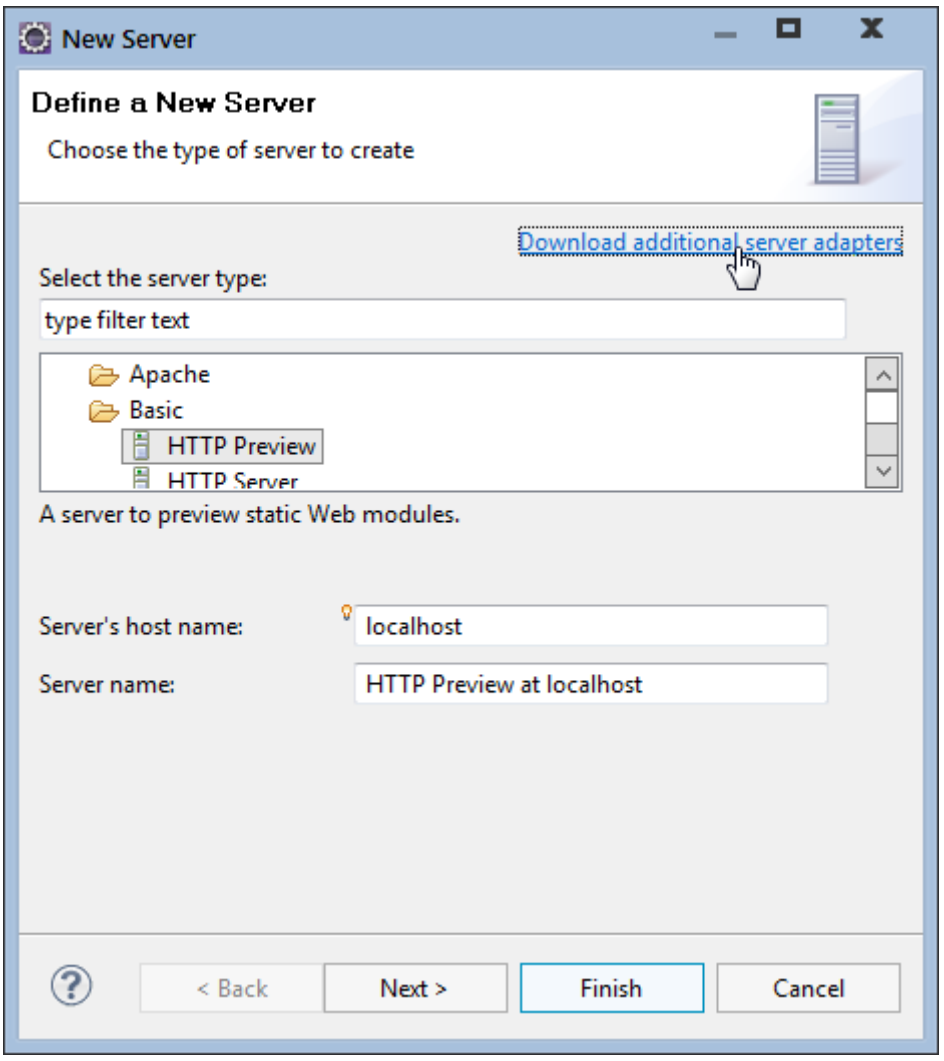
Es beginnt eine durch länger dauern Installation, die rechts-unten verfolgt werden kann.



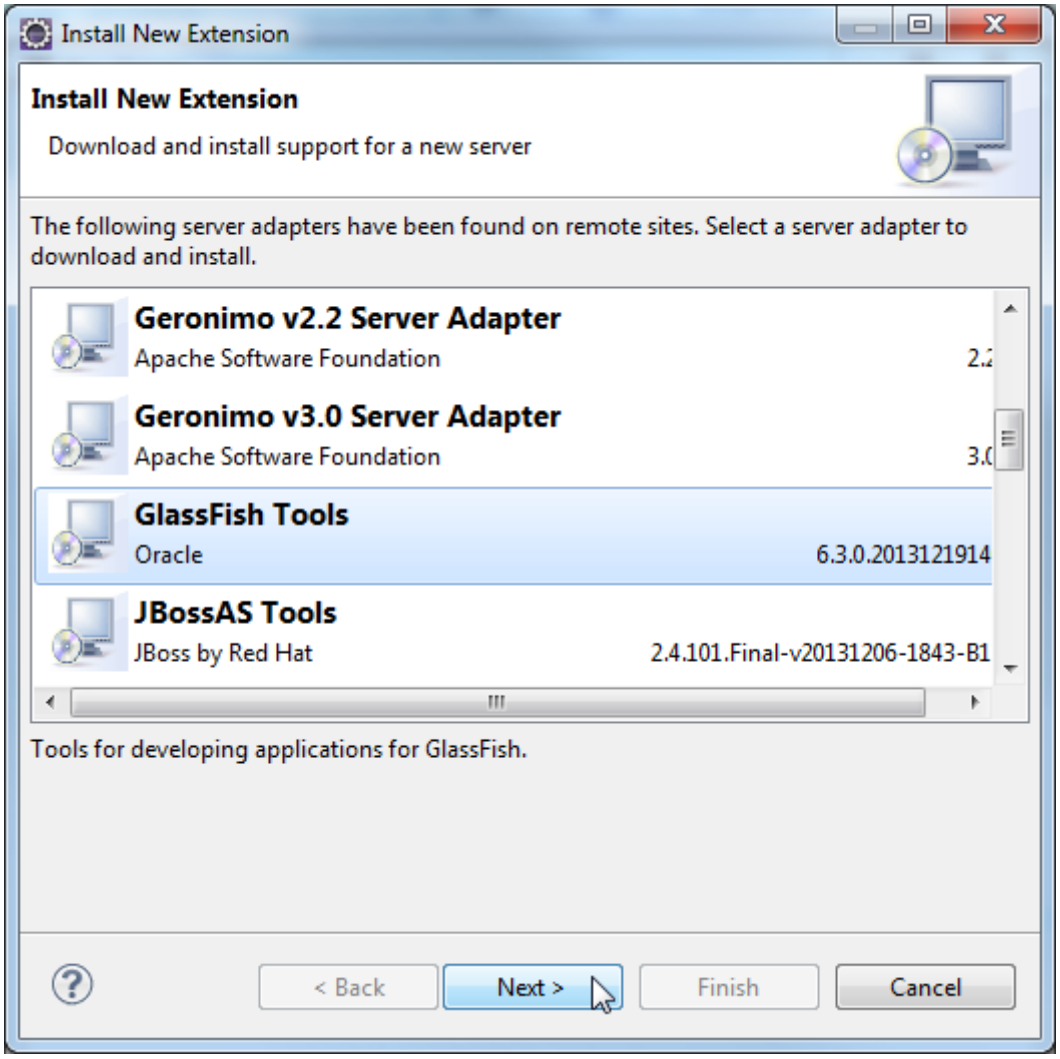
Es wird ein Restart vorgeschlagen, der durchzuführen ist. Danach wird mit dem folgenden Unterkapitel fortgefahren.



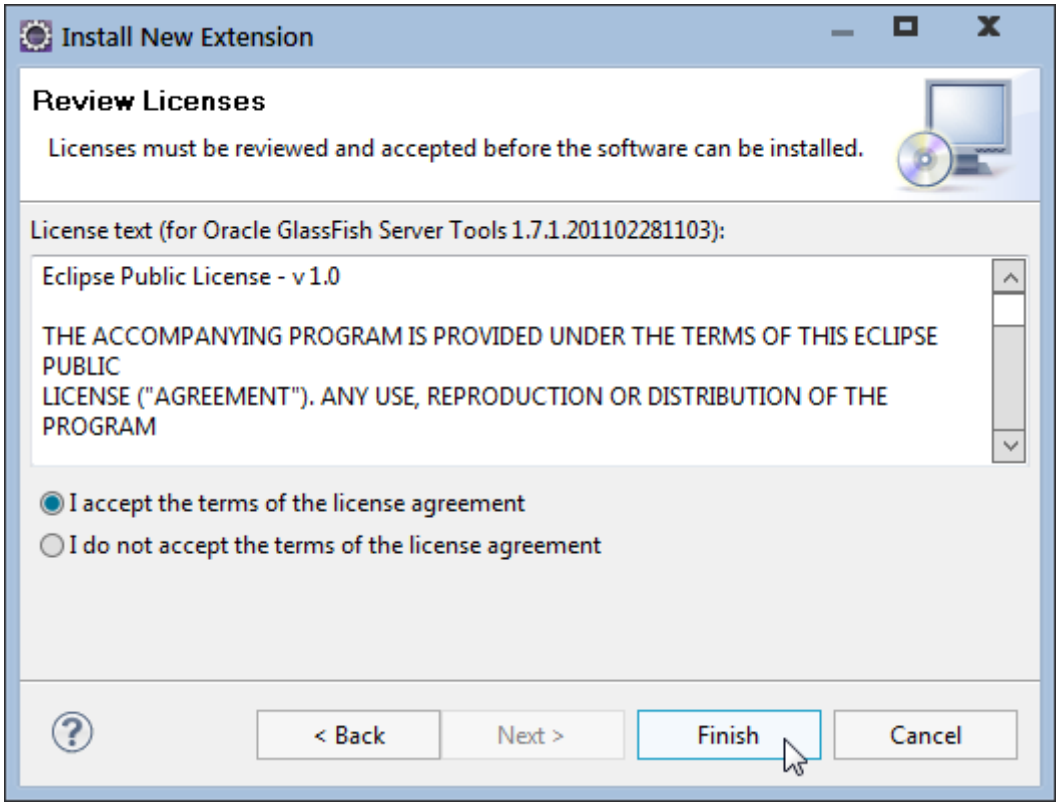
Ist GlassFish noch nicht unter den schon vorhandenen Adaptern zur Einbindung von Servern, so dass rechts-oben der Link „Download additional server adapters“ geklickt wird.



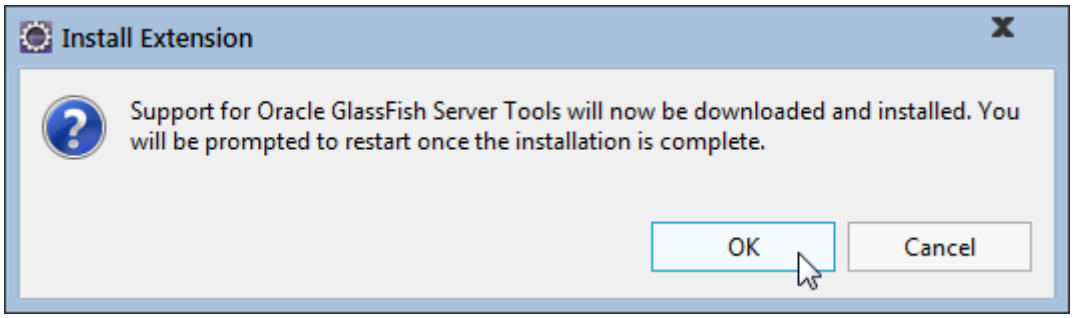
Nach etwas Wartezeit werden weitere Adapter angegeben. Zunächst wird oben „GlassFish Tools“ und dann unten „Next>“ angeklickt.



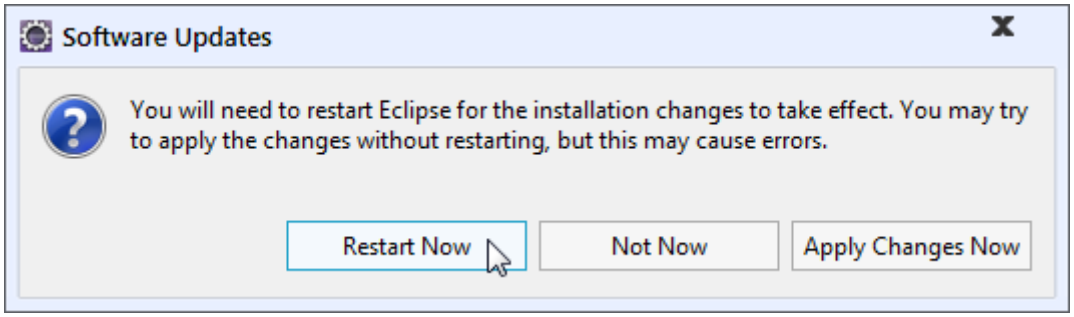
Die Lizenzbedingungen werden gelesen, die Markierung neben „I accept…“ gesetzt und „Finish“ gedrückt.



Man wird über die anstehende Installation informiert und drückt „OK“.

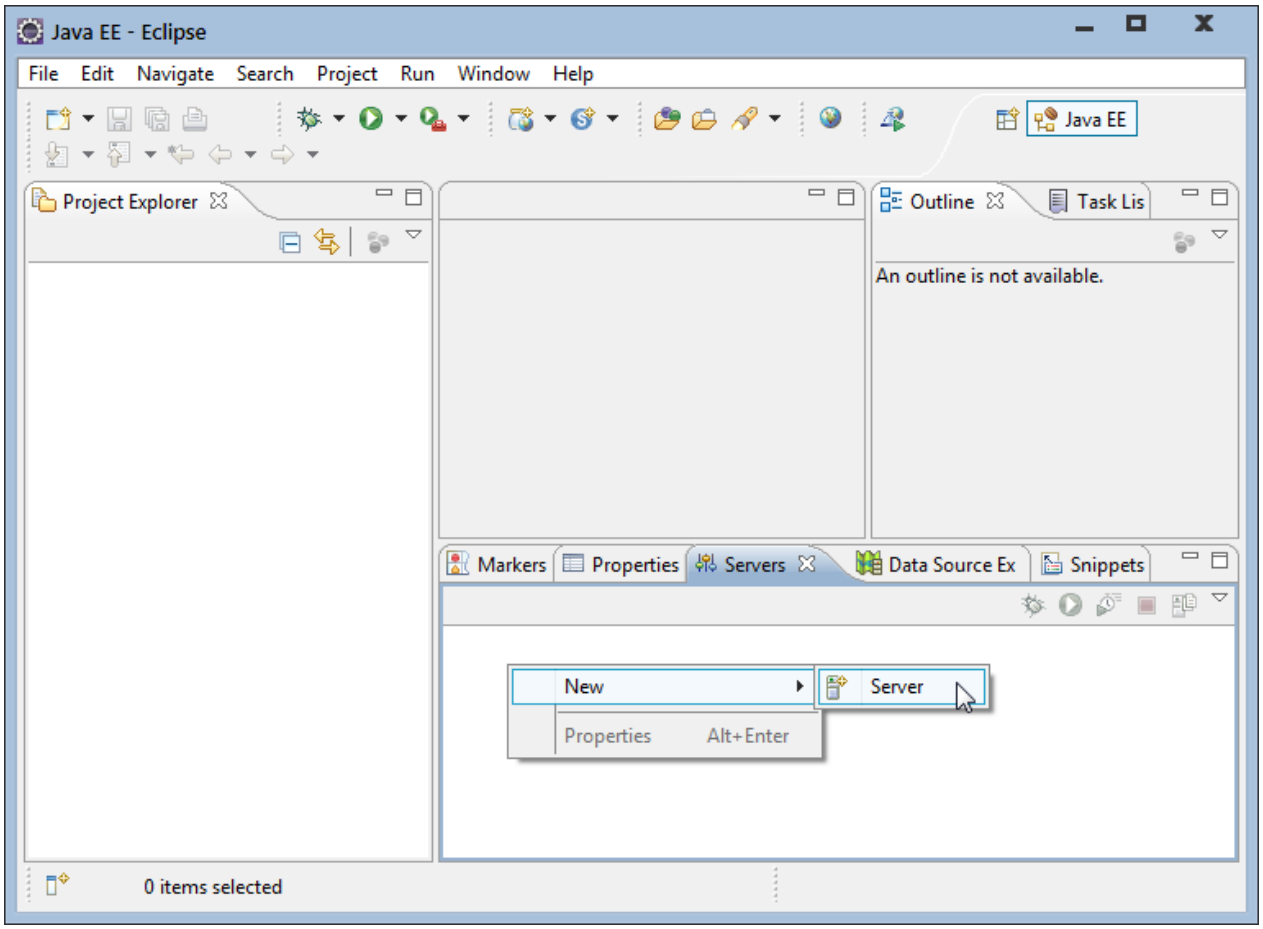


Die Installation dauert etwas. Die abschließende Empfehlung zum Neustart von Eclipse sollte man annehmen.

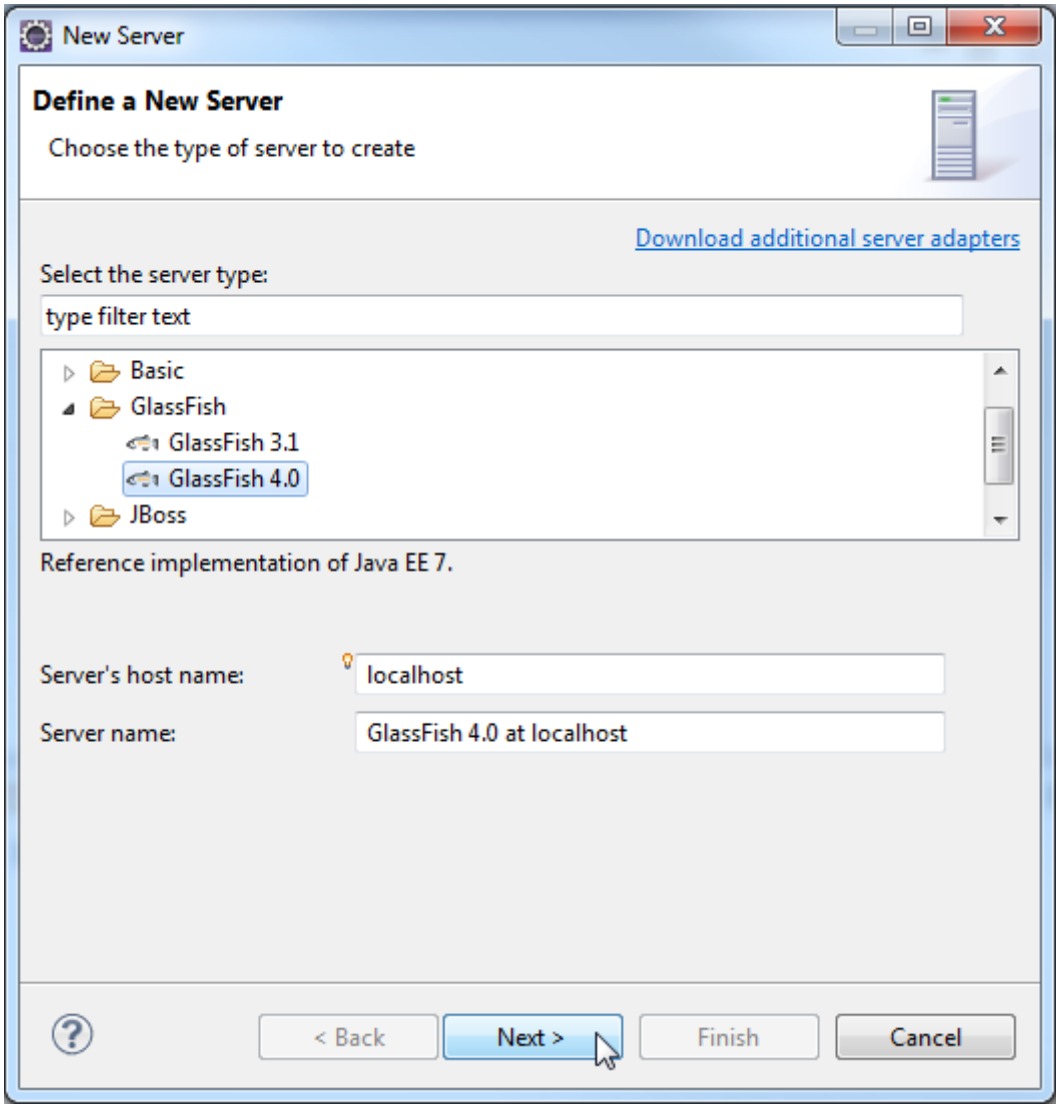


18.5 Einrichtung von GlassFish in Eclipse JEE

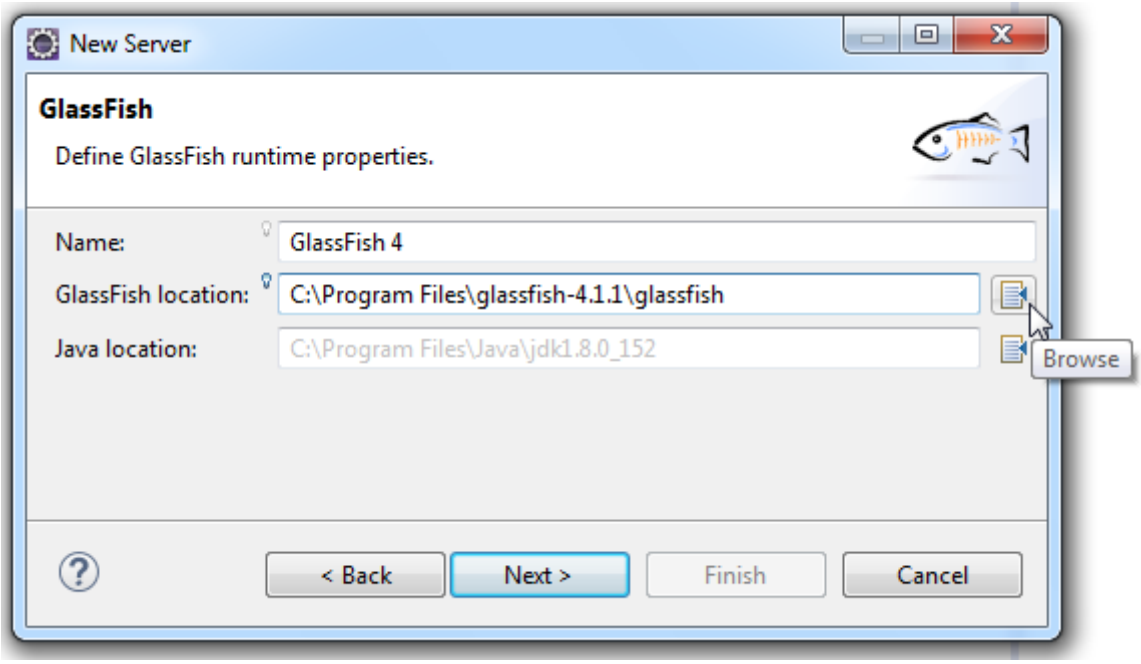
Nachdem die Integration abgeschlossen ist, wird wieder in den Reiter „Servers“ geklickt und „New“ mit „Server“ gewählt.



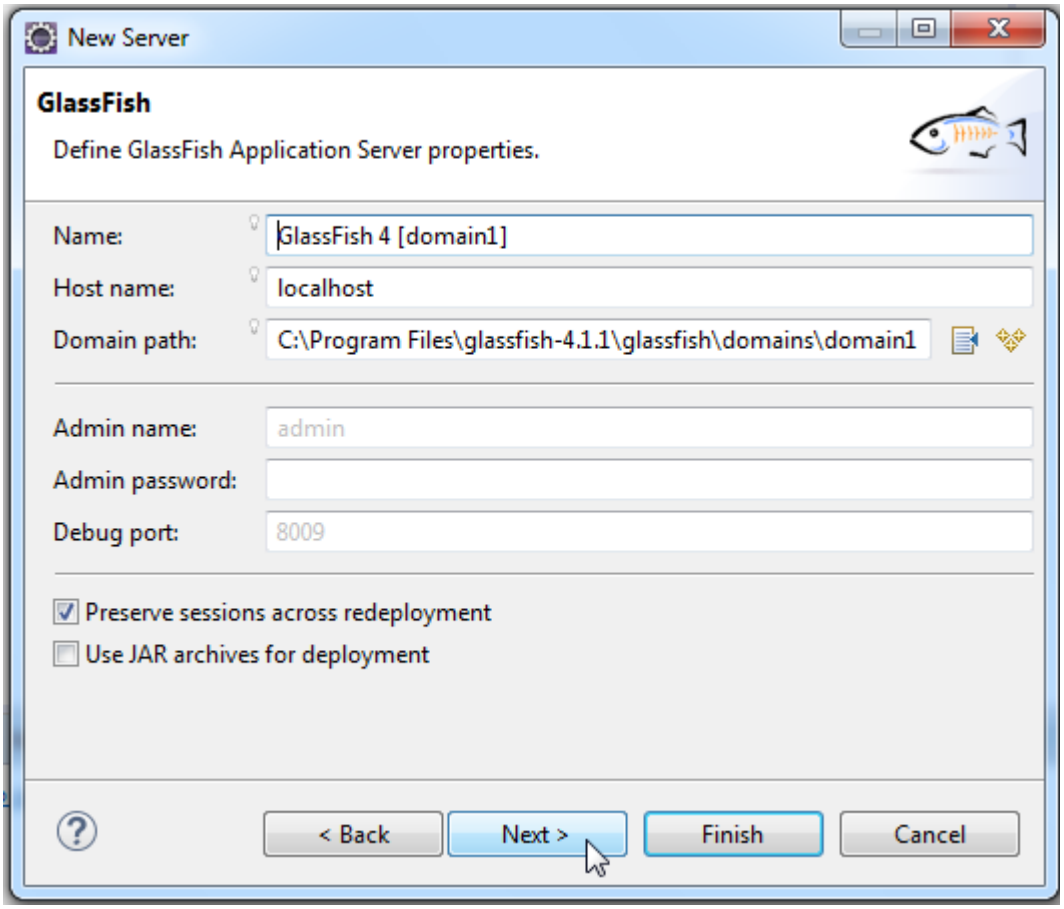
In der Auswahl findet man jetzt den GlassFish in der gewünschten Variante, wählt ihn aus und klickt „Next>“.



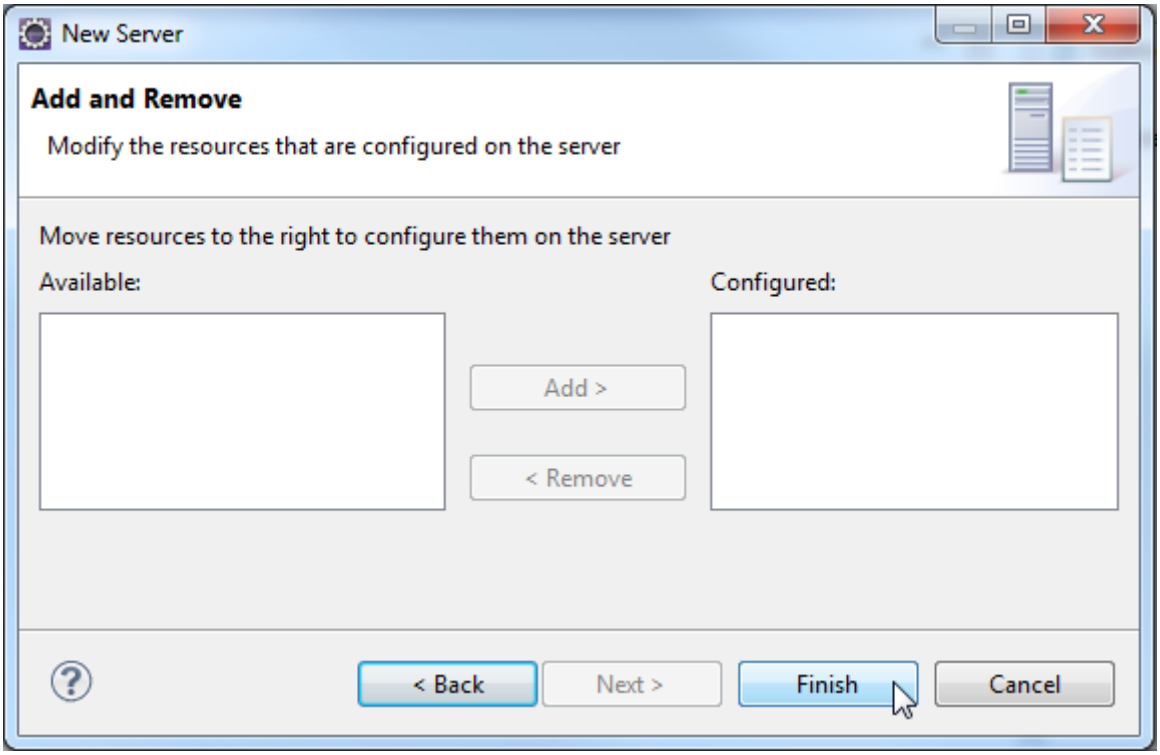
Im folgenden Fenster werden das GlassFish- und das Java-Installationsverzeichnis eingetragen, die über die Auswahlknöpfe rechts wählbar sind. Typischerweise wird beim Eintrag des GlassFish-Verzeichnisses, die Java location automatisch ergänzt. Es wird „Next>“ geklickt



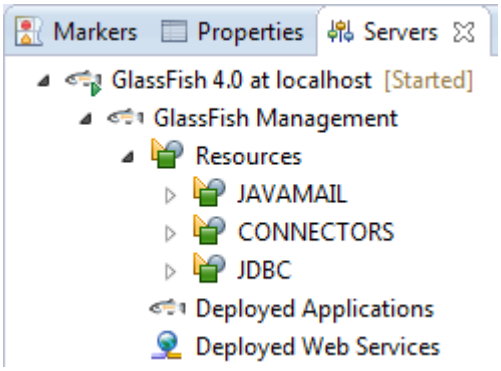
Die Einstellungen im folgenden Fenster können übernommen werden. Danach kann die Installation mit „Finish“ abgeschlossen werden, hier wird zur genaueren Betrachtung „Next>“ geklickt.



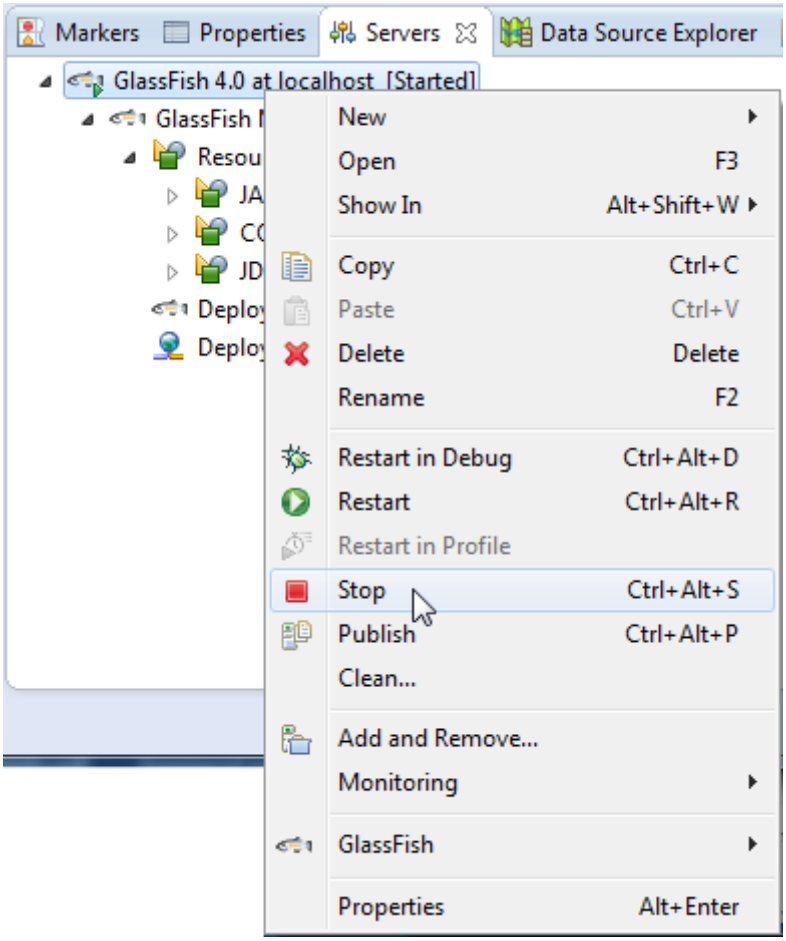
Da es noch keine installierten Applikationen gibt, kann die Konfiguration jetzt mit „Finish“ abgeschlossen werden.



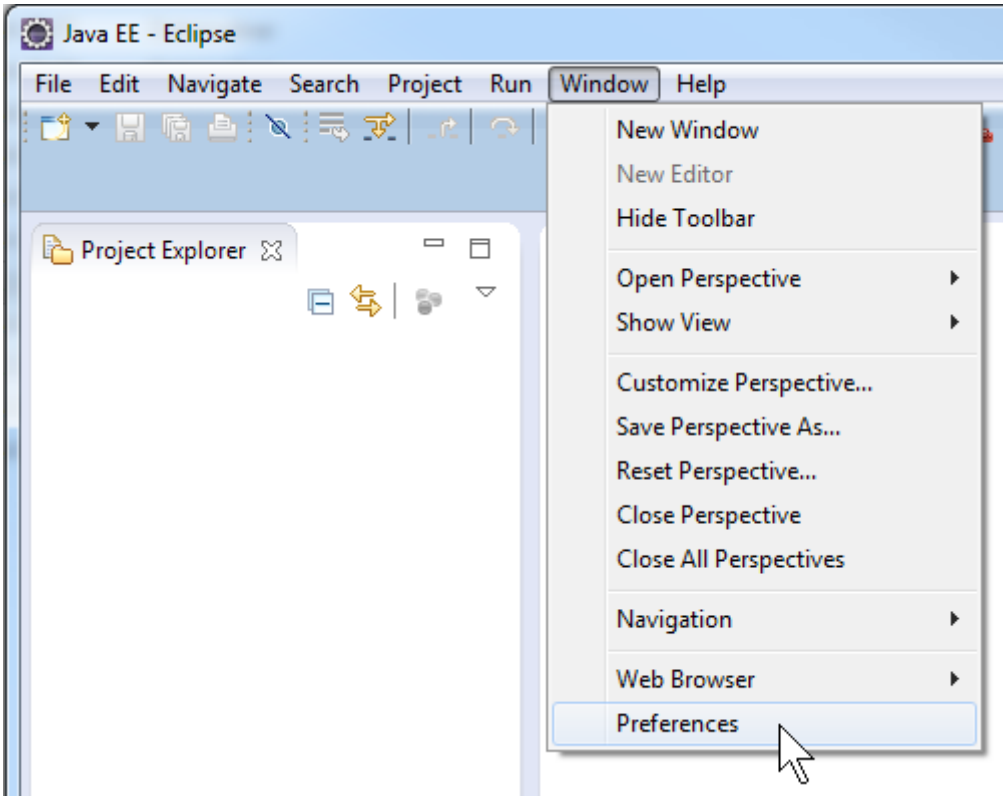
Nun befindet sich im Reiter Servers der jetzt nutzbare GlassFish. Sollte man GlassFish gerade nach dem vorgeschlagenen Verfahren installiert haben, kann es sein, dass er aktuell läuft, was an dem kleinen grünen Dreieck in der folgenden Abbildung sichtbar ist. Weiterhin erhält man Informationen über installierte Applikationen.



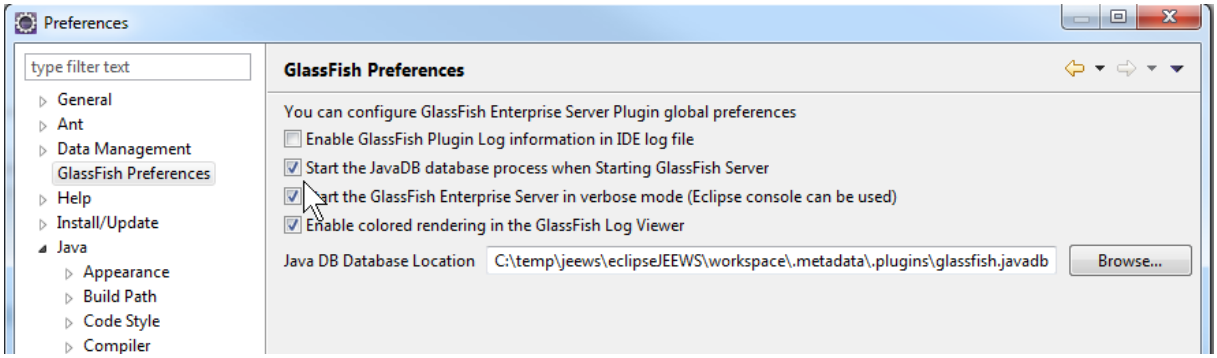
Durch einen Rechtsklick auf den Elementen können diese bearbeitet werden. So kann z. B. der Server auch gestoppt werden.



Meist ist es sinnvoll, die mit dem Server zusammen ausgelieferte Datenbank JavaDB mit dem Server zu starten. Man bedenke allerdings, dass es durchaus um zwei getrennte Produkte handelt und hier jede andere Datenbank eingebunden sein könnte. Für Java DB gibt es eine direkt Unterstützung in Eclipse. Dazu wird oben zunächst „Window > Preferences“ gewählt.



Nach einem Klick auf „GlassFish Preferences“ wird der Haken bei „Start the JavaDB database when starting GlassFish Server“ gesetzt und unten bestätigt. Der untere Pfad-Eintrag hängt von der gewählten GlassFish-Installation ab und muss nicht geändert werden.



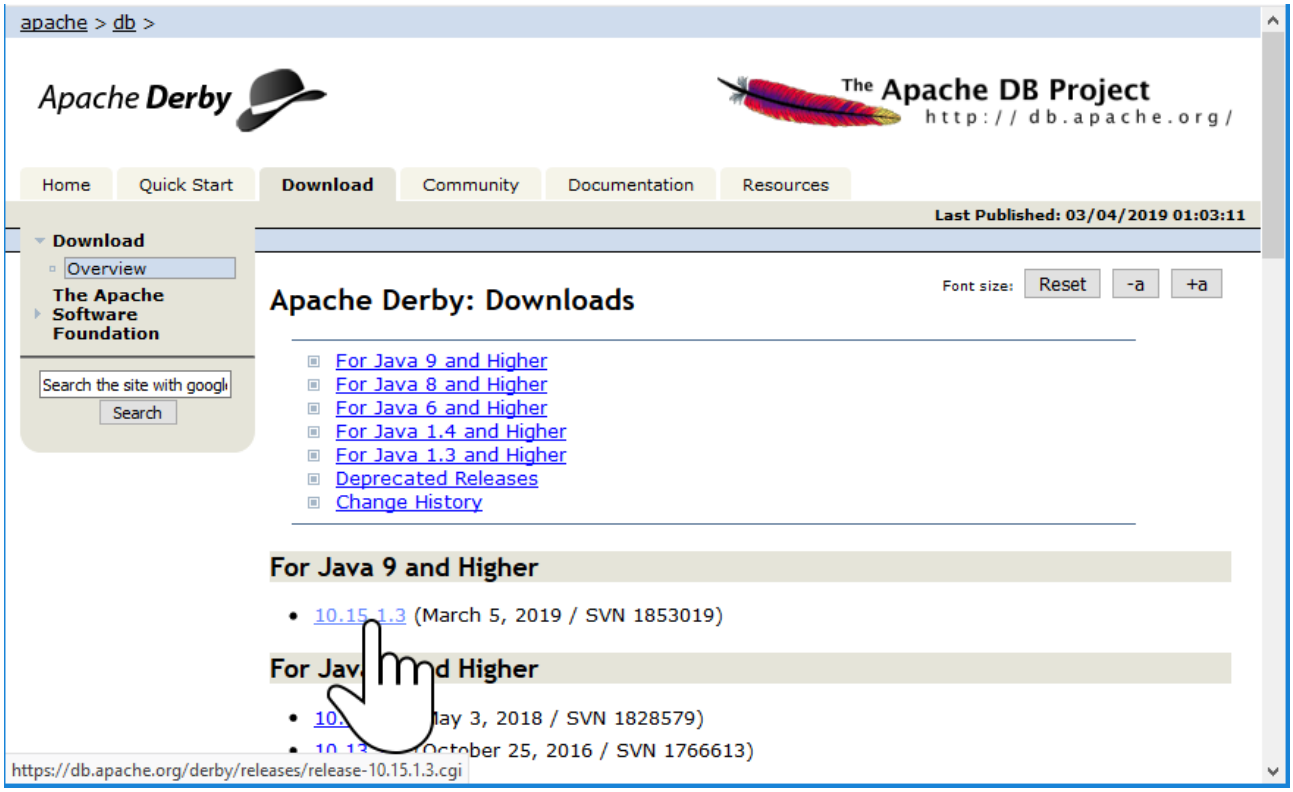
19 Datenbankanbindung und JPA (4.11.0)

In diesem Abschnitt wird ein direkter Weg zur Nutzung einer Apache Derby-Datenbank und die Nutzung der JPA-Realisierung EclipseLink im JEE-Umfeld beschrieben, wobei dieser Ansatz auch in der klassischen Java-Version von Eclipse genutzt werden kann. Generell erfolgt eine Datenbankanbindung über einen JDBC-Treiber, der vom Datenbankhersteller mitgeliefert wird, der typischerweise als jar-Datei zur Nutzung in das Projekt eingebunden wird. Weiterhin erfolgen der Start und die Terminierung des Datenbankmanagementsystems, das die eigentlichen Datenbanken verwaltet, üblicherweise außerhalb von Eclipse. In diesem Abschnitt wird dazu nur ein kurzer Überblick gegeben, da der Focus hier auf der einfachen JDBC und JPA-Unterstützung liegt. Generell gilt es für Datenbanken, dass es sich lohnt, nach Eclipse-Erweiterungen zu suchen, die die Datenbanknutzung von Eclipse aus unterstützen, was das Starten und Terminieren der Datenbanken angeht, aber auch das Anlegen von Datenbanken mit Tabellen, sowie die Ausführung von SQL-Anfragen und der Transaktionssteuerung ermöglicht. Detailliertere Informationen z. B. zur Trigger-Entwicklung sind in http://home.edvsz.hs-osnabrueck.de/skleuker/querschnittlich/DBNutzung.pdf nachzulesen.

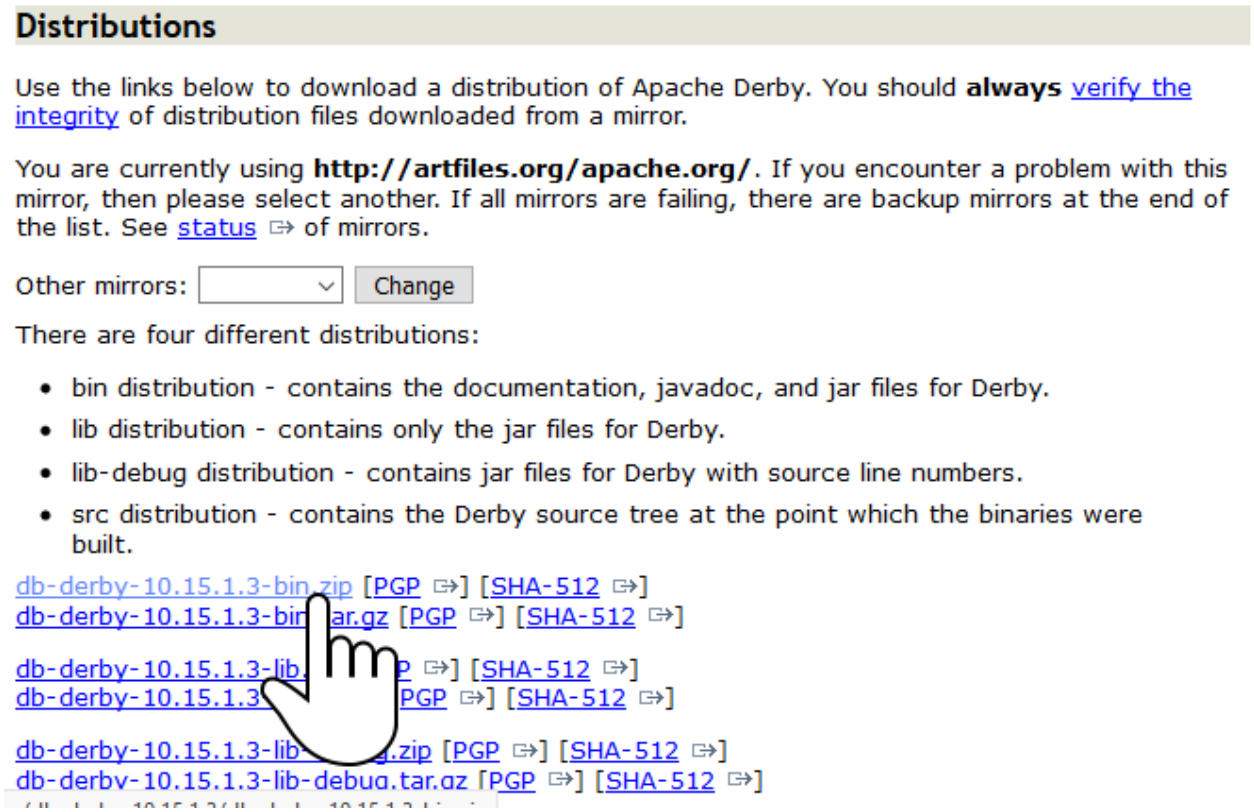
19.1 Installation und Start von Apache Derby

Die Installation beginnt mit einem Klick auf den Download-Reiter auf der Webseite https://db.apache.org/derby/.

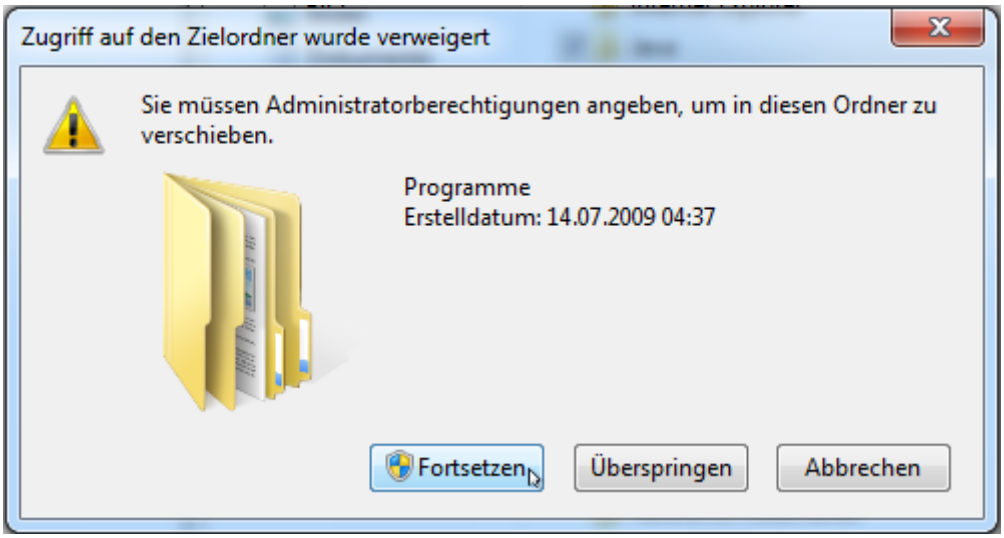
Es wird eine aktuelle zur installierten Java-Version passende Derby-Version angeklickt.



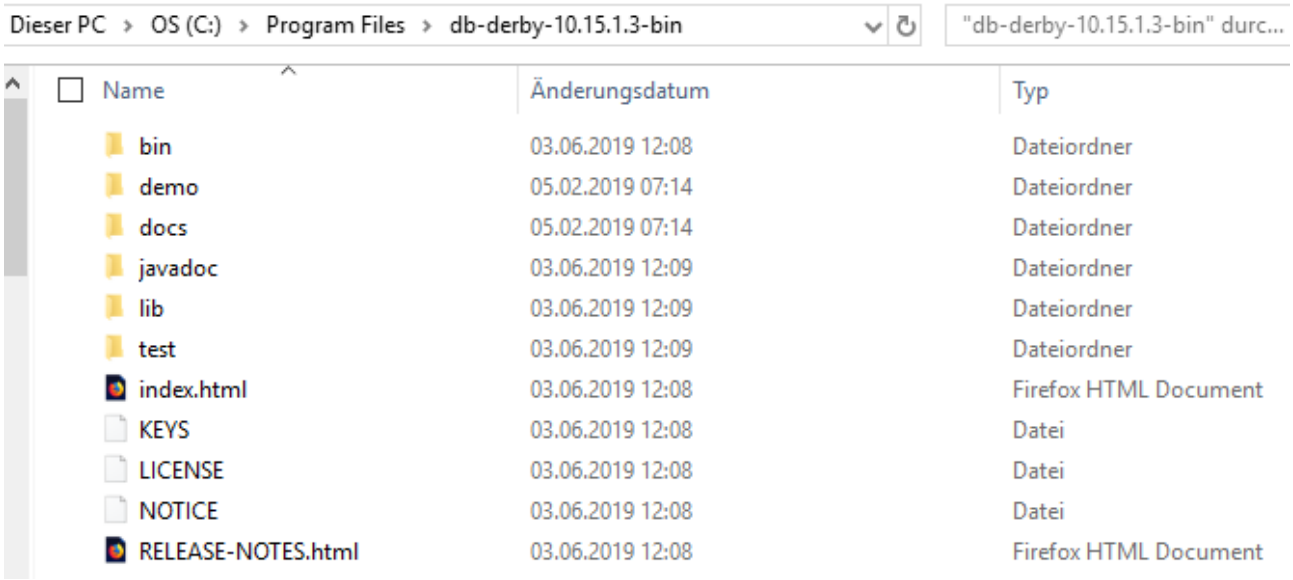
Es wird die aktuelle …bin.zip heruntergeladen.



Die Zip-Datei wird ausgepackt und das entstehende Verzeichnis an einen sinnvollen Ort geschoben, hier z. B. C:\Programme (oder C:\Program Files). Für diesen Zielort müssen Administatorrechte vorliegen, was für andere Orte nicht notwendig ist.



Das entstehende Verzeichnis sieht wie folgt aus.



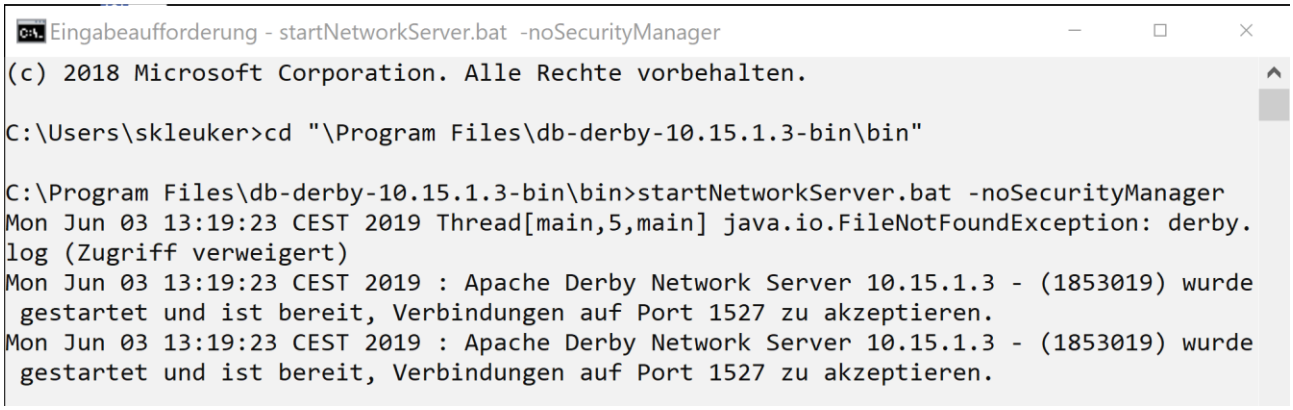
Soll intensiver direkt mit Derby gearbeitet werden, ist die Aufnahme des bin-Verzeichnisses in die PATH-Variable von Windows sinnvoll, wird aber sonst nicht benötigt. Bei direkter Nutzung muss die Systemvariable DERBY_HOME auf das Installationsverzeichnis gesetzt werden.

19.2 Start und Stopp der Datenbank

Zur Nutzung muss Java installiert und in der PATH-Variablen eingetragen sein. Zum Starten wird in einem Konsolenfenster in das bin-Verzeichnis von Derby gesteuert und dort

```
startNetworkServer.bat -noSecurityManager
```

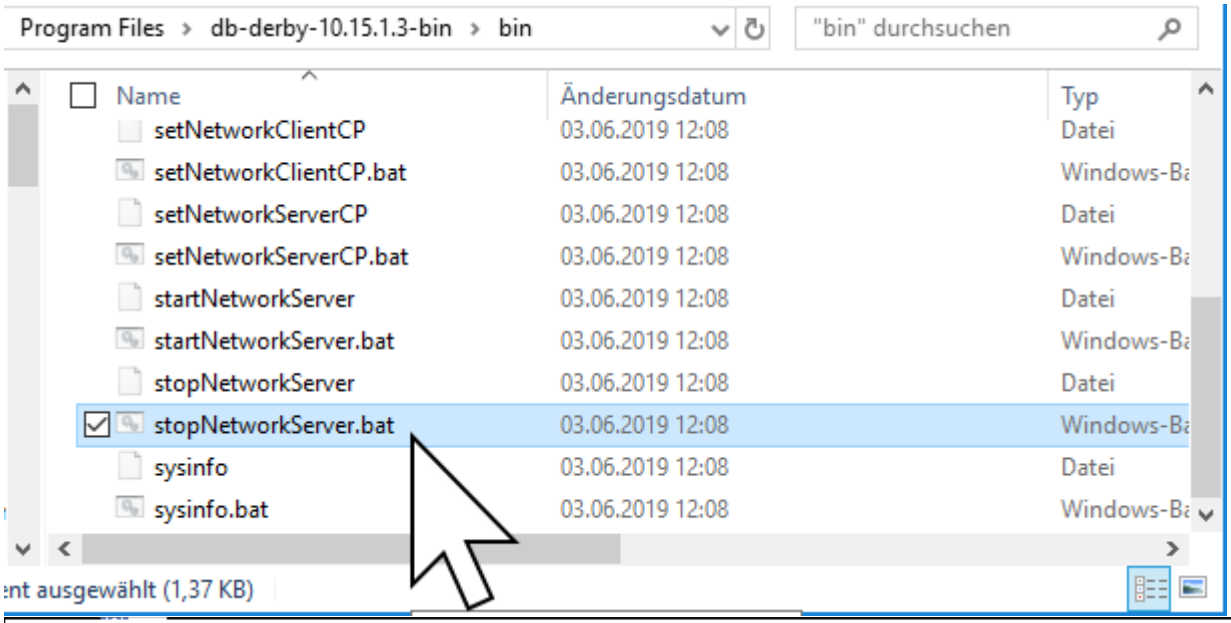
aufgerufen. Der Parameter ist relevant, da im späteren Verlauf Trigger für die Datenbank geschrieben werden sollen, die in die Datenbank integriert werden. Alternativ wären die Security-Einstellungen der Java-Installation anzupassen. Vor der Derby-Version 10.15 war der Parameter nicht notwendig. Für den Startbefehl kann eine eigene Batch-Datei, z. B. mit Verknüpfung zur Oberfläche angelegt werden.

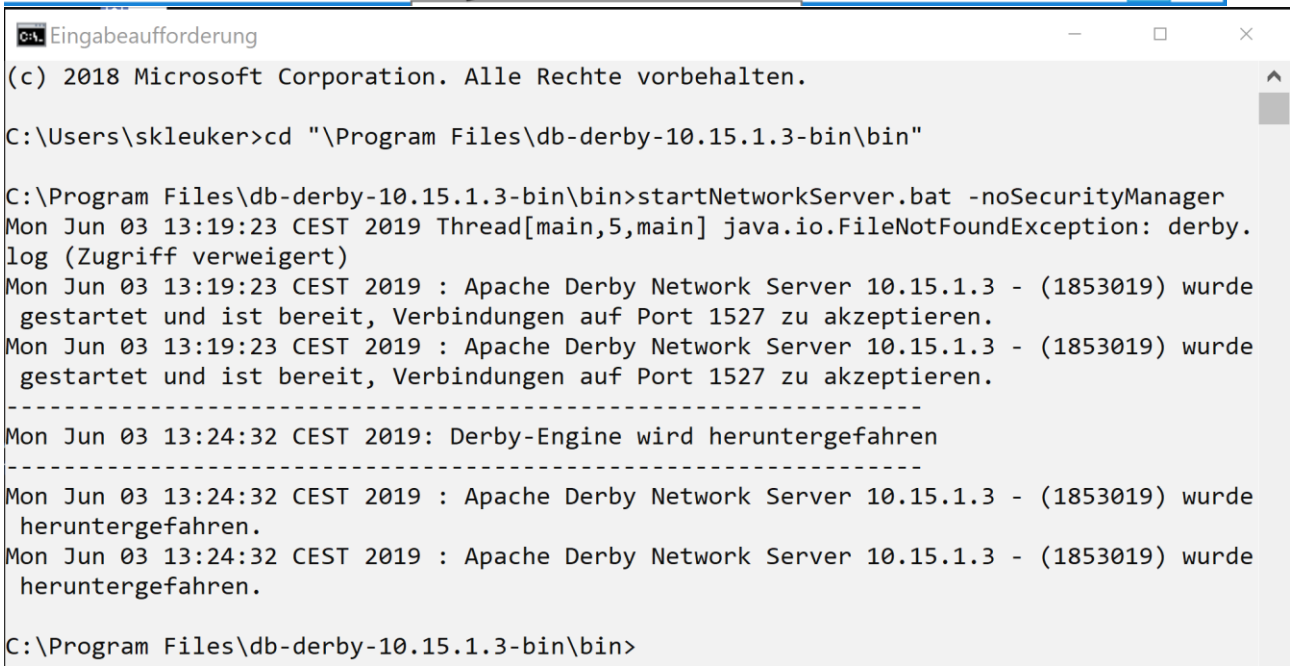


Bei einer Meldung der folgenden Form ist die installierte Java-Version zu alt für die installierte Derby-Version. Java sollte dann aktualisiert werden.

```
C:\Program Files\db-derby-10.15.1.3-bin\bin>startNetworkServer.bat -noSecurityManager
Error: A JNI error has occurred, please check your installation and try again
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/apache/derby/drda/NetworkServerControl has been compiled by a more recent version of the Java Runtime (class file version 53.0), this version of the Java Runtime only recognizes class file versions up to 52.0
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:763)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
```

Das Konsolenfenster darf nicht geschlossen werden, da sonst auch die Datenbank geschlossen wird. Sauberer wird die Datenbank durch die Ausführung von „stopNetworkServer.bat“ im bin-Verzeichnis beendet. Danach kann das Konsolenfenster geschlossen werden.



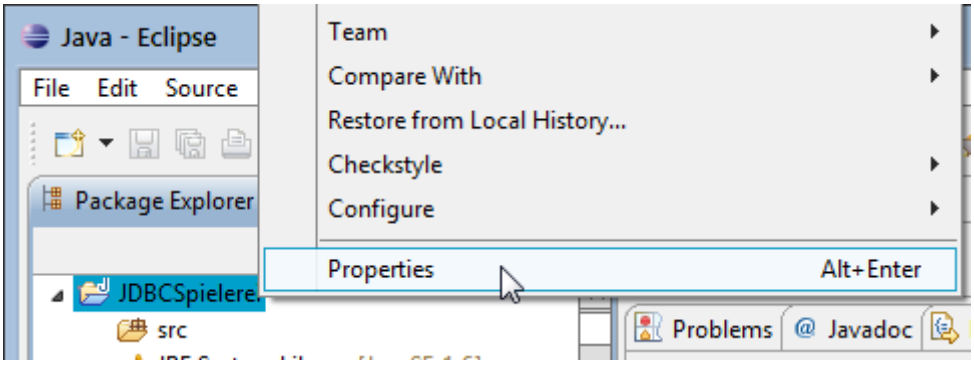


19.3 Direkte JDBC-Nutzung

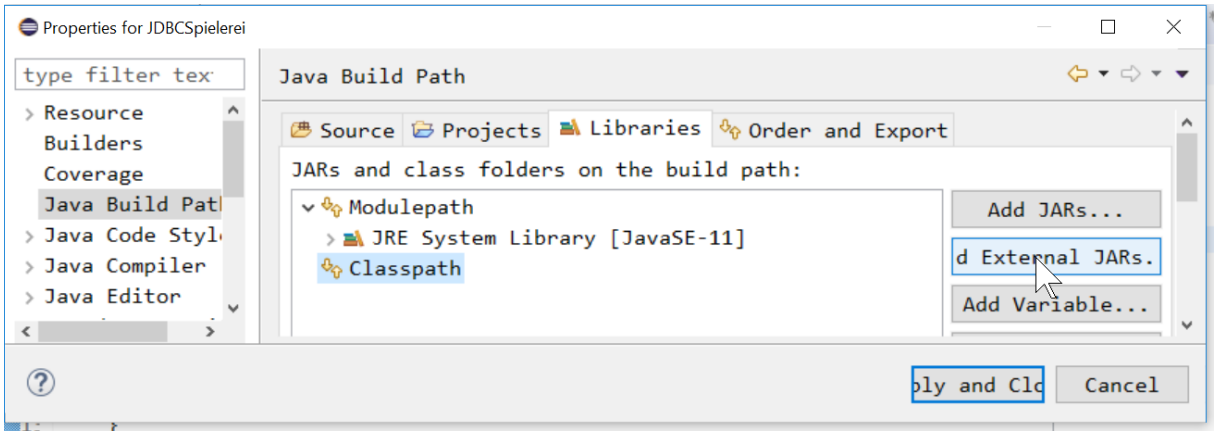
Hinweis: Weitere Details können eventuell http://home.edvsz.hs-osnabrueck.de/skleuker/querschnittlich/DBNutzung.pdf entnommen werden.

Die folgenden Aktionen sind auch ohne die JEE-Variante von Eclipse möglich und zeigen den einfachen Datenbankzugriff über JDBC, dabei wird von einer laufenden Datenbank, wie im vorherigen Abschnitt beschrieben, ausgegangen.

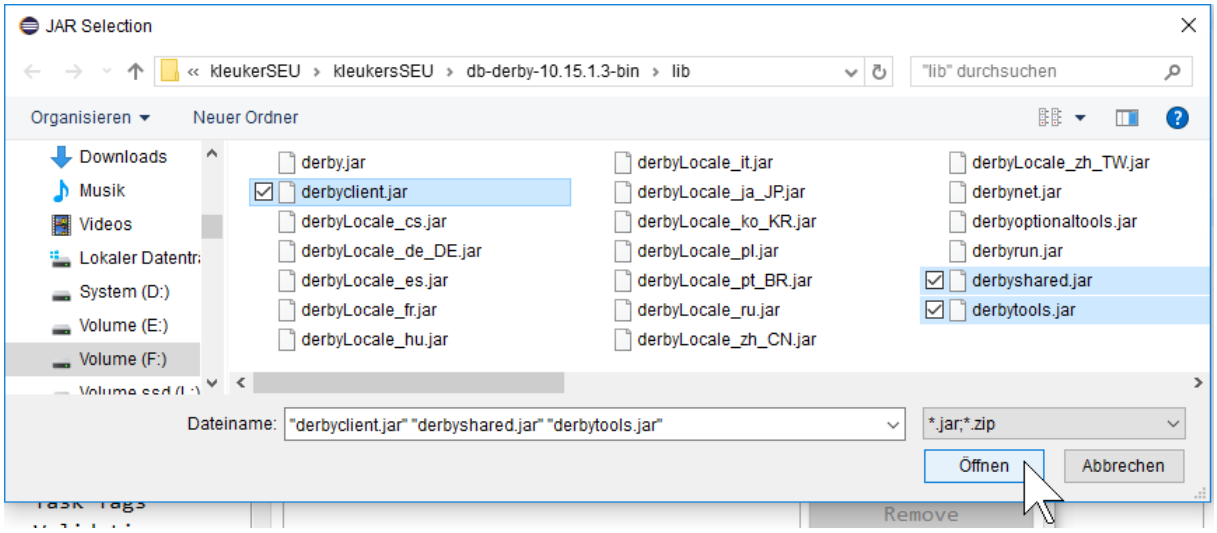
Zunächst wird der JDBC-Treiber zu einem neuen einfachen Java-Projekt JDBCSpielerei hinzugefügt. Dies erfolgt z. B. durch einen Rechtsklick auf dem Projekt, nachdem dann im Menü der Punkt „Properties“ ausgewählt wird.



Auf der linken Seite wird „Java Build Path“, dann der Reiter „Libraries“, dann „Classpath“ und dann „Add External JARs…“ geklickt.



Nun wird zur installierten Datenbank und dem zugehörigen Treiber manövriert, und dieser über „Öffnen“ und „Apply and Close“ in das Projekt eingebunden. Für Apache Derby werden die Dateien derbyclient.jar, derbyshared.jar und derbytools.jar benötigt, die mit gedrückter Strg-Taste zusammen ausgewählt werden können.



Systematischer wäre ein Ansatz im Projekt einen Unterordner (Folder) lib einzurichten, den Treiber in dieses Verzeichnis zu kopieren und dann, wie in „5.1 Einzelne Dateien importieren“ beschrieben, einzubinden.

Nun wird ein einfaches Programm geschrieben, das zeigt, wie eine Datenbankverbindung aufgebaut wird. Der Zusatz „ ;create=true“ garantiert, dass die Datenbank angelegt wird, falls sie noch nicht existieren sollte. Falls sie existiert, wird dieser Parameter ignoriert.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DBTest {

	private Connection con;
	private Statement stmt;

	public void verbinden() throws SQLException {
		con = DriverManager.getConnection(
				"jdbc:derby://localhost:1527/"
				+ "F:\\workspaces\\datenbanken\\Hochschule;create=true"
				  ,"kleuker", "kleuker");
		stmt = con.createStatement();
		con.setAutoCommit(false);
	}

	public void datenEintragen() throws SQLException {
		boolean tabelleXExistiert = false;
		ResultSet rs = con.getMetaData().getTables(null, null, null, null);
		while (rs.next())
			if (rs.getString(3).equals("X"))
				tabelleXExistiert = true;
		if (!tabelleXExistiert) {
			stmt.execute("CREATE TABLE X ( id INTEGER, name VARCHAR(10))");
			stmt.execute("INSERT INTO X VALUES (42,'Ute')");
			stmt.execute("INSERT INTO X VALUES (43,'Uwe')");
		}
	}

	public void datenLesen() throws SQLException {
		ResultSet rs = stmt.executeQuery("SELECT * FROM X");
		while (rs.next())
			System.out.println(rs.getInt(1) + " :: " + rs.getString(2));
	}

	public void beenden() throws SQLException {
		if (con != null) {
			con.commit();
			con.close();
		}
	}

	public static void main(String[] args) {
		DBTest db = new DBTest();
```

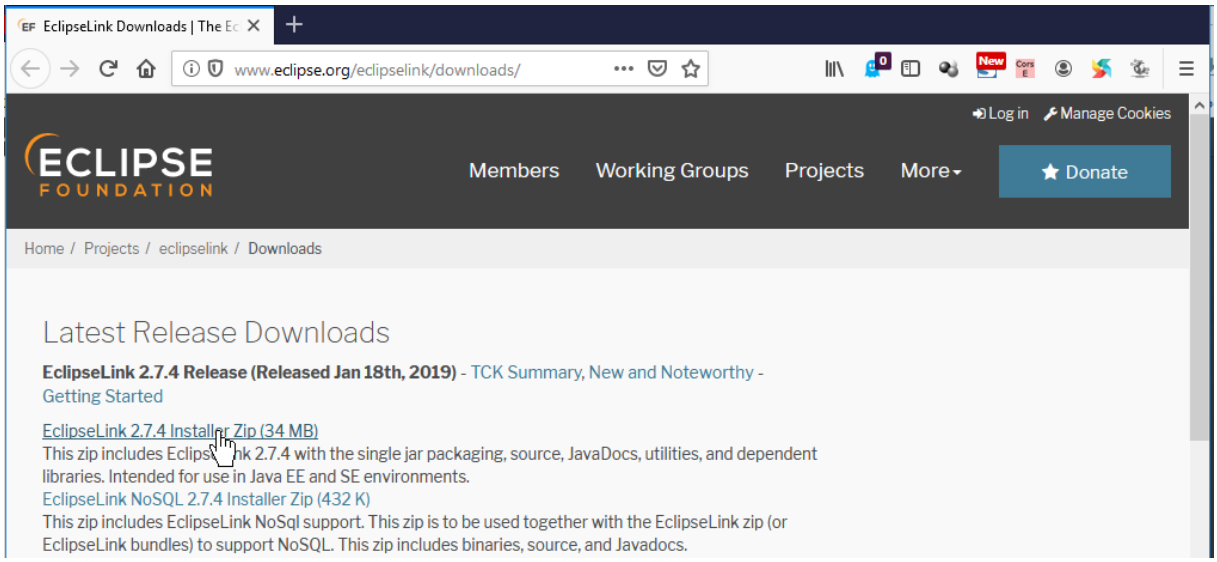
```
                try {
                        db.verbinden();
                        db.datenEintragen();
                        db.datenLesen();
                        db.beenden();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }
}
```

Die Datenbank-Software selbst muss natürlich vorher gestartet werden. Das Programm liefert folgende Ausgabe:

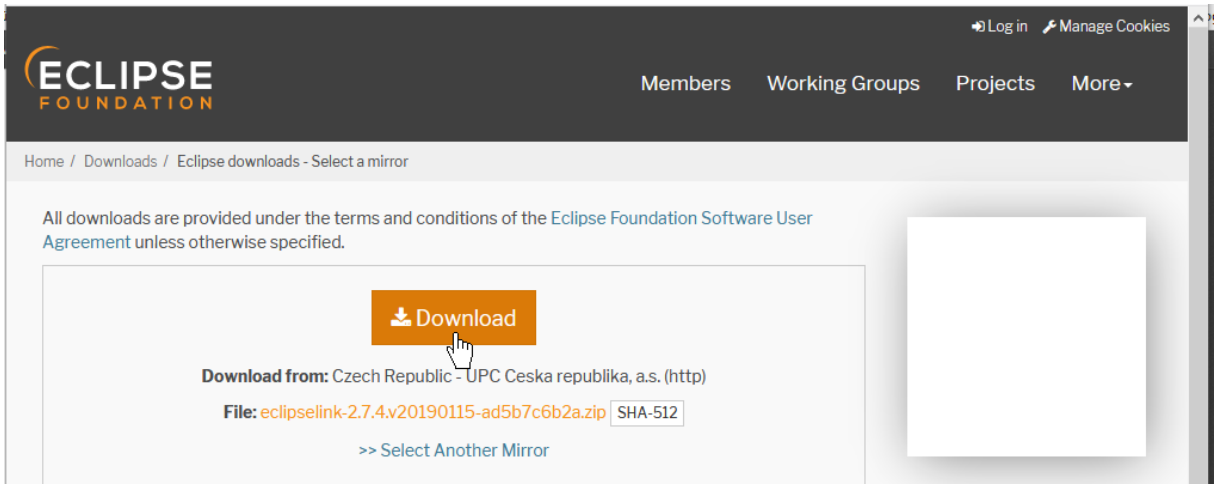
```
42 :: Ute
43 :: Uwe
```

19.4 Direkte JPA-Nutzung ohne Eclipse JEE

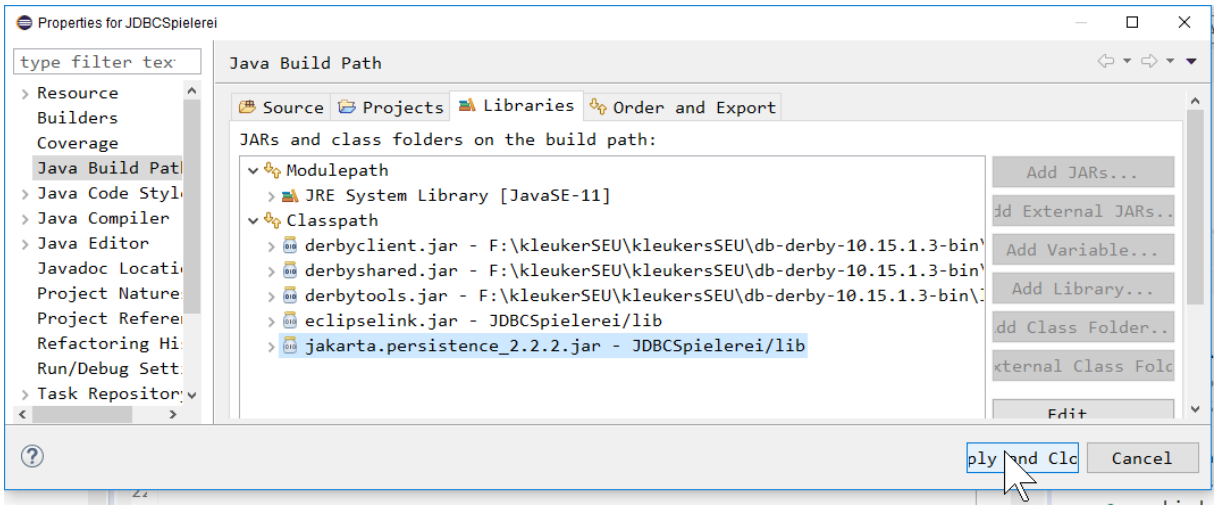
Möchte man in einfachen Programmen JPA nutzen, muss eine JPA-Realisierung in das Projekt eingebunden werden. Hier wird EclipseLink genutzt, das unter http://www.eclipse.org/eclipselink/downloads/ geladen werden kann.



Der Download-Knopf wird geklickt, es kann dann etwas dauern.



Analog wie im vorherigen Abschnitt beschrieben, muss der JDBC-Treiber zum Projekt ergänzt werden. Weiterhin werden zwei Jar-Dateien aus EclipseLink benötigt und in der gleichen Form eingebunden. Man beachte, dass die eine JAR-Datei im Unterordner jlib und die andere Datei im Unterordner jlib/jpa liegt. Insgesamt ergibt sich folgendes Bild bezüglich der verwendeten Bibliotheken.



Nun wird ein kleines Beispiel mit einer Entität programmiert. Es werden die folgenden beiden Klassen angelegt.

```
package entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Mitarbeiter {
 @Id @GeneratedValue(strategy=GenerationType.AUTO)
 private int minr;
 private String name;

 public Mitarbeiter(){} //parameterloser Konstruktor benötigt
```

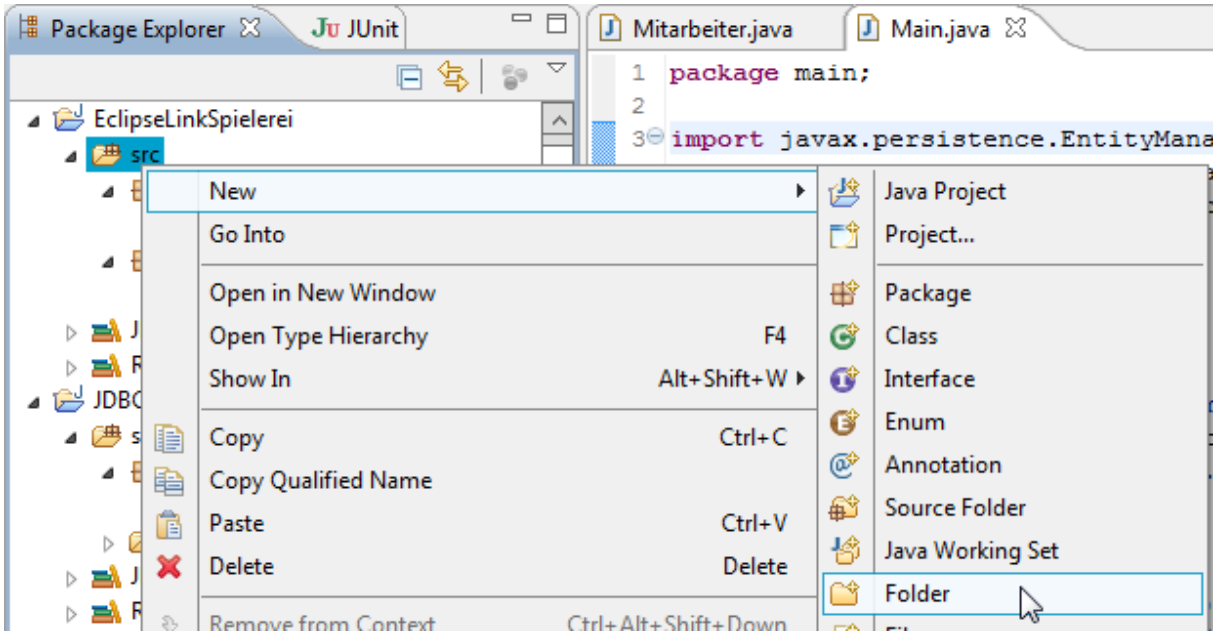
```
  public Mitarbeiter(String name) {
    this.name = name;
  }

  public int getMinr() {return minr;}
  public void setMinr(int minr) {this.minr = minr;}
  public String getName() {return name;}
  public void setName(String name) {this.name = name;}
}
```

und

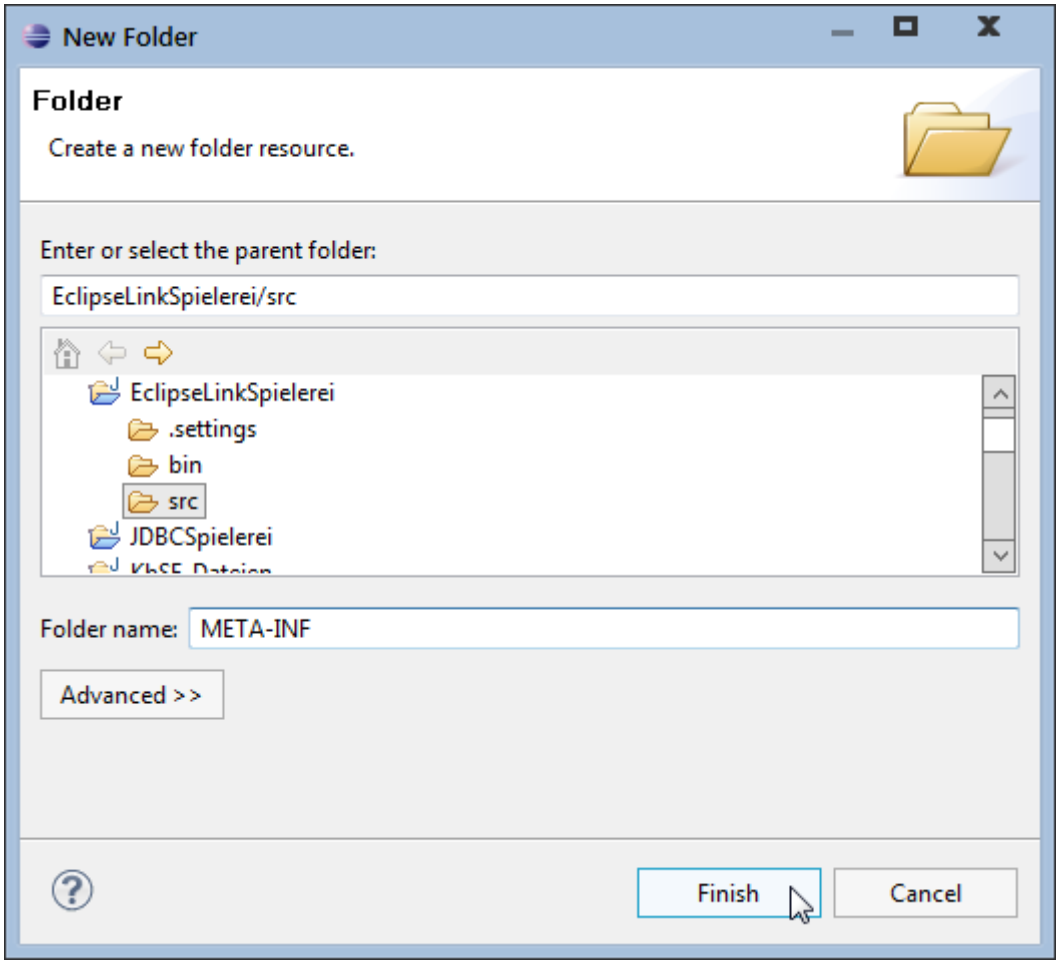
```
package main;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import entity.Mitarbeiter;

public class Main {

	private EntityManagerFactory emf = Persistence
			.createEntityManagerFactory("Spielerei2");
	private EntityManager em = emf.createEntityManager();

	public void beispieldaten() {
		String namen[] = { "Egon", "Erwin", "Ute", "Aische" };
		em.getTransaction().begin();
		for (int i = 0; i < namen.length; i++)
			em.persist(new Mitarbeiter(namen[i]));
		em.getTransaction().commit();
	}

	public void datenZeigen() {
		for (Mitarbeiter m : em.createQuery("SELECT m FROM Mitarbeiter m",
				Mitarbeiter.class).getResultList()) {
			System.out.println(m.getMinr() + ": " + m.getName());
		}
	}

	public void schliessen() {
		if (em != null && em.isOpen()) {
			em.close();
		}
		if (emf != null && emf.isOpen()) {
			emf.close();
		}
	}

	public static void main(String[] args) {
		Main m = new Main();
		m.beispieldaten();
		m.datenZeigen();
		m.schliessen();
```

```
        }
}
```

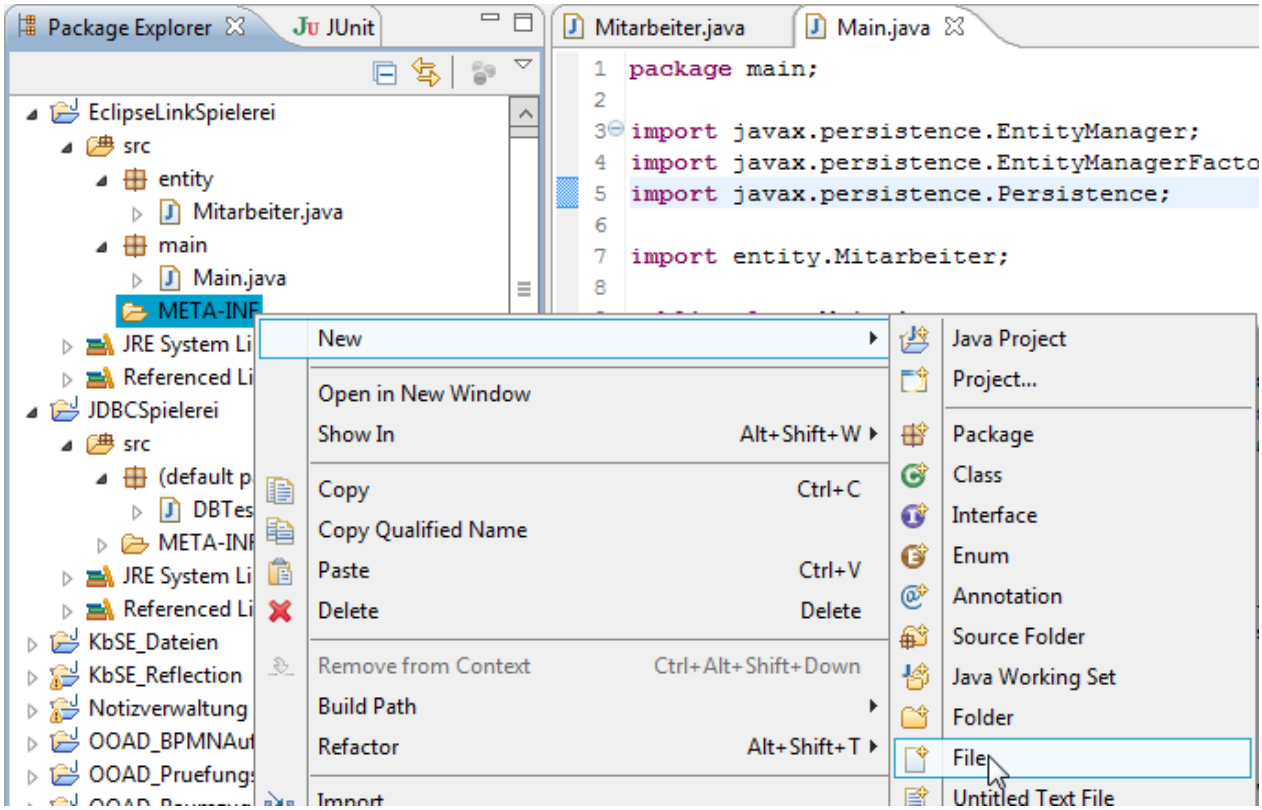
Weiterhin wird eine Konfigurationsdatei mit dem Namen persistence.xml benötigt, die die Informationen zur benutzten Datenbank und zu den zu verwaltenden Entitätsklassen enthält. Neben weiteren Einstellungen kann festgelegt werden, wie mit den Datenbanken umgegangen werden soll. Im laufenden Betrieb wird nichts Besonderes (NONE) passieren. In der laufenden Entwicklung ist es oft sinnvoll, die Tabellen erst wieder vollständig zu löschen und dann wieder neu anzulegen, wie es hier im Beispiel der Fall sein wird. Damit die Konfigurationsdatei gefunden wird, muss diese in einem Unterverzeichnis META-INF des src-Verzeichnisses liegen. Dies kann z. B. durch einen Rechtsklick auf dem src-Ordner und der Auswahl „New“ und „Folder“ geschehen.



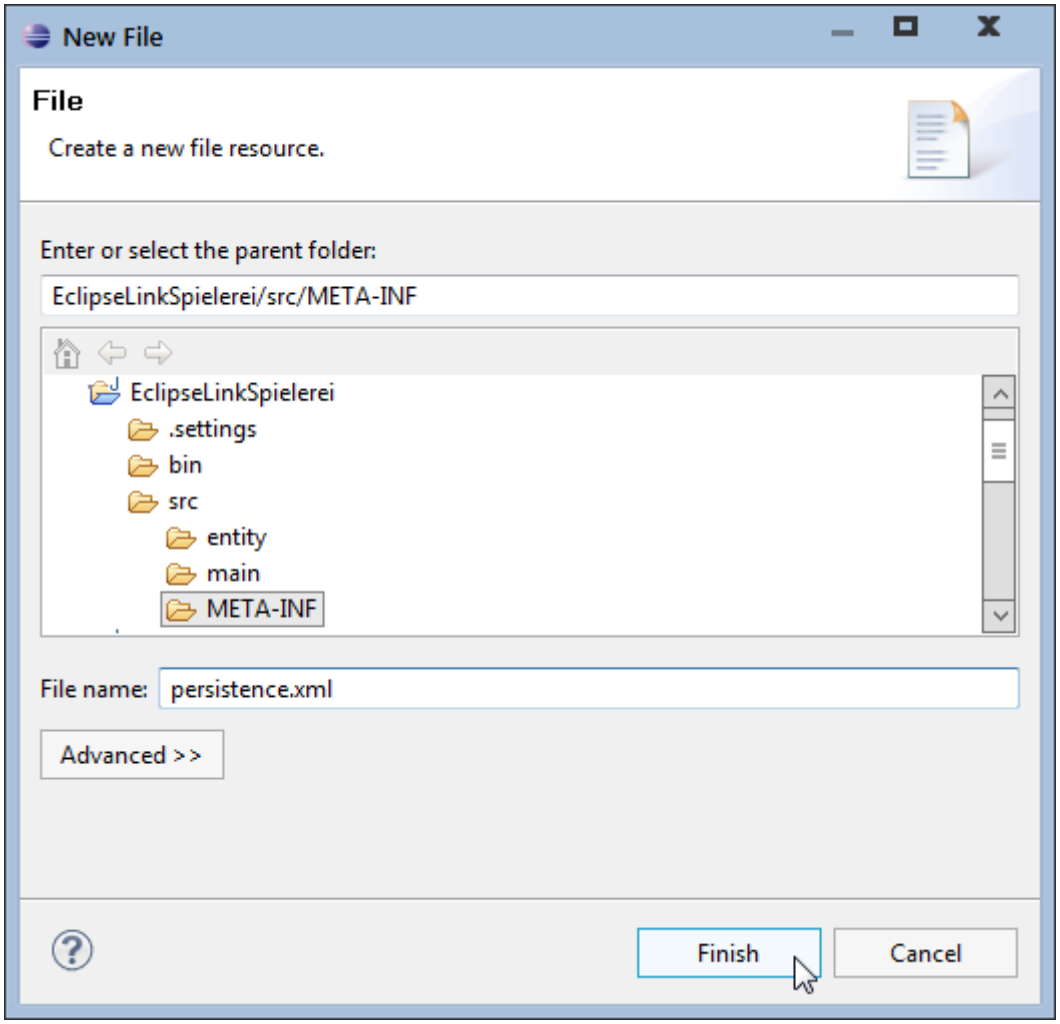
Jetzt wird der Name des Ordners eingegeben und mit „Finish“ bestätigt.



Nun muss noch die XML-Datei erstellt werden, dies kann über einen XML-Editor erfolgen, insofern dieser in Eclipse integriert ist (siehe 11 XML in Eclipse) oder als einfache Textdatei über ein Rechtsklick auf den Ordner META-INF mit „New“ und „File“ erstellt werden.



Danach wird der Dateiname eingegeben und mit „Finish“ die Eingabe abgeschlossen. Man beachte, dass man bei diesem Ansatz keine Unterstützung für die Syntaxprüfung durch Eclipse erhält.

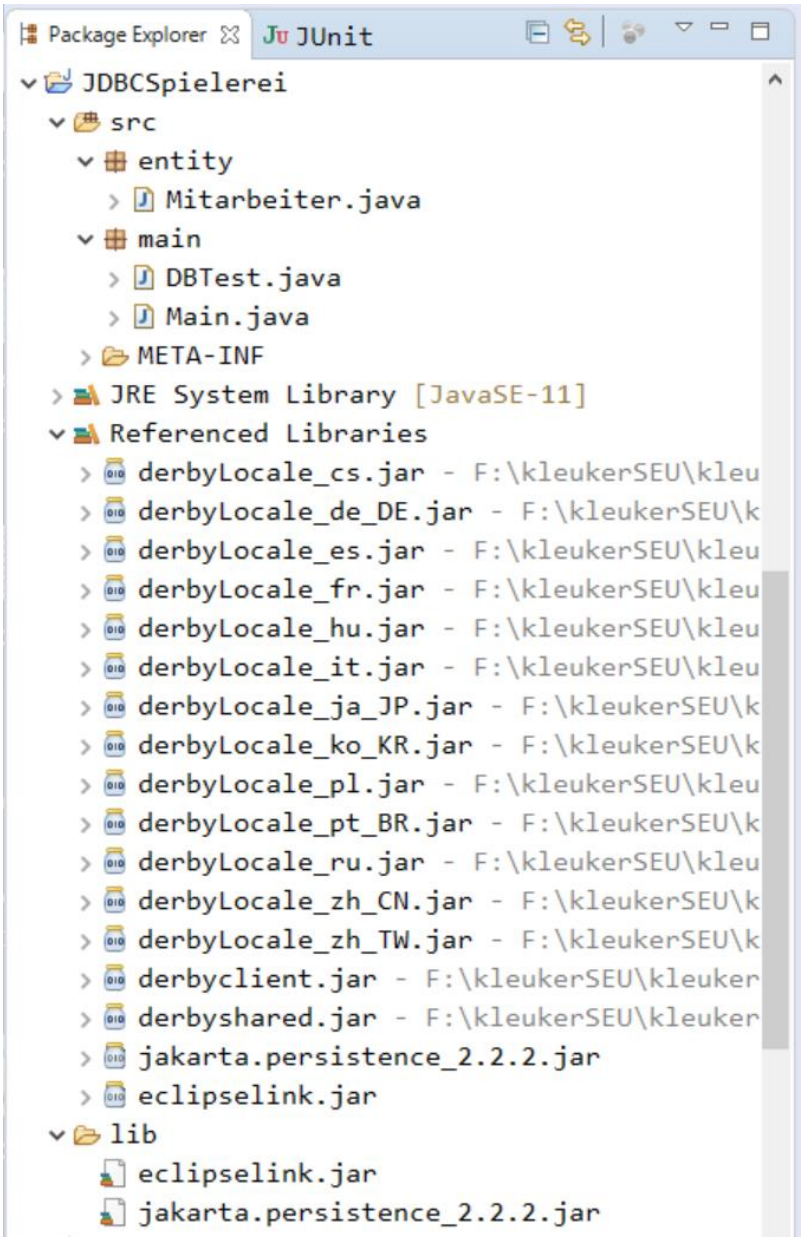


Die benötigte Datei kann wie folgt aussehen, die erstellte Datenbank heißt wie die Persistence-Unit „Spielerei2“.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
          xmlns="http://java.sun.com/xml/ns/persistence"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
          <persistence-unit name="Spielerei2"
                    transaction-type="RESOURCE_LOCAL">
                    <provider>
      org.eclipse.persistence.jpa.PersistenceProvider
  </provider>
                    <class>entity.Mitarbeiter</class>
                    <properties>
                              <property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/F:\\workspaces\\datenbanken\\Spielerei2;create=true" />
                              <property name="javax.persistence.jdbc.password"
                                        value="kleuker" />
                              <property name="javax.persistence.jdbc.driver"
```

```
                        value="org.apache.derby.client.ClientAutoloadedDriver" />
                <property name="javax.persistence.jdbc.user" value="kleuker" />
                <property name="eclipselink.ddl-generation"
                        value="drop-and-create-tables" />
            </properties>
        </persistence-unit>
</persistence>
```

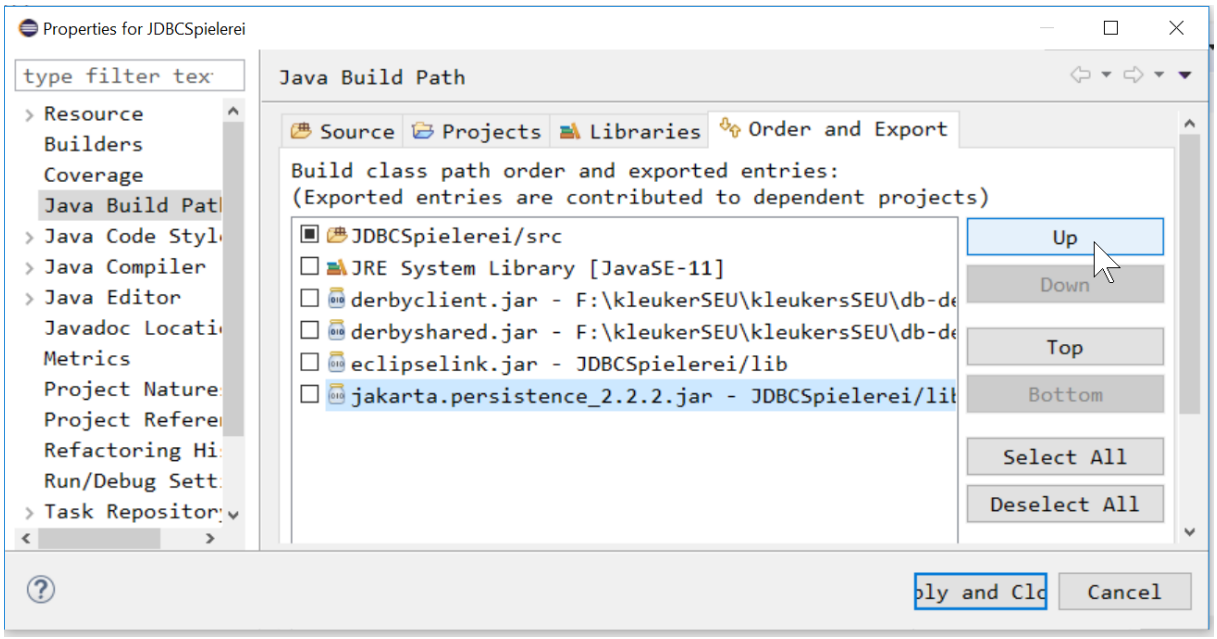
Die Projektstruktur sieht dann, wie in der folgenden Abbildung gezeigt aus, dabei wurden von allen benötigten EclipseLink-Jar-Dateien Kopien im lib-Verzeichnis angelegt.



Die Ausgabe sieht wie folgt aus, dabei kann die Reihenfolge der Datensätze anders sein, da Tabellen als Mengendarstellung keine Reihenfolge haben:

```
[EL Info]: 2019-08-09 11:22:34.97--ServerSession(1219273867)--EclipseLink, version: Eclipse Persistence
Services - 2.7.4.v20190115-ad5b7c6b2a
[EL Info]: connection: 2019-08-09 11:22:35.739--ServerSession(1219273867)--
/file:/F:/workspaces/eclipseWS/JDBCSpielerei/bin/_Spielerei2 login successful
4: Aische
1: Egon
2: Erwin
3: Ute
[EL Info]: connection: 2019-08-09 11:22:36.124--ServerSession(1219273867)--
/file:/F:/workspaces/eclipseWS/JDBCSpielerei/bin/_Spielerei2 logout successful
```

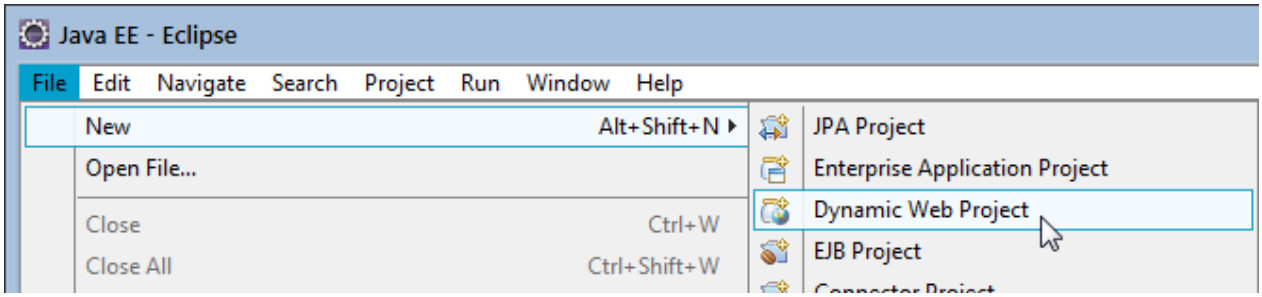
Sollte es eine Fehlermeldung der Form „Exception in thread "main" java.lang.SecurityException: class "javax.persistence.PersistenceUtil"'s signer information does not match signer information of other classes in the same package“ geben, ist ein EclipseLink Problem noch nicht behoben. Wird unter „Properties“ dann „Java Build Path > Order and Export“ angesehen, steht jakarta.persistence unter eclipselink, wie in der folgenden Abbildung gezeigt. Durch einen Klick auf „Up“ wird die Position mit eclipselink getauscht. Danach wird „Apply and Close“ geklickt und das Programm sollte laufen.



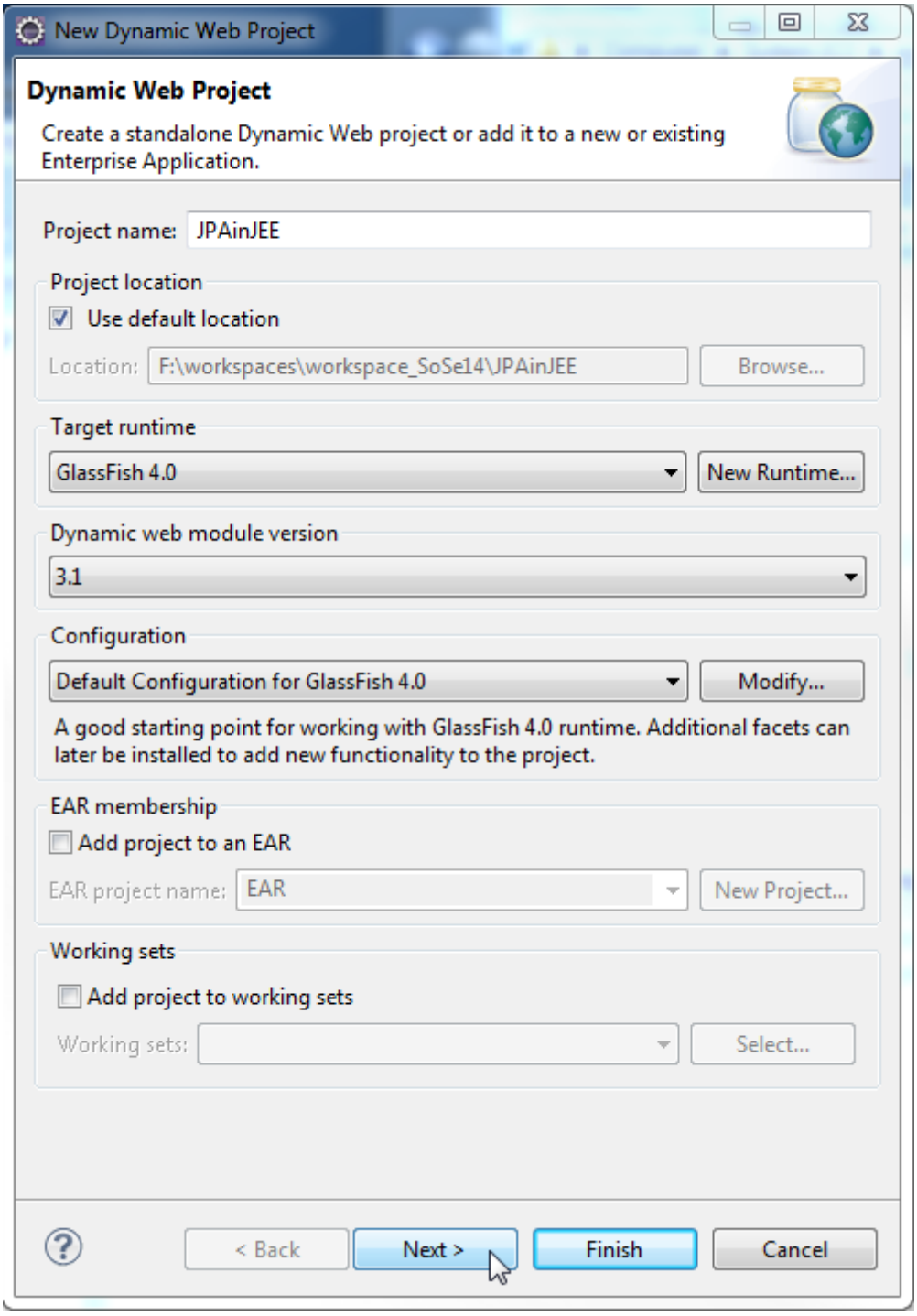
19.5 JPA-Nutzung mit Eclipse JEE (4.3.1)

In der Eclipse JEE-Version sind die zur Nutzung von JPA mit EclipseLink notwendigen Dateien bereits enthalten. Weiterhin wird einiger Entwicklungs-Support geliefert, von dem hier ein Teil vorgestellt werden soll.

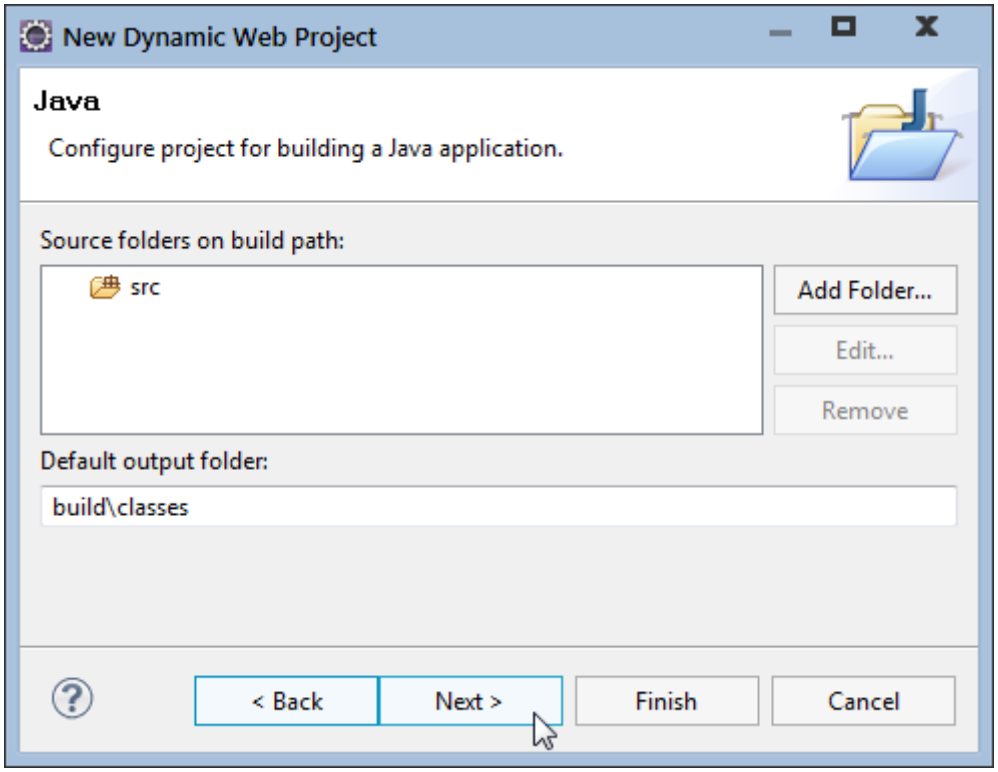
Zunächst wird ein neues Projekt angelegt. Hierbei kann bereits die Auswahl „JPA Project“ genutzt werden, was für Projekte genutzt wird, die typischerweise in andere Projekte eingebunden werden. Bei kleineren Projekten, die alleine eine Datenbankanbindung nutzen, ist dies nicht notwendig, so dass für das Beispiel ein „Dynamic Web Project“ über „File“ und „New“ angelegt wird.



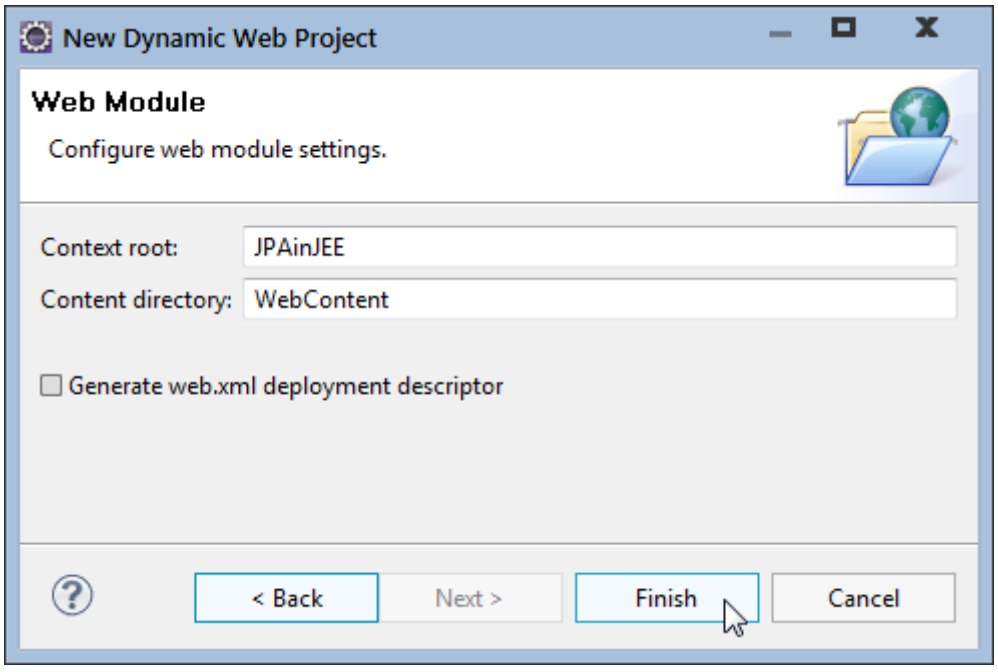
Es wird der Projektname eingegeben, die weiteren Einstellungen übernommen und „Next>“ gedrückt. Für ein einfaches Beispiel kann auch „Finish“ genutzt werden, da hier keine zusätzlichen Einstellungen passieren.



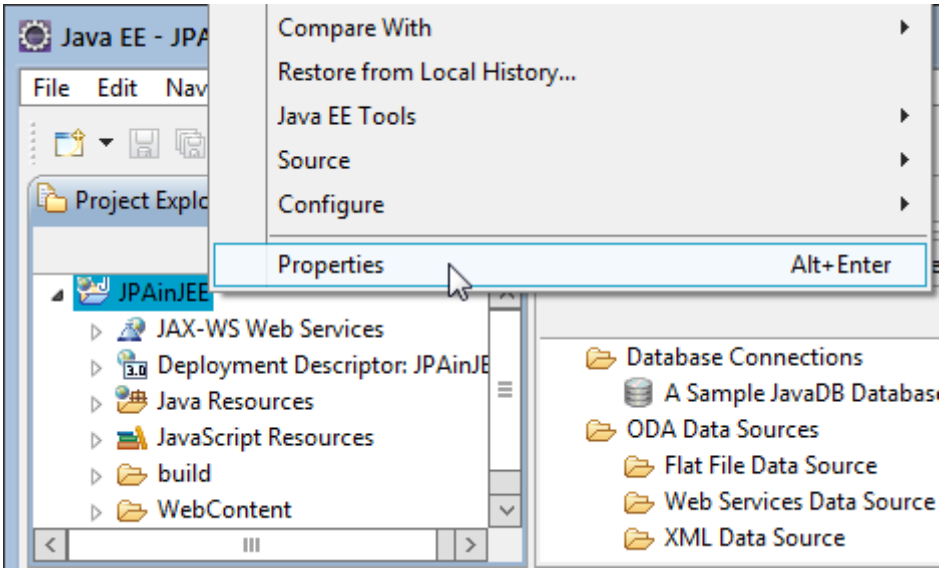
Im nächsten Fenster wird wieder „Next >“ gedrückt“.



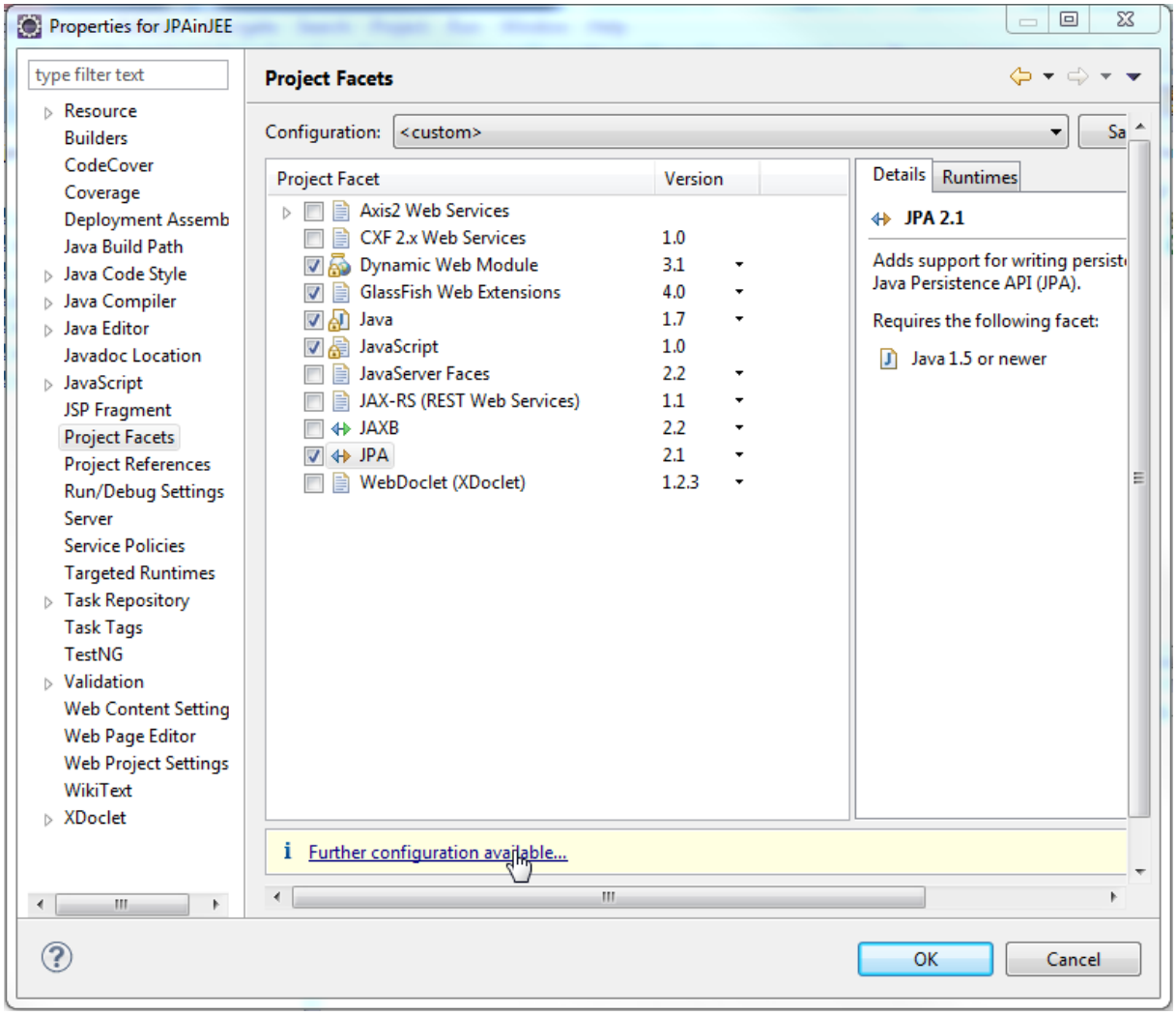
Die Projekterstellung wird mit „Finish“ abgeschlossen.



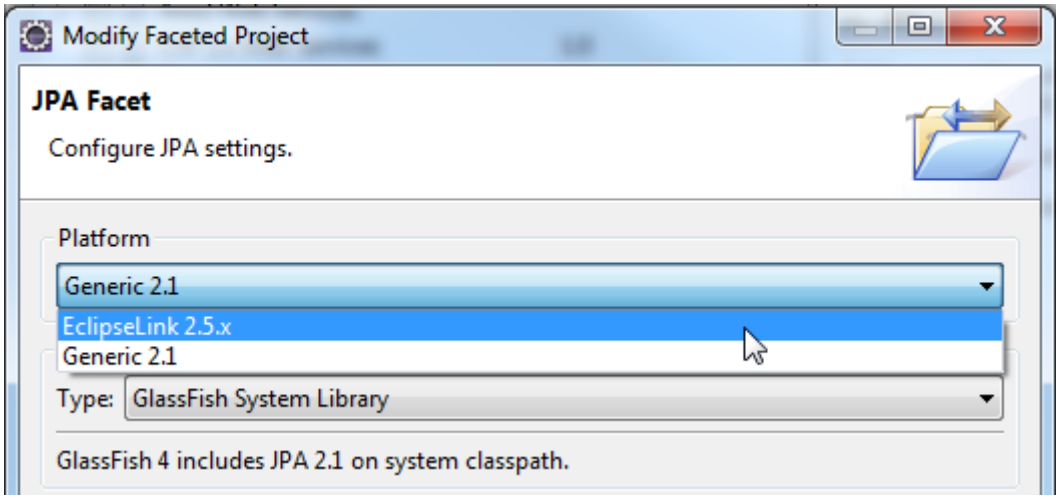
Im nächsten Schritt muss die JPA-Funktionalität in das Projekt eingebunden werden. Dazu wird ein Rechtsklick auf dem Projekt gemacht und „Properties“ gewählt.



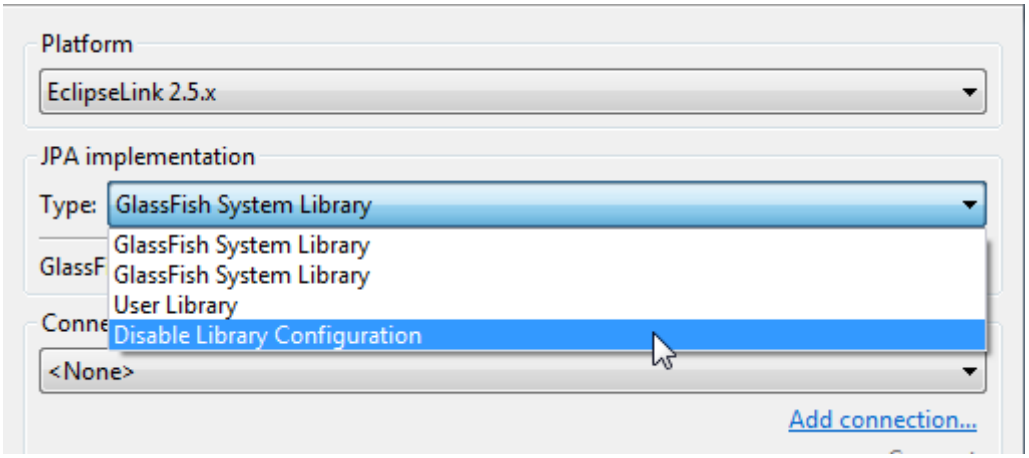
Auf der linken Seite wird „Project Facet“ gewählt, rechts zusätzlich ein Haken bei „JPA“ gesetzt und dann unten auf „Further configuration available…“ geklickt.



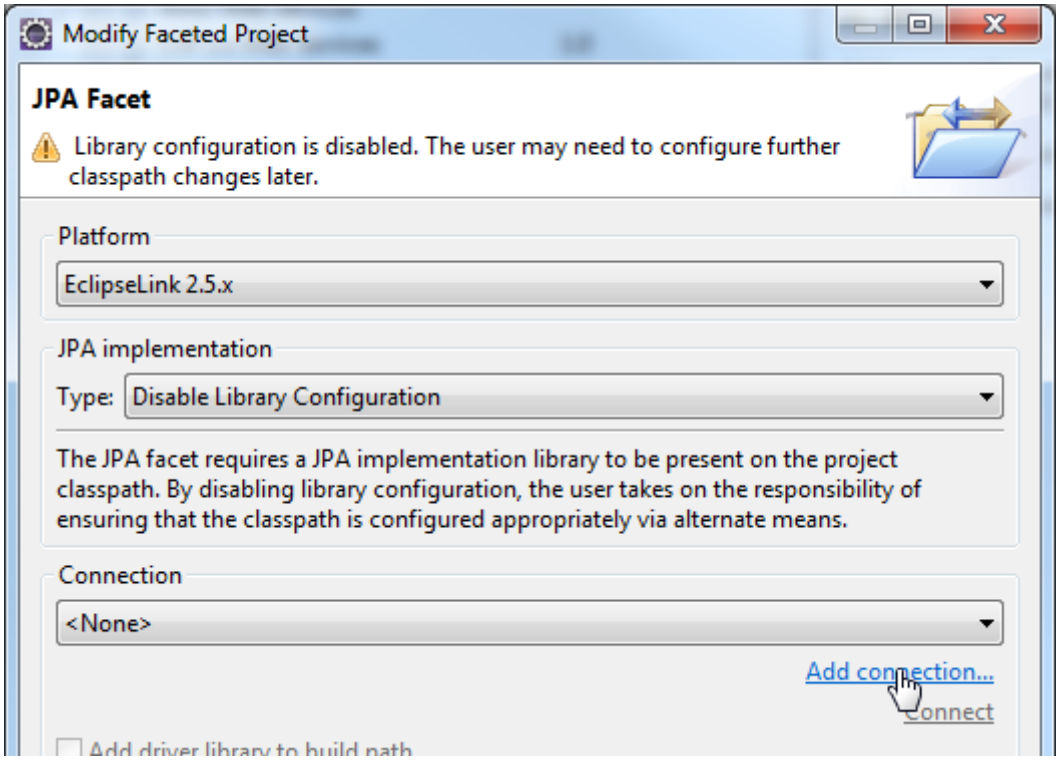
Zunächst wird oben als Platform „EclipseLink 2.5.x“ ausgewählt.



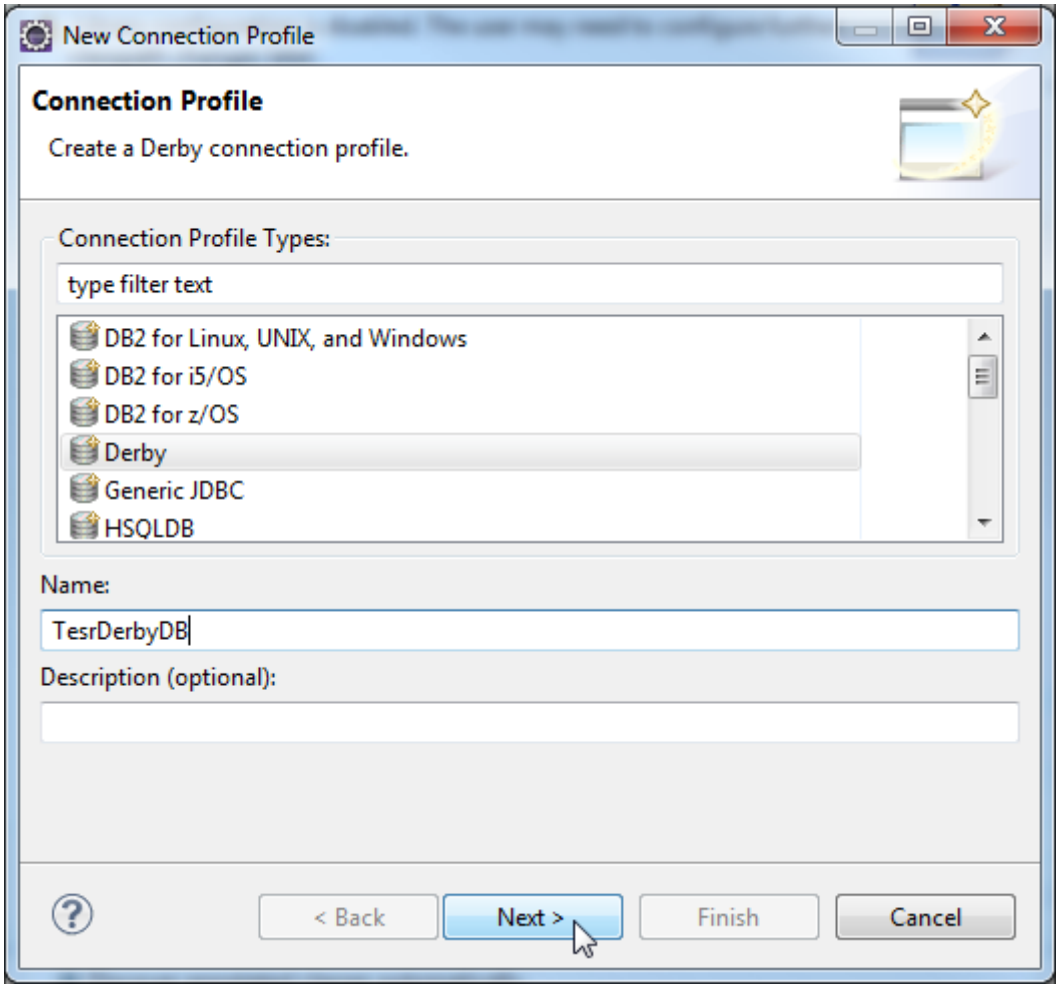
Bei der JPA Implemention wird „Disable“ gewählt, da der benutzte GlassFish-Server die Implementierung bereits zur Verfügung stellt. Ansonsten müssen hier die zur Implementierung gehörenden Dateien angegeben werden.



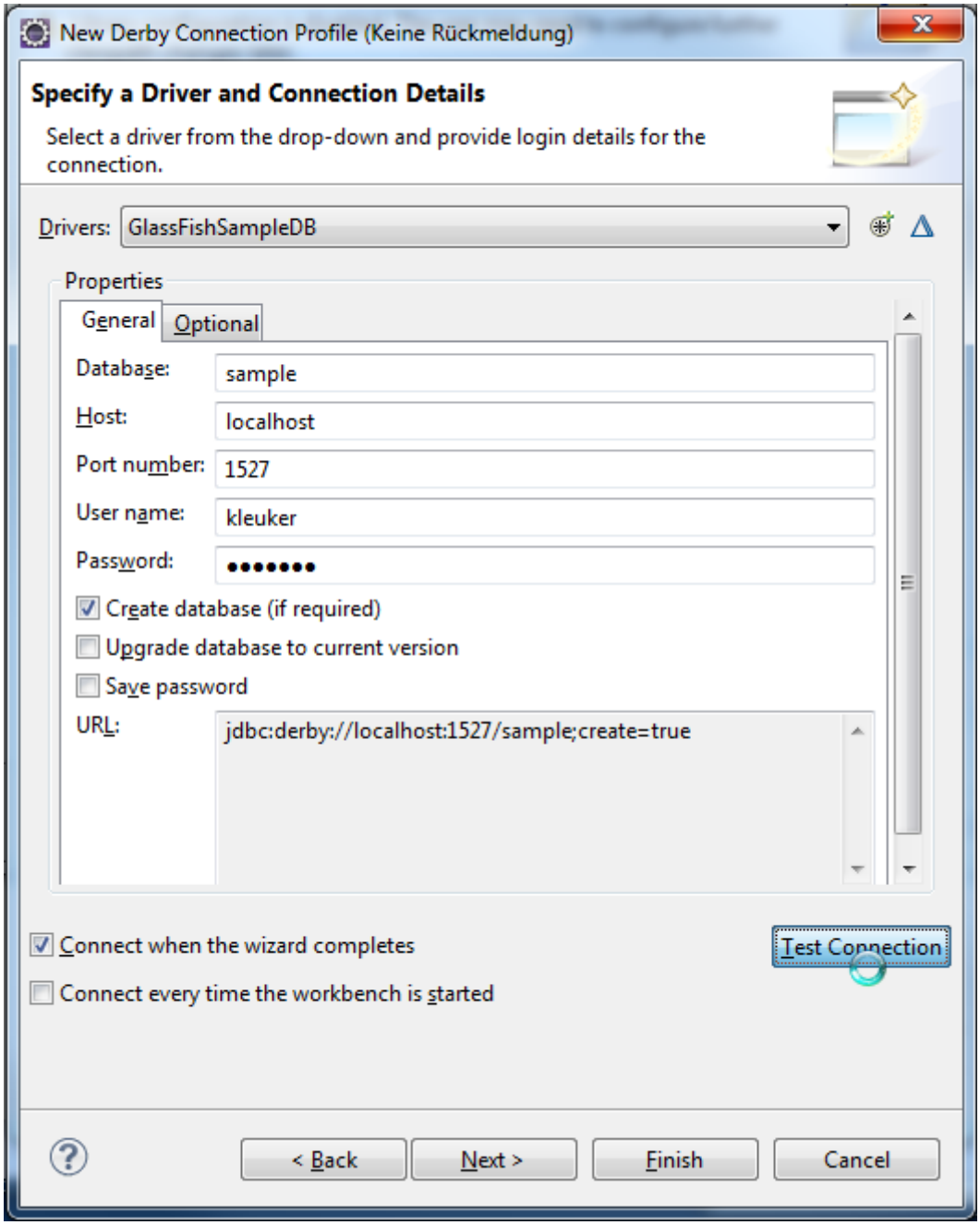
Bei Connection ist die zu nutzende Datenbankverbindung anzugeben. Dies kann auch später in der entstehenden persistence.xml-Datei geändert werden, soll hier aber mit Hilfe einer neuen Datenbank geschehen. Dazu wird der Link „Add connection…“ angeklickt.

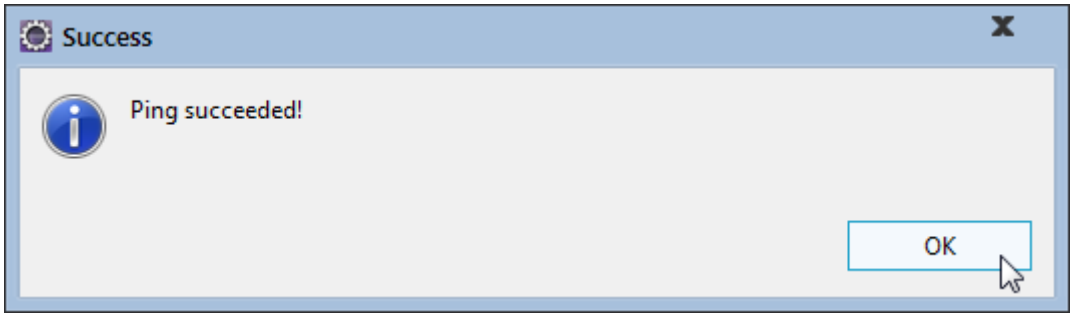


Hierzu wird als Connection Profile zunächst „Derby“ ausgewählt, weiterhin kann unten dem Profil ein Name gegeben werden und es wird „Next>“ gedrückt.

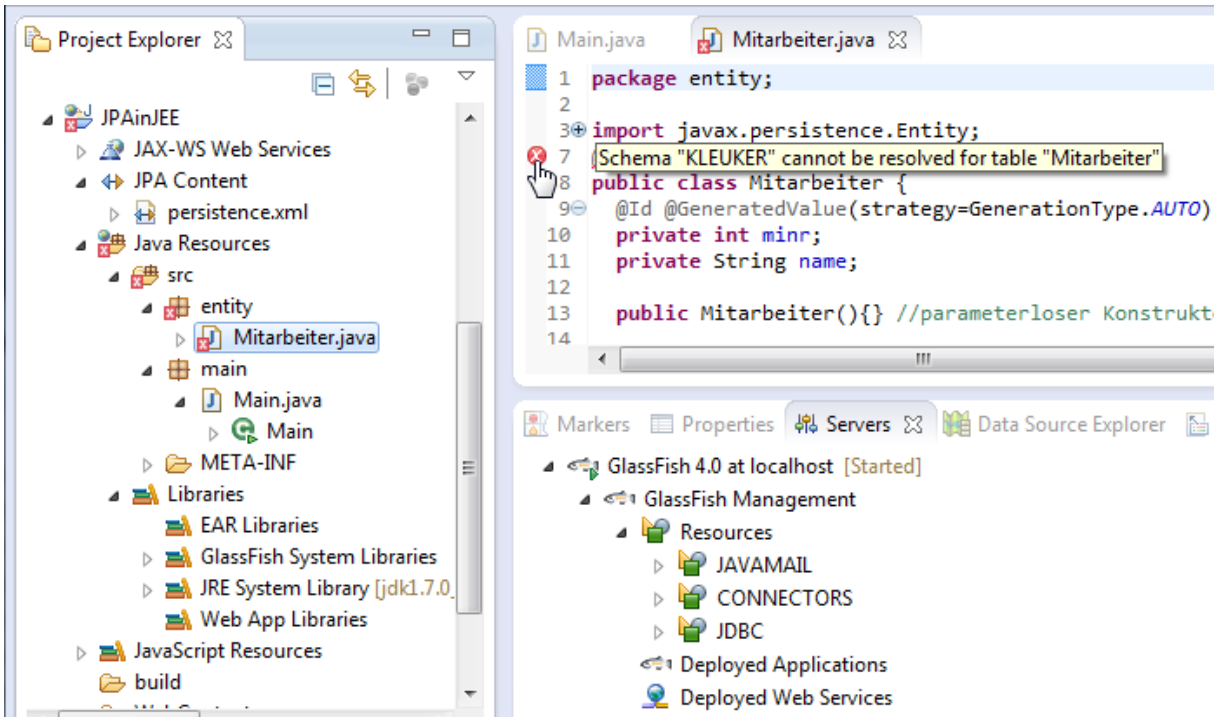


Jetzt kann der Datenbank ein Name gegeben werden, weiterhin ist die IP-Adresse anzugeben. Generell können die Einstellungen so bleiben, abschließend ist mit „Test Connection“ die Verbindung zu prüfen und die Einrichtung mit „Finish“, „OK“ und „OK“ abzuschließen.


New Derby Connection Profile (Keine Rückmeldung)
Specify a Driver and Connection Details
Select a driver from the drop-down and provide login details for the connection.
Drivers: GlassFishSampleDB
Properties
General
Optional
Database: sample
Host: localhost
Port number: 1527
User name: kleuker
Password: •••••••
Create database (if required)
Upgrade database to current version
Save password
URL: jdbc:derby://localhost:1527/sample;create=true
Connect when the wizard completes
Test Connection
Connect every time the workbench is started
< Back
Next >
Finish
Cancel


Success
Ping succeeded!
OK

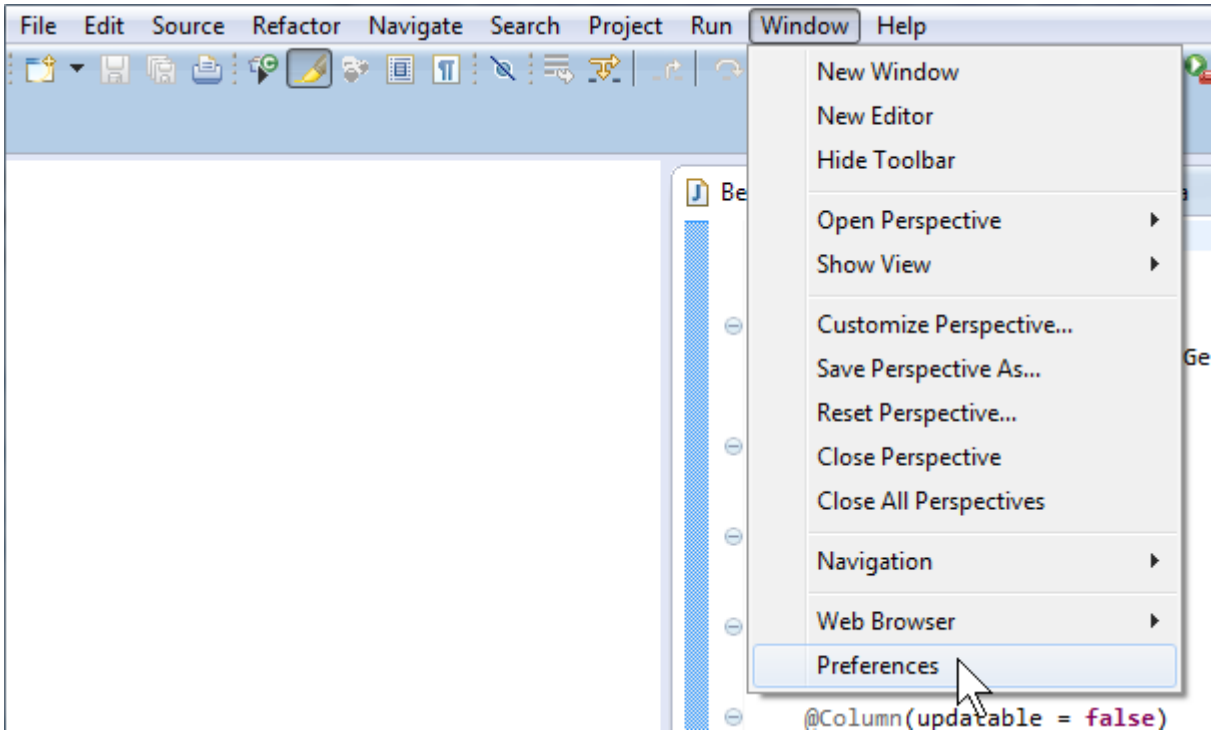
Im nächsten Schritt werden die Klassen Mitarbeiter und Main aus dem vorherigen Abschnitt hier angelegt, wobei der Name der Persistence Unit auf „JPAinJEE“ statt „Spielerei2“ in Main angepasst wird. Es fällt auf, dass die Klasse Mitarbeiter eine Fehlermeldung enthält, die hier ärgerlich, aber nicht relevant ist, da hier eine lokale Datenbank und nicht der Server mit seiner Datenbank genutzt wird, wie es für eine JSF-Nutzung typisch wäre. Das Projekt hat folgendes Aussehen.



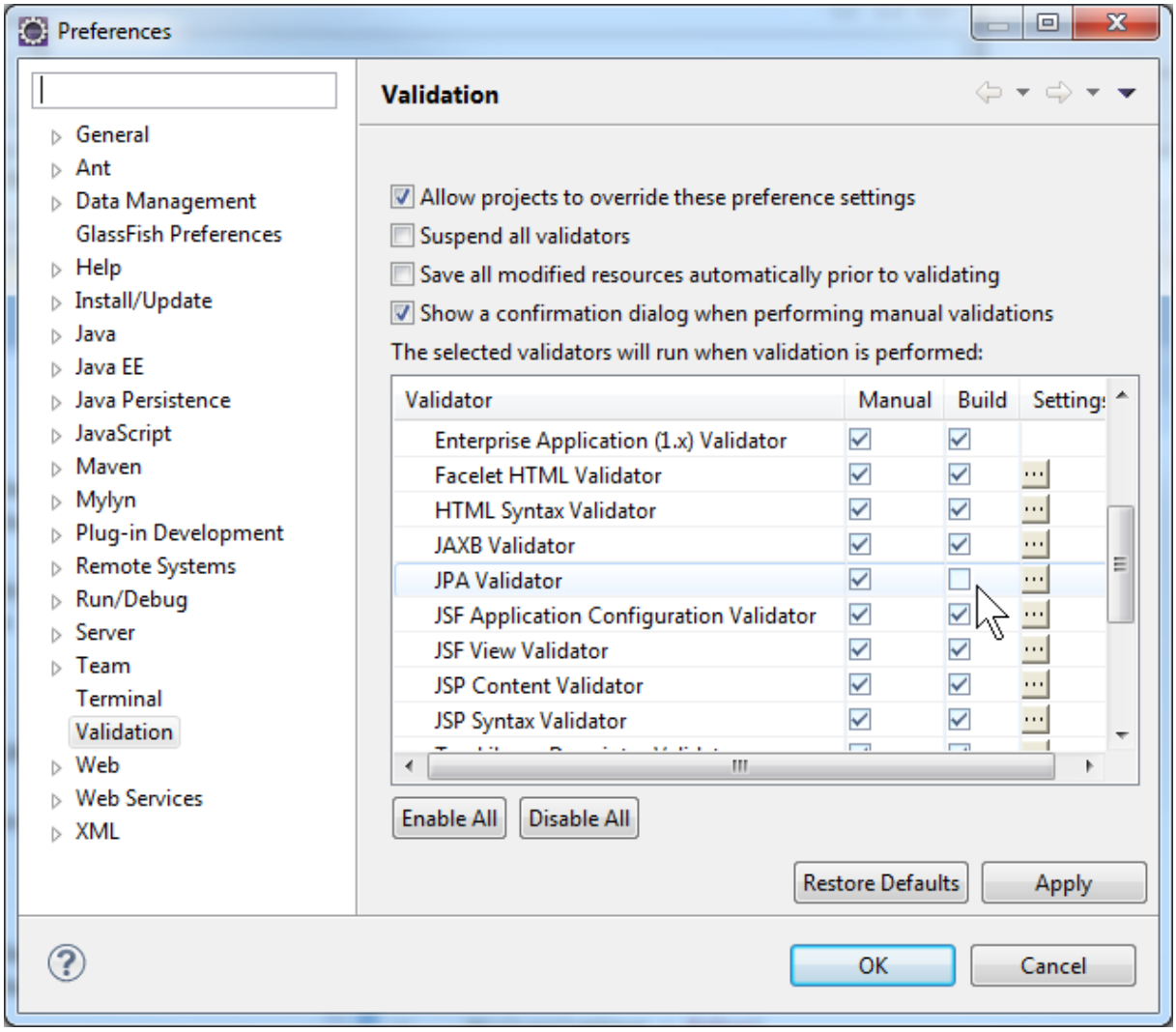
Sollte diese Fehlermeldung, die sonst keine Auswirkung hat, irritieren, kann man dies abändern.

Der folgende Weg funktioniert (wohl) immer, greift aber in die Funktionalität ein und könnte zu verspäteten Fehlern führen, der Eingriff ist aber recht marginal. Danach folgt ein zweiter Weg, der anscheinend nicht immer funktioniert. Dieses Thema wird auch in Foren oft diskutiert, man suche nach „Eclipse JPA table cannot be resolved“.

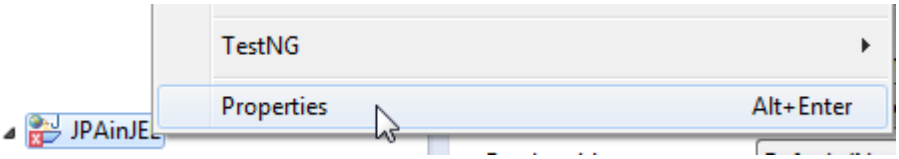
Für den ersten Weg wird oben in Eclipse „Window > Preferences“ ausgewählt.



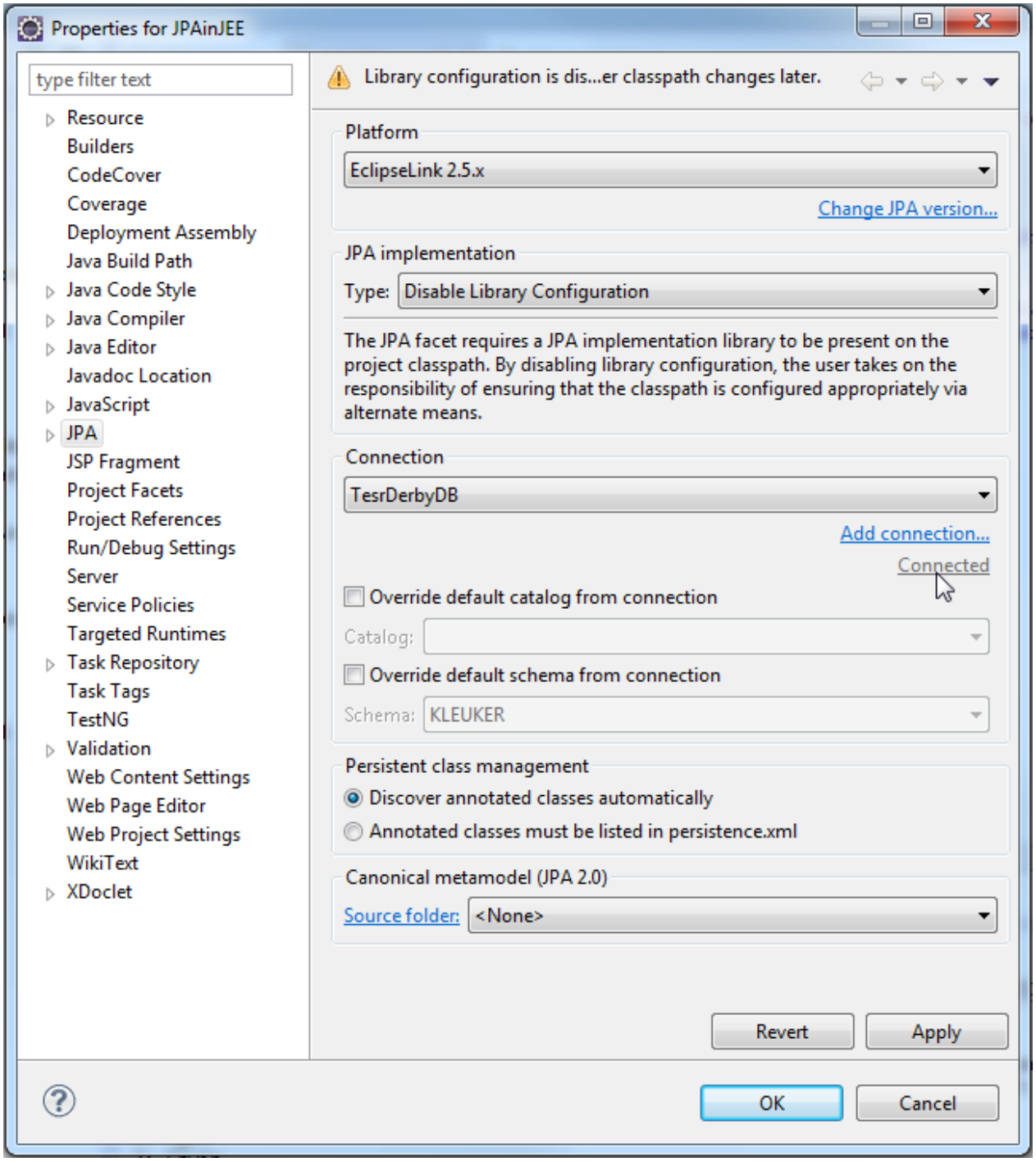
Auf der linken Seite wird „Validation“ ausgewählt, rechts in der Validator-Tabelle zu JPA-Validation gescrollt und der Haken in der Spalte Build weggenommen. Die Aktion wird mit „Ok“ abgeschlossen.



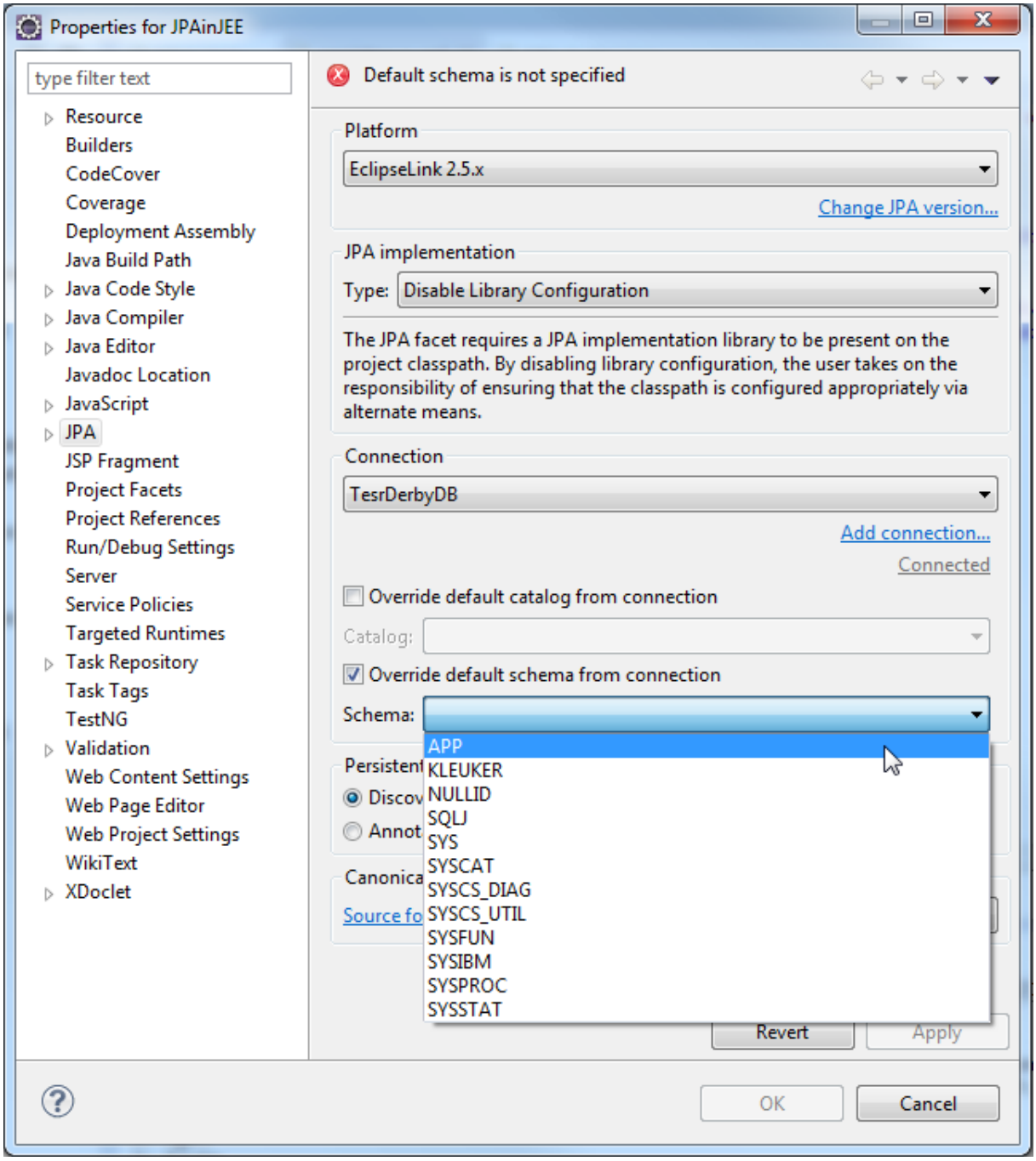
Der zweite Weg geht wie folgt (der Haupttext setzt mit „Genau wie im vorherigen Abschnitt“) fort. Es wird wieder für das Projekt „Properties“ ausgewählt.



Links wird „JPA“ ausgewählt, dann recht in der Mitte auf „Connect…“ geklickt, das sich dann, wie in der folgenden Abbildung gezeigt, in „Connected“ verwandelt.

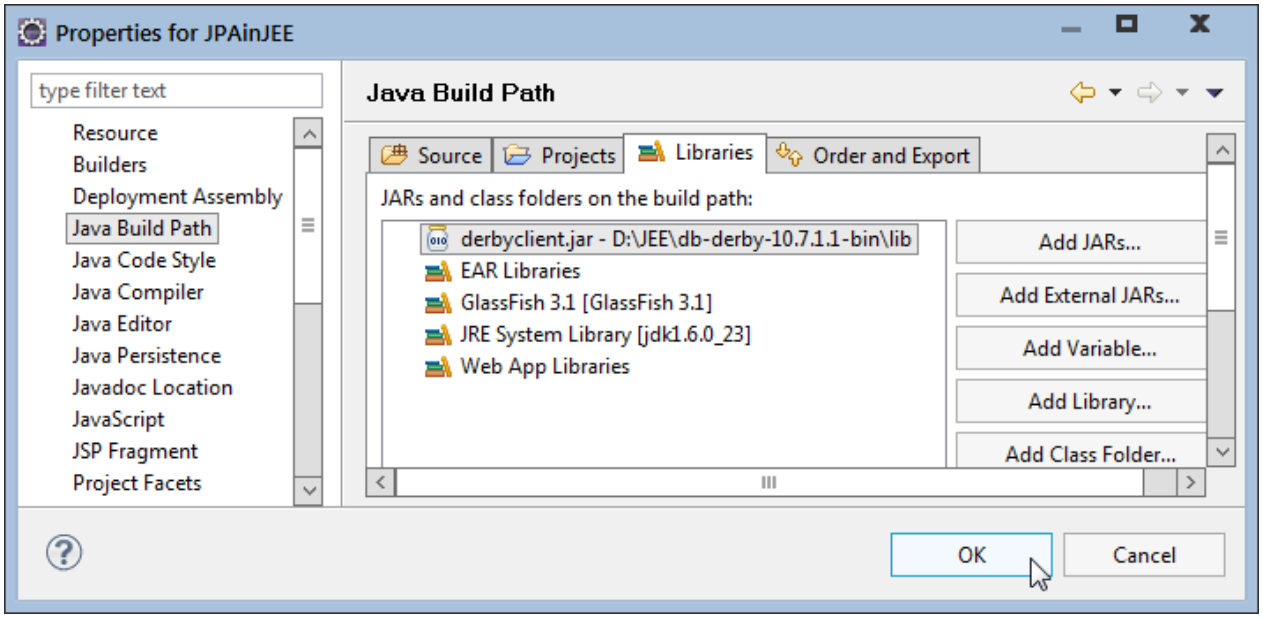


Nun wird zunächst ein Haken bei „Override default schema from connection“ gesetzt und dann das Schema vom Nutzer, hier „KLEUKER“, auf „APP“ geändert, zunächst „Apply“ geklickt und dann mit „OK“ bestätigt.

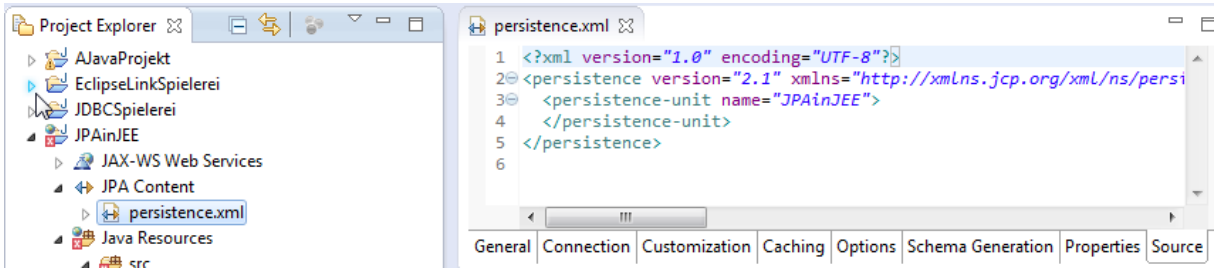


Die Fehlermeldung ändert sich, da die Datenbank nicht unter Kontrolle eines laufenden JEE-Servers steht.

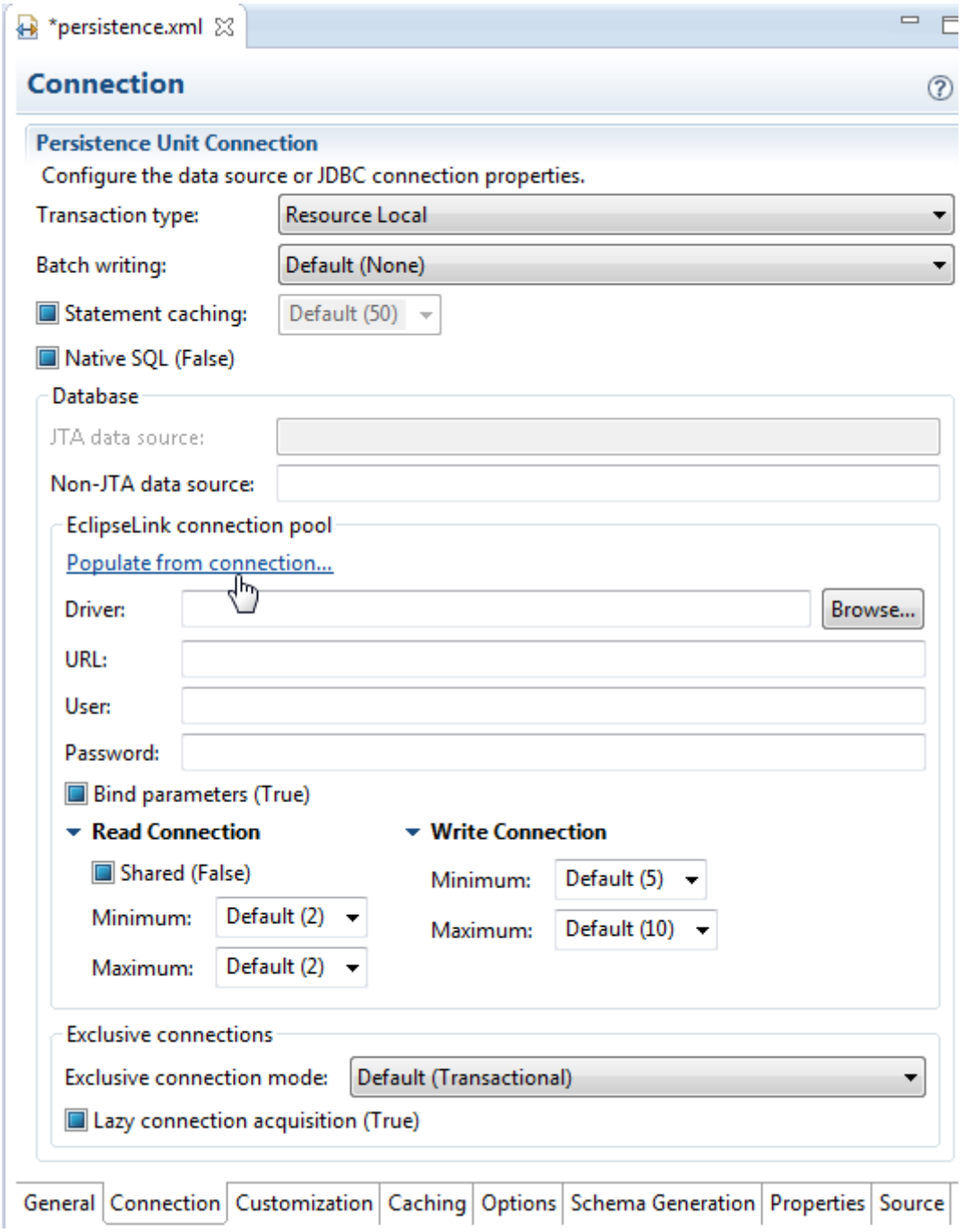
Genau wie in vorherigen Abschnitt, muss der JDBC-Treiber noch ergänzt werden (alternativ über Konfigurationseinstellungen als Default-System).



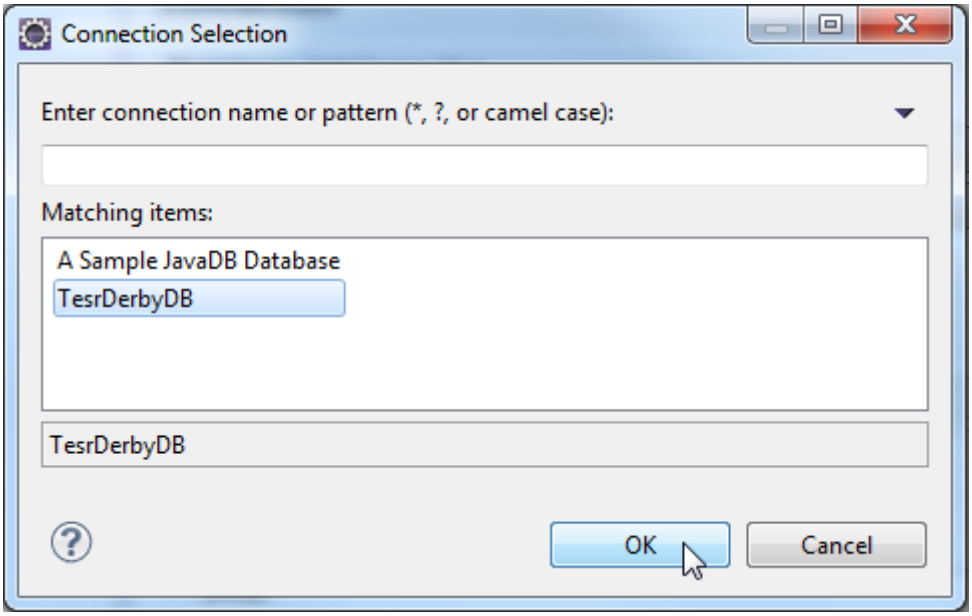
Im Projekt liegt im Ordner „JPA Content“ eine persistence.xml-Datei, die durch einen Doppelklick geöffnet werden kann. Man erhält einen strukturierten Editor, bei dem durch die Auswahl der unteren Reiter weitere Konfigurationsinformationen angegeben werden können. Diese Einstellungen hängen davon ab, ob eine Datenbank direkt im Server oder eine außerhalb des Servers vorhandene Datenbank genutzt werden soll. Hier wird der Fall der externen Datenbank betrachtet und zunächst der Reiter „Connection“ gewählt.



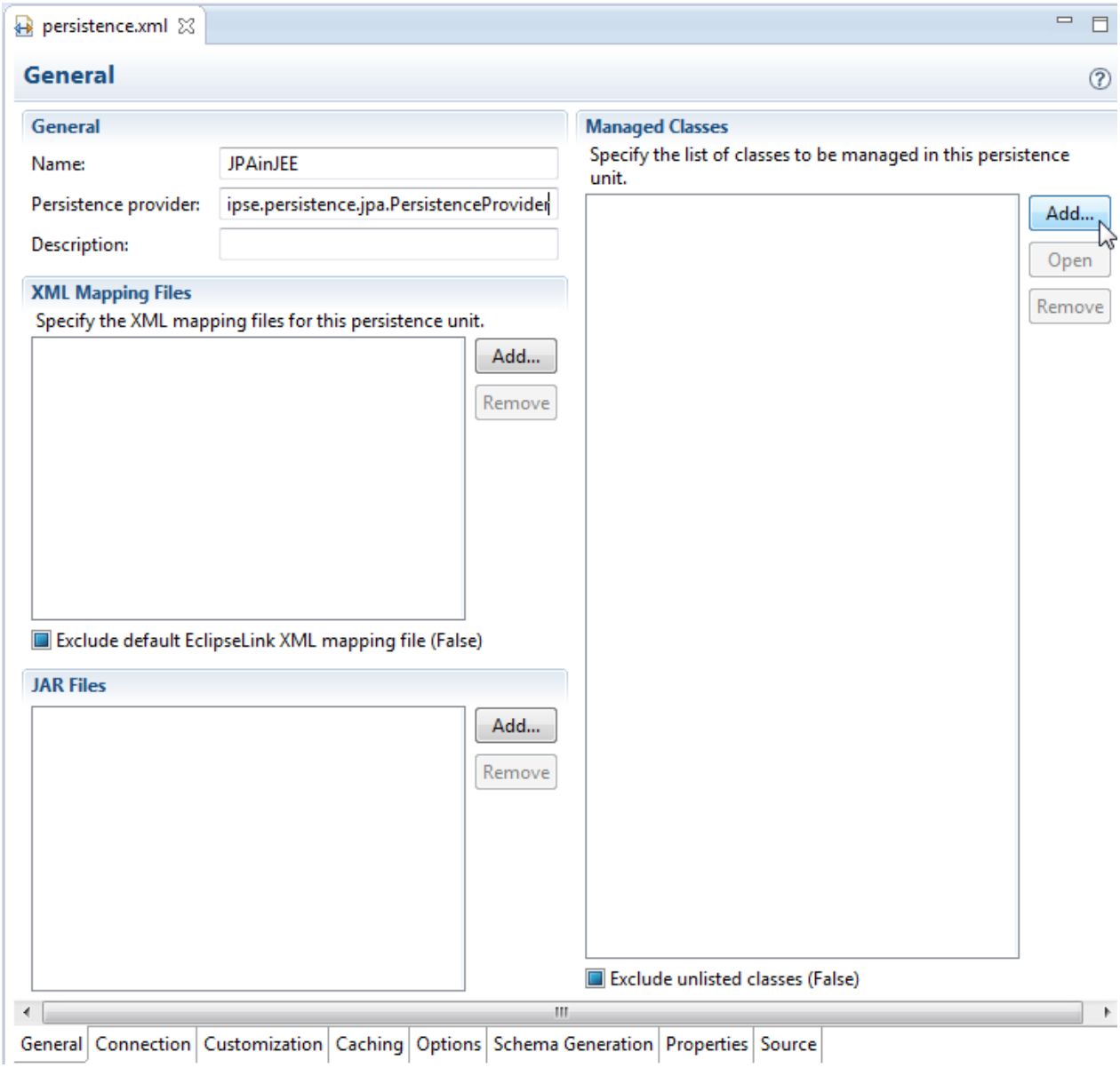
Im Reiter Connection wird zunächst oben der Transaction Type auf „Resource Local“ gestellt und die JDBC-Verbindung konfiguriert, die im Beispiel über den Link „Populate from Connection…“ zugängig ist. Generell kann man sich überlegen, ob diese Datei nicht wesentlich Schneller durch Copy & Paste und leichte Änderungen entsteht.



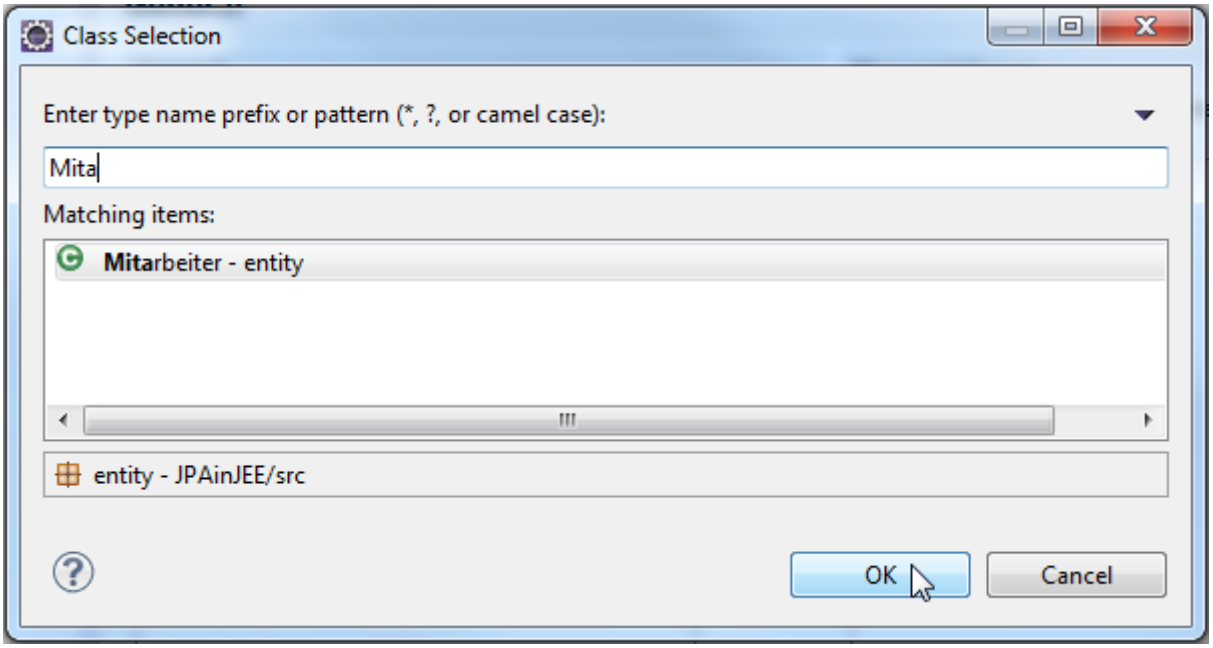
Hier kann die vorher eingerichtete Verbindung TestDerbyDB gewählt werden, so dass man weitere Verbindungseinträge nach „OK“ erhält.



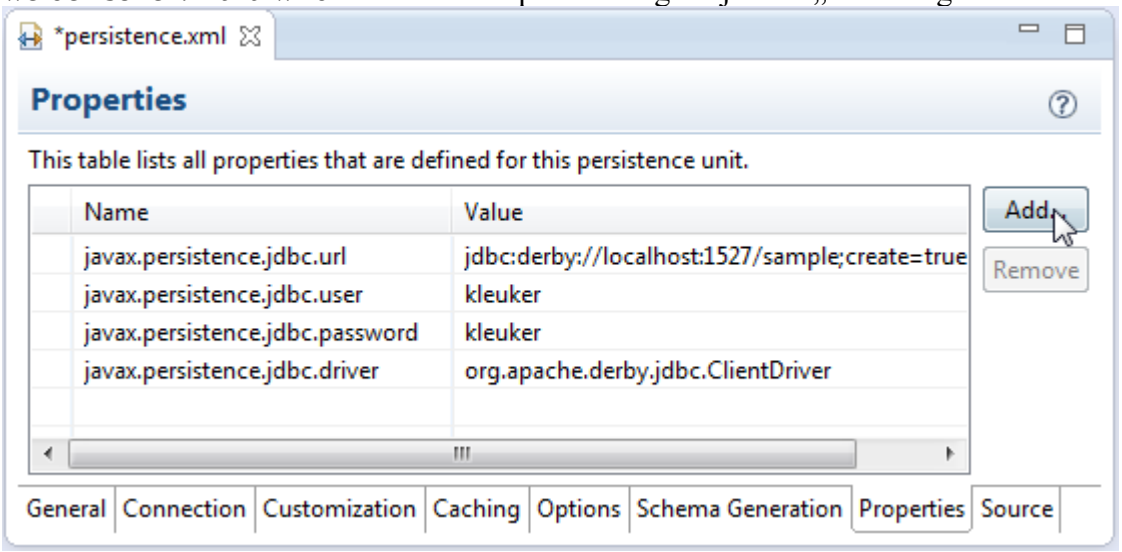
Danach wird der Reiter „General“ gewählt und der Eintrag unter Persistence Provider auf „org.eclipse.persistence.jpa.PersistenceProvider“ gesetzt. Im nächsten Schritt werden die zu persistierenden Klassen mir „Add…“ bei den Managed CLasses ausgewählt.



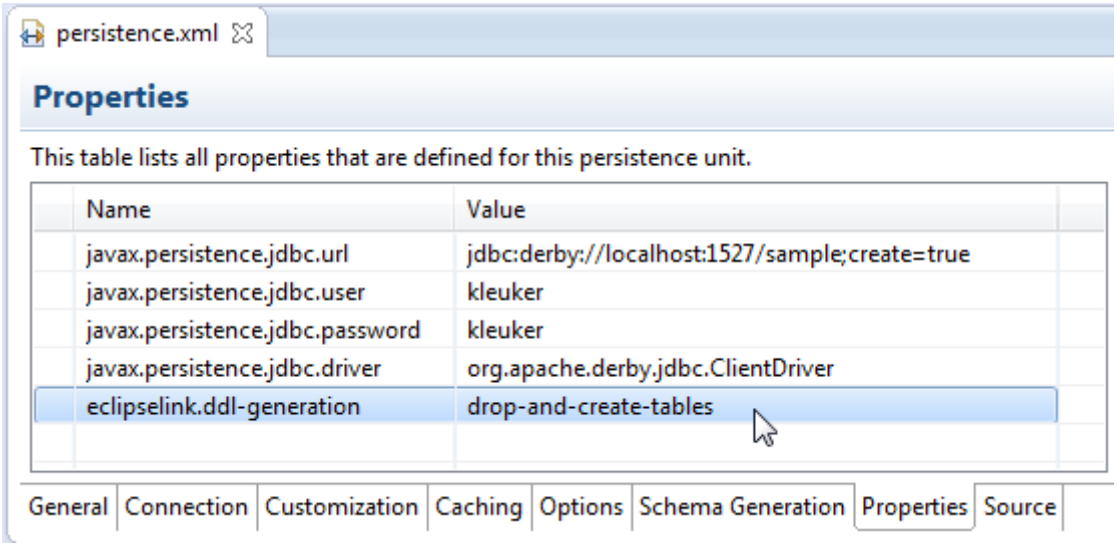
Hier wird der Klassenname oben eingegeben, die gewünschte Klasse gefunden und mir „OK“ bestätigt.



Abschließend wird noch festgelegt, dass die Tabellen immer wieder neu beim Start angelegt werden sollen. Dazu wird der Reiter Properties ausgewäjlt und „Add…“ geklickt.



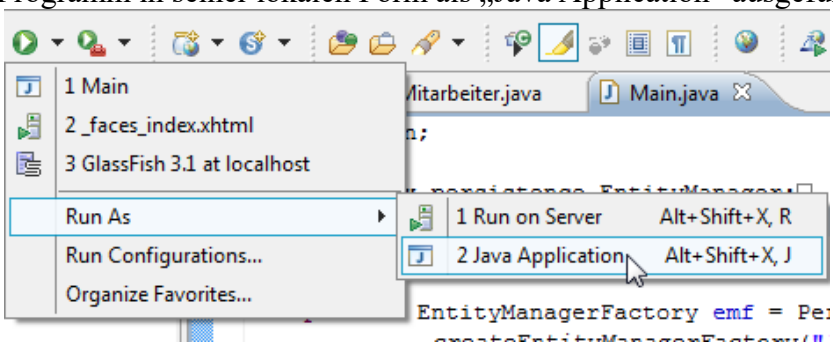
Es wird ein Eintrag mit dem Namen eclipselink.ddl-generation und dem Wert "drop-and-create-tables ergänzt.



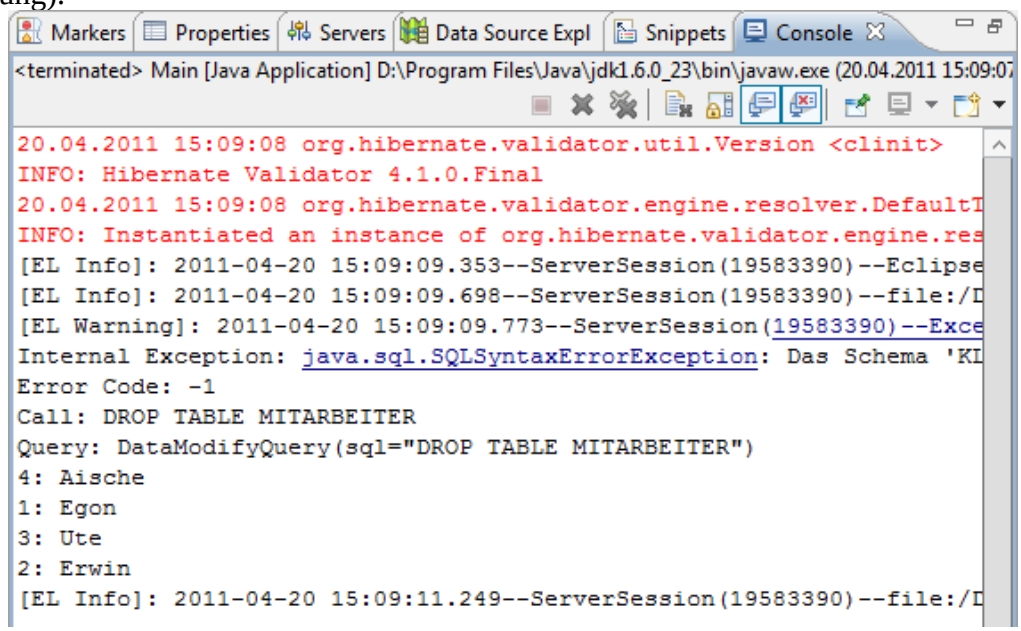
Insgesamt sieht die resultierende Datei wie folgt aus.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
        <persistence-unit name="JPAinJEE" transaction-type="RESOURCE_LOCAL">
                <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
                <class>entity.Mitarbeiter</class>
                <properties>
                        <property name="javax.persistence.jdbc.url"
                value="jdbc:derby://localhost:1527/sample;create=true"/>
                        <property name="javax.persistence.jdbc.user" value="kleuker"/>
                        <property name="javax.persistence.jdbc.password" value="kleuker"/>
                        <property name="javax.persistence.jdbc.driver"
                value="org.apache.derby.jdbc.ClientDriver"/>
                        <property name="eclipselink.ddl-generation"
                value="drop-and-create-tables"/>
                </properties>
        </persistence-unit>
</persistence>
```

Nun kann das Programm in seiner lokalen Form als „Java Application“ ausgeführt werden.



Man erhält folgendes Ergebnis (Warnings und Infos sind zu lesen, haben hier aber keine Bedeutung).

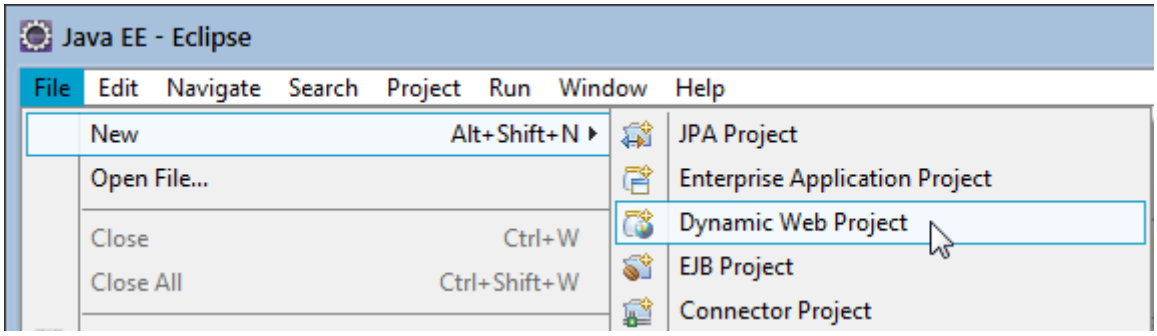


20 JSF-Applikationen in Eclipse JEE (4.3.1)

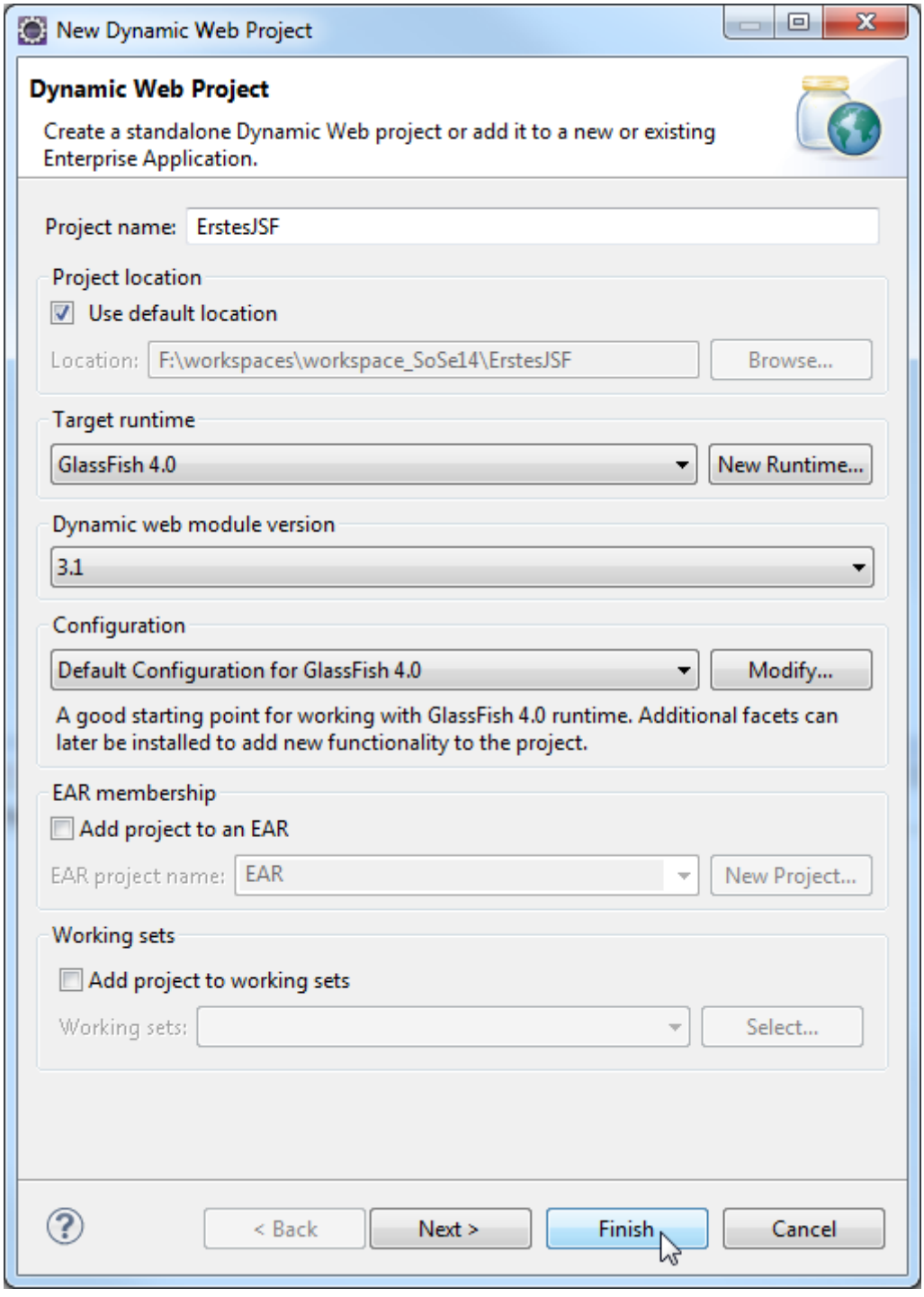
Dieses Kapitel gibt einen Überblick über die Entwicklung von JSF-Applikationen in der Eclipse JEE-Umgebung. Dabei wird nur der grundsätzliche Prozess beschrieben und keine Varianten diskutiert.

20.1 Einrichten eines JSF-Projekts

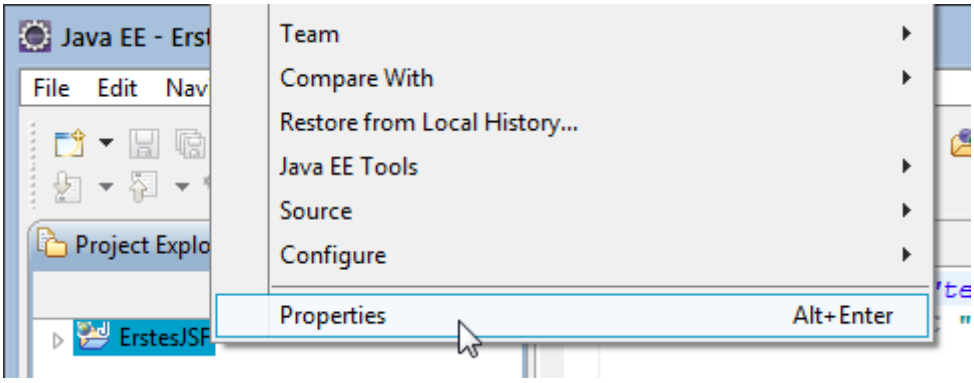
Zunächst wird über „File“ und „New“ ein „Dynamic Web Project“ angelegt.



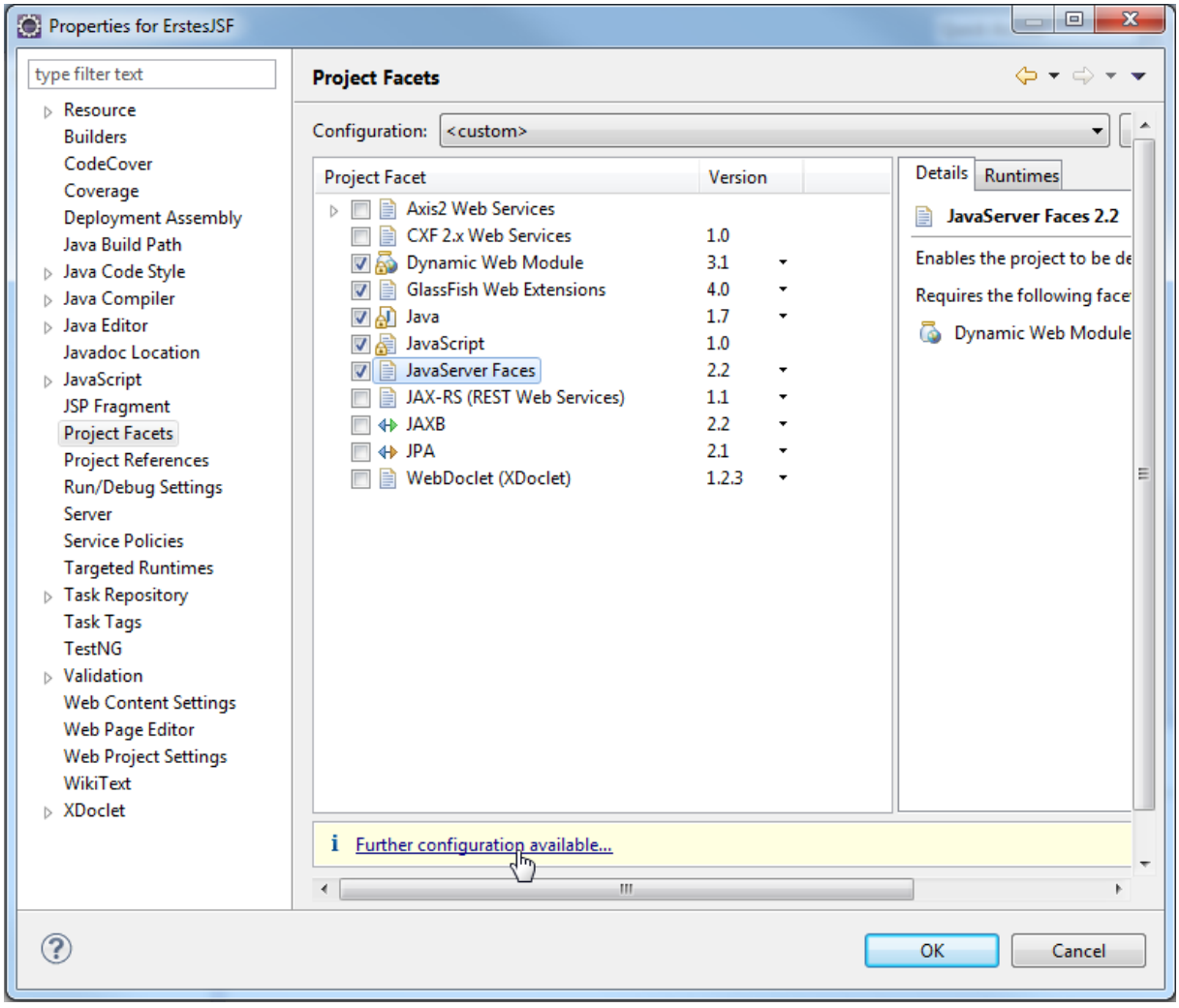
Nun wird dem Projekt ein Name gegeben und „Finish“ gedrückt, da weitere Einstellungen auch später bearbeitet werden können. Es wird davon ausgegangen dass nur ein GlassFish 4-Server für Eclipse konfiguriert ist, so dass alle Einträge übernommen werden können.



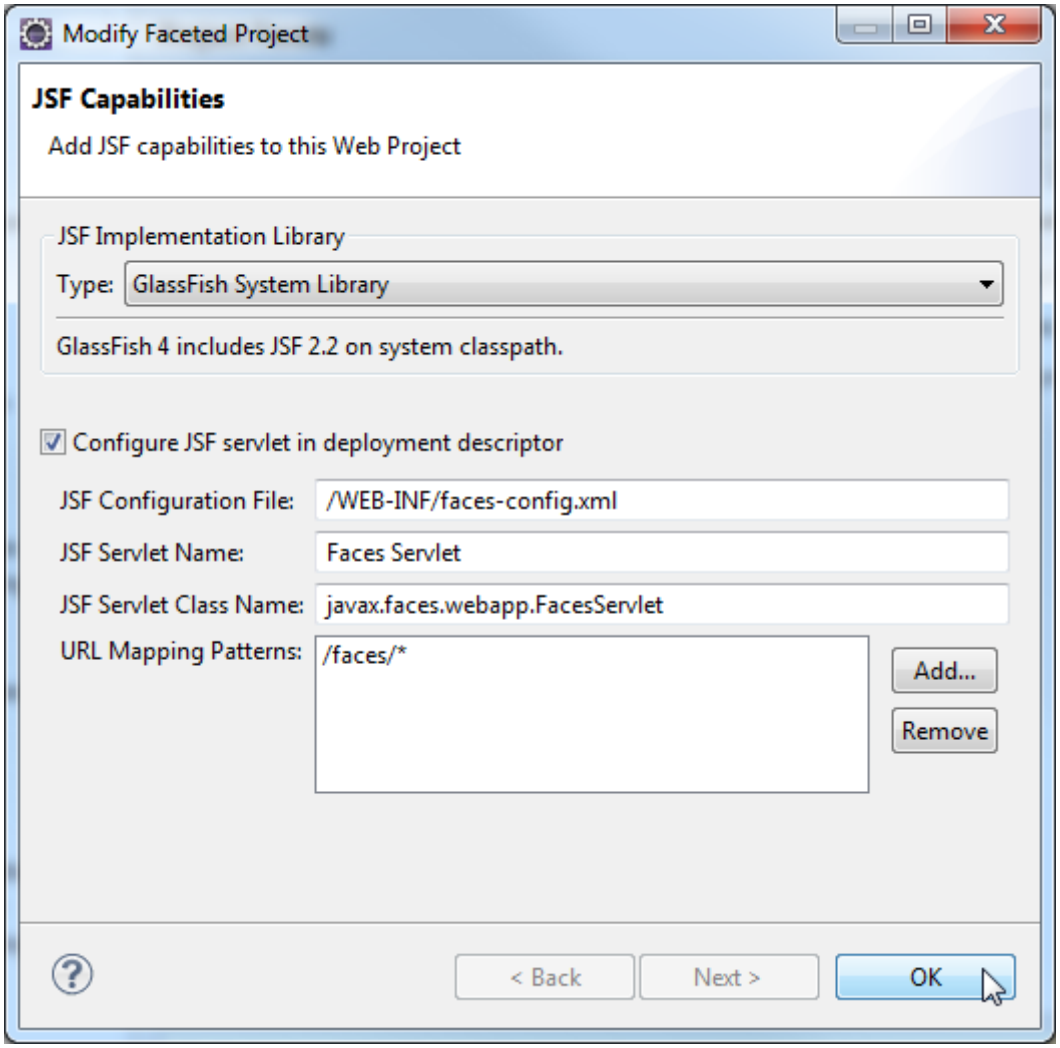
Nach einem Rechtsklick auf dem Projektnamen wird „Properties“ gewählt.



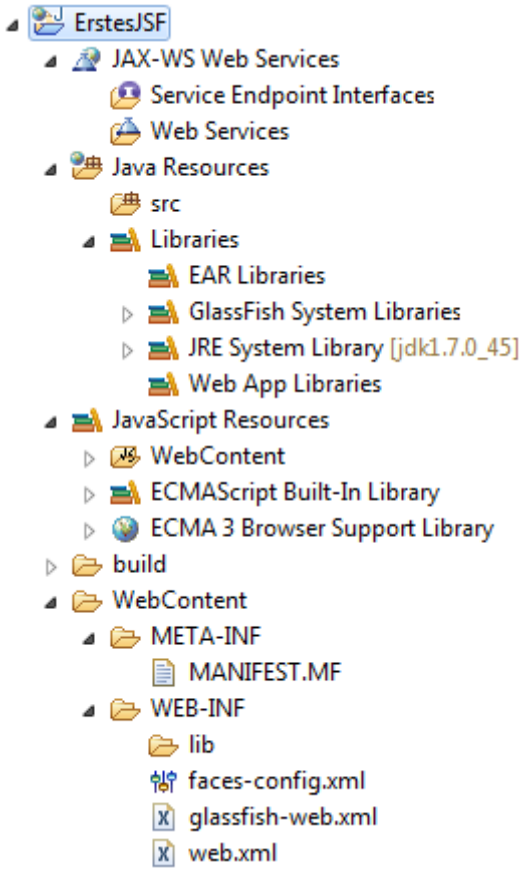
Hier wird auf der linken Seite „Project Facets“ ausgewählt und rechts ein Haken bei JSF ergänzt. Danach wird der jetzt erscheinende Link „Further configuration available…“ zur Information gedrückt.



Die Einträge können so übernommen werden, Experten könnten Änderungen z. B. bei den „URL Mapping Patterns“ vornehmen.



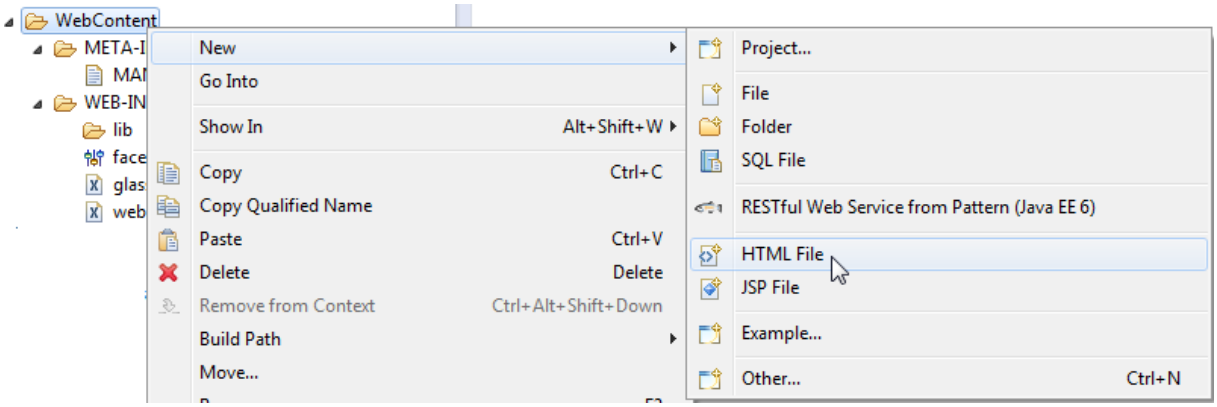
Man erhält ein Projekt mit folgender Struktur, wobei noch keine Web-Seite angelegt ist, die später unter WebContent stehen. Man sieht, dass drei Konfigurationsdateien, wen.xml generell für die Web-Applikationen, faces-config.xml für JSF-spezifische Einstellungen und glassfish-web.xml für serverspezifische Einstellungen erzeugt wurden.



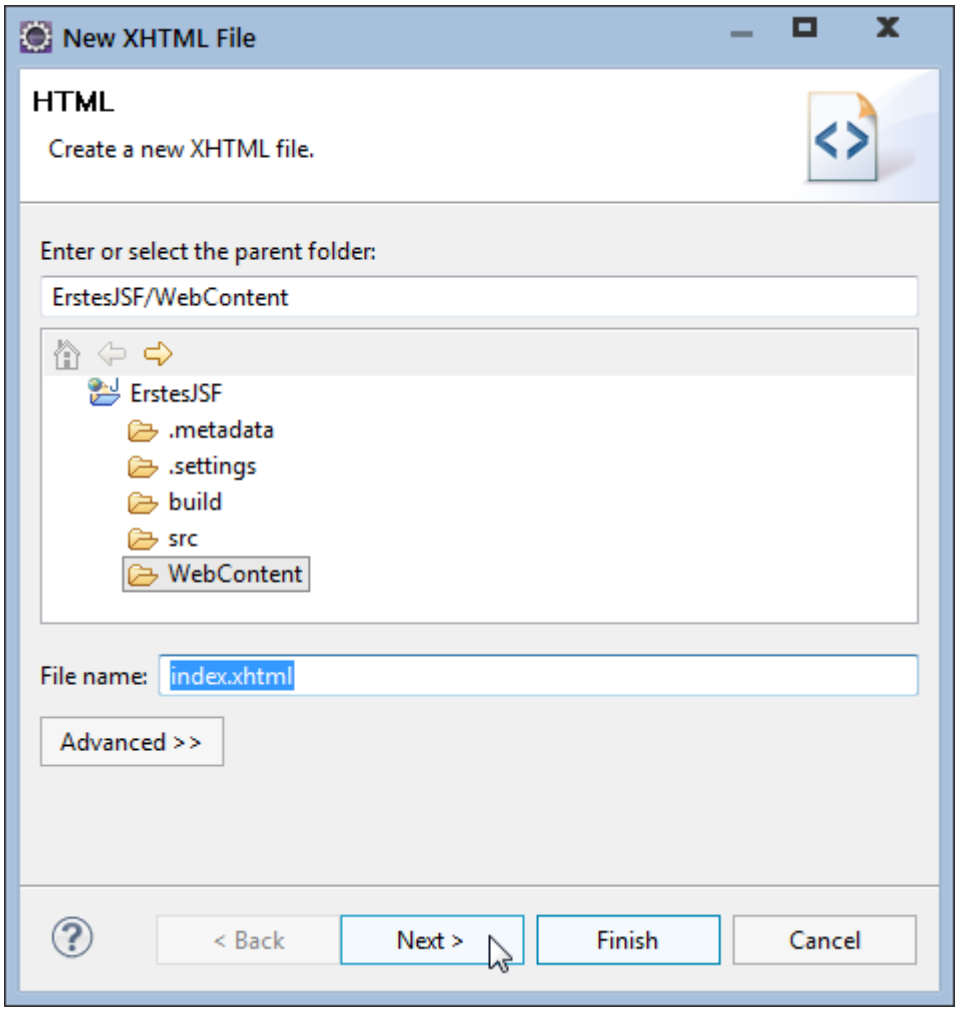
20.2 Erstellung einer einfachen JSF-Applikation

In diesem Abschnitt wird gezeigt, wie man eine einfache JSF-Applikation schrittweise entwickeln kann. Dabei wird auf die aktuell in JSF 2 üblichen CDI-Annotationen wie @Named zugegriffen. Als Ausgangspunkt wurde ein JSF-Projekt, wie im vorherigen Abschnitt beschrieben, angelegt. Es soll eine Applikation entstehen, mit der der Name eines Moduls und eine zugehörige Nummer eingegeben und auf einer Folgeseite wieder ausgegeben werden. Von dieser Seite aus, soll man wieder zum Anfang gelangen.

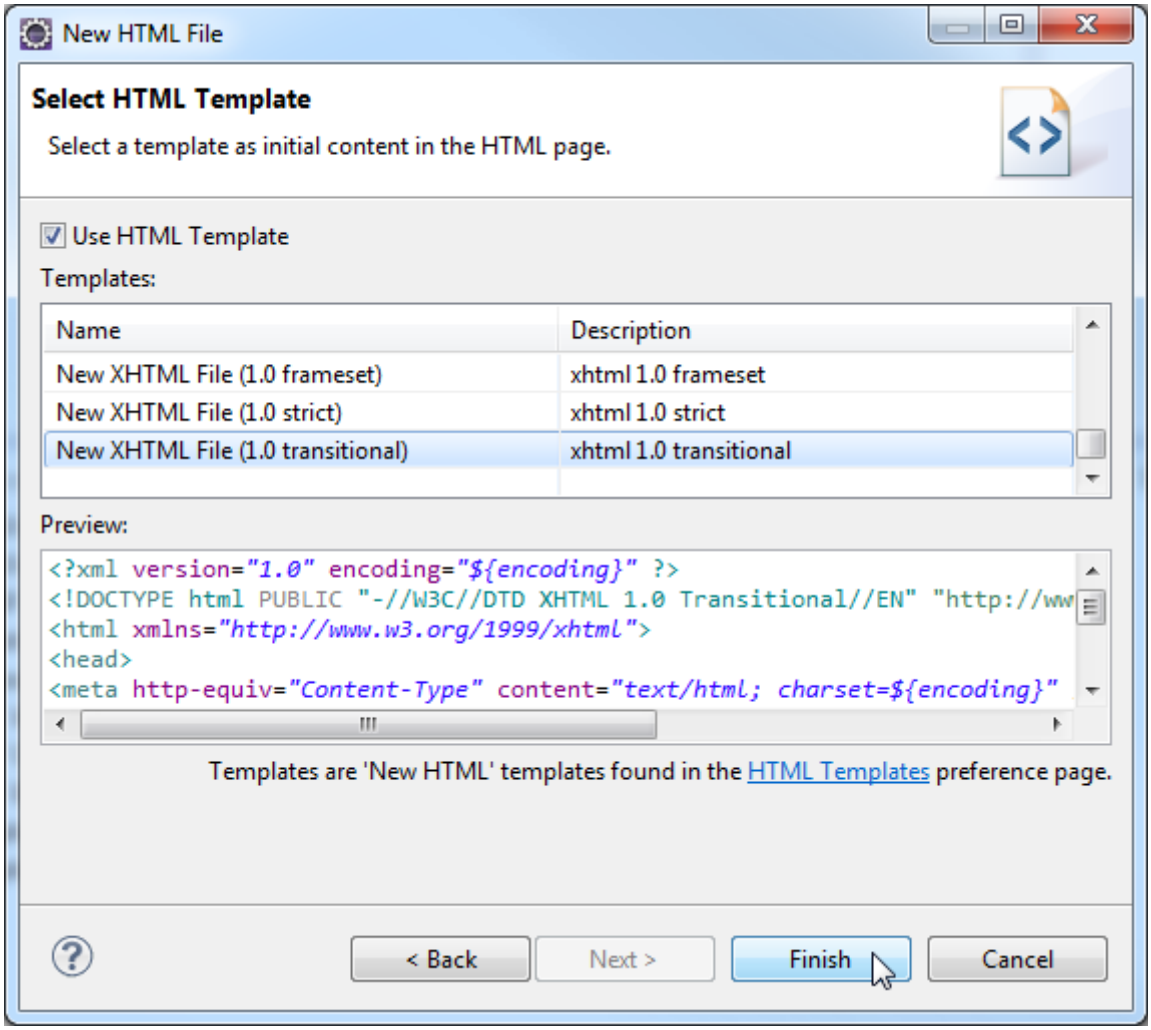
Zunächst wird die Startseite index.xhtml (Name beliebig) erstellt. Dazu wird ein Rechtsklick auf dem Ordner WebContent gemacht und „New“ mit „HTML File“ ausgewählt.



Hier wird der Name der Seite eingegeben und „Next>“ gedrückt.



Im oberen Teil wird dann „New XHTML File (1.0 transitional)“ ausgewählt und „Finish“ gedrückt.



Es steht dann ein Editor für XHTML-Dateien zur Verfügung der wie üblich über Strg+<Leertaste> Code-Vervollständigungen anbietet. Im Beispiel wird folgende Seite eingegeben. Man beachte, dass beim html-Tag zusätzlich der Namensraum `xmlns:h="http://xmlns.jcp.org/jsf/html"` ergänzt wurde.

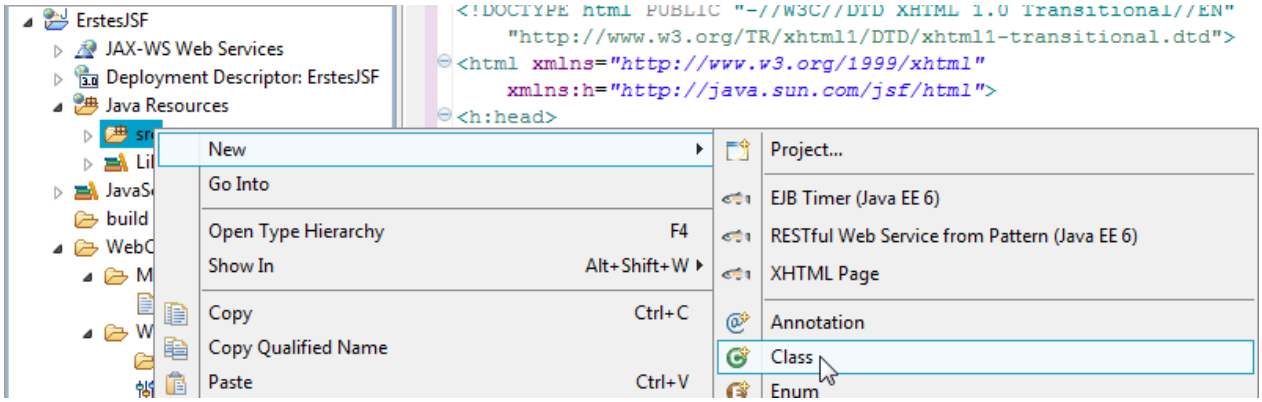
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
         xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
         <title>Moduleingabe</title>
</h:head>
<h:body>
         <h:form>
                  <h:outputText value="Modulname " />
                  <h:inputText id="mname" value="#{modul.name}" required="true" />
                  <br />

                  <h:outputText value="Modulnummer " />
                  <h:inputText id="mnr" value="#{modul.nr}" required="true" />
```

```
                <br />
                <h:commandButton value="Abschicken" action="#{modul.uebernehmen}" />
        </h:form>
</h:body>
</html>
```

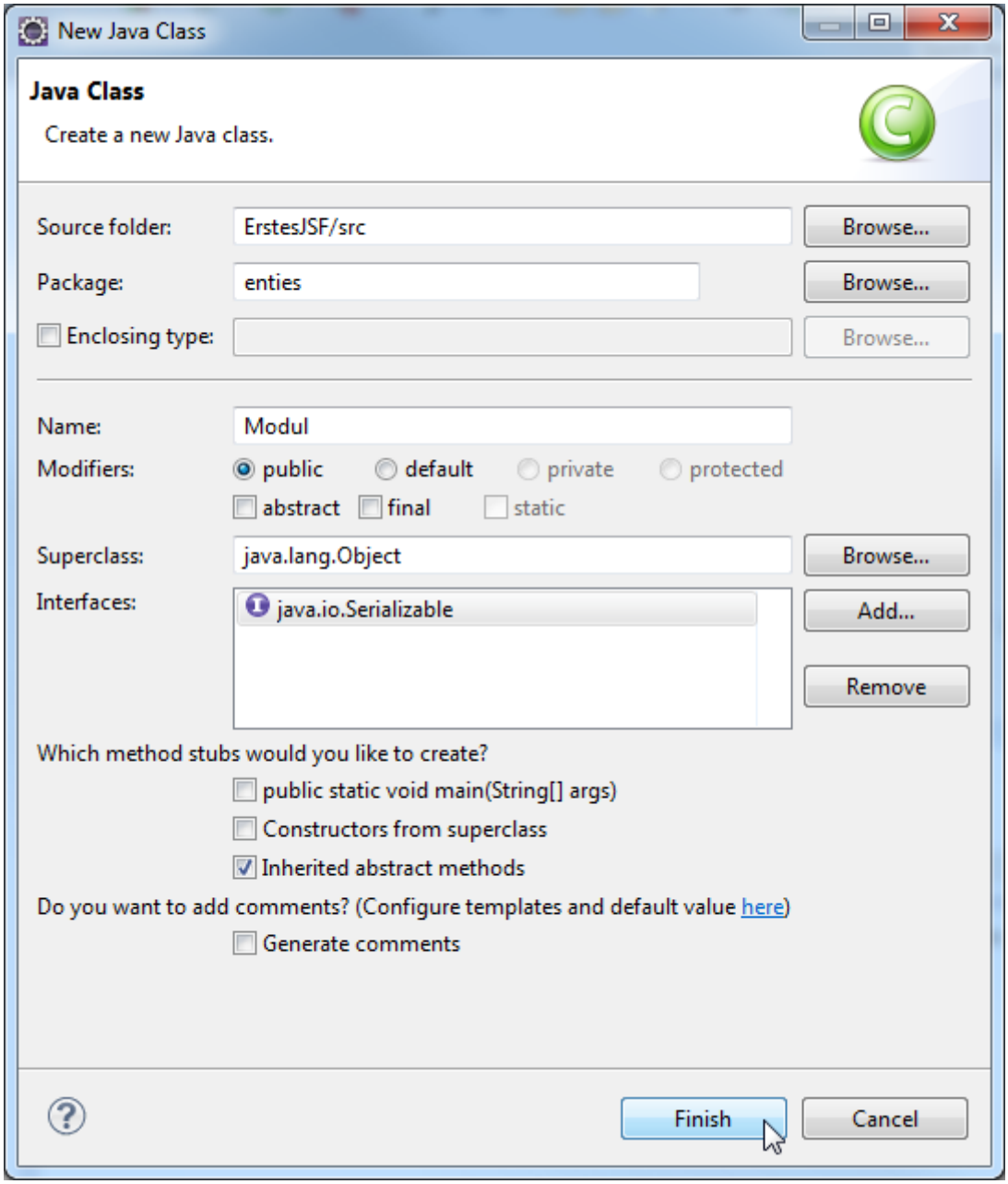
Analog zur vorherigen Seite wird jetzt eine zweite Seite mit dem Namen ausgabe.xhtml und folgendem Inhalt eingegeben.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
        <title>Modulausgabe</title>
</h:head>
<h:body>
        <h:form>
                <h:outputText value="Modulname: " />
                <h:outputText id="mname" value="#{modul.name}" />
                <br />

                <h:outputText value="Modulnummer: " />
                <h:outputText id="mnr" value="#{modul.nr}" />
                <br />
                <h:commandButton value="Zur Eingabe" action="#{modul.eingeben}" />
        </h:form>
</h:body>
</html>
```

Nun fehlt noch die Managed Bean, die über den Namen modul in den XHTML-Seiten angesprochen wird. Hierzu wird die folgende Klasse Modul angelegt. Dazu wird ein Rechtsklick auf den Unterordner src des Ordners Java Resources gemacht, „New“ und „Class“ gewählt und dann wie gewohnt die Klasse angelegt.



Man hier bereits das Interface Serializable angeben.



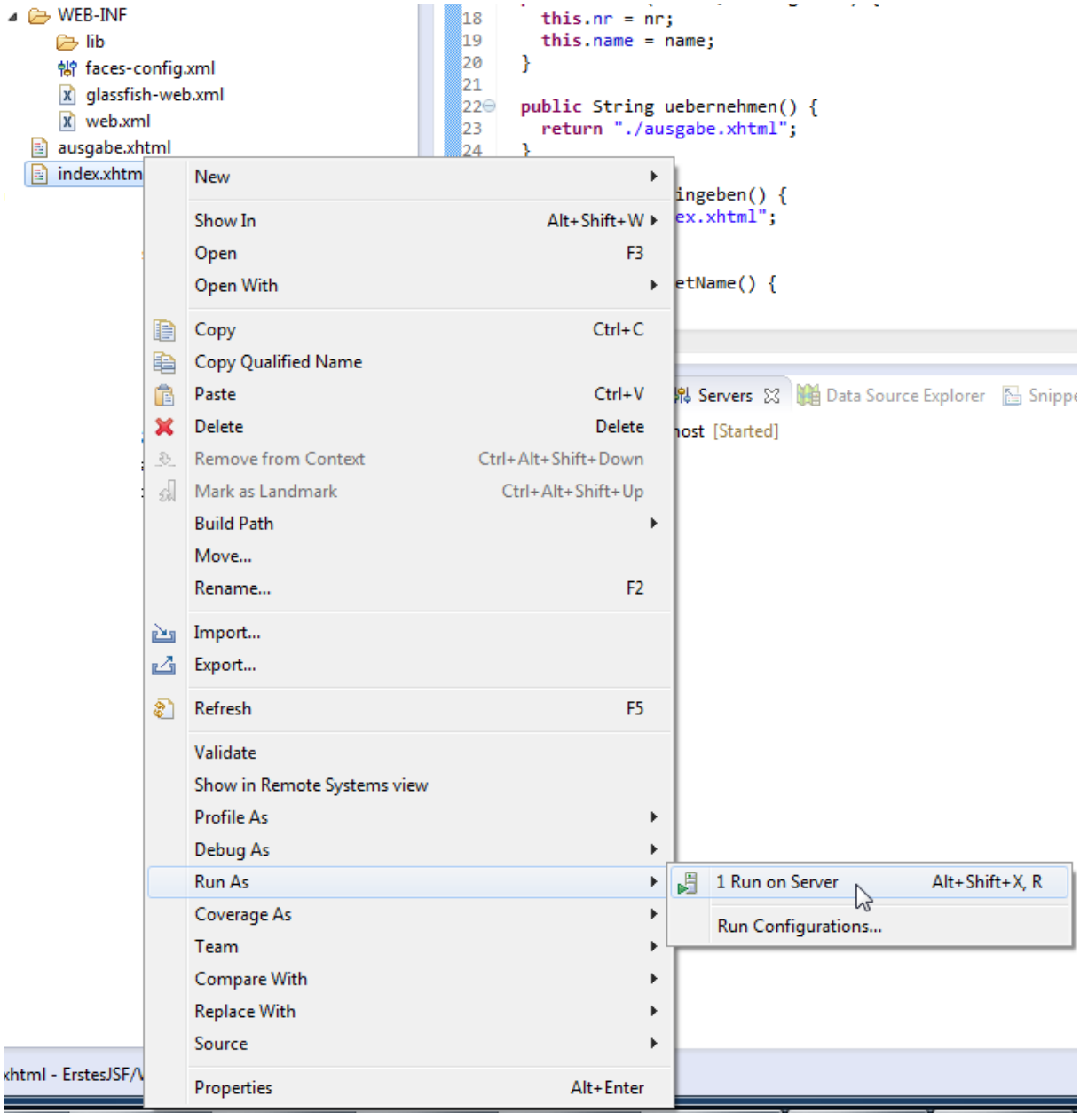
Die Klasse hat folgenden Inhalt.

```
package entities;

import java.io.Serializable;
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named("modul")
@RequestScoped
public class Modul implements Serializable {
	private static final long serialVersionUID = 1L;
	private int nr;
	private String name;
```

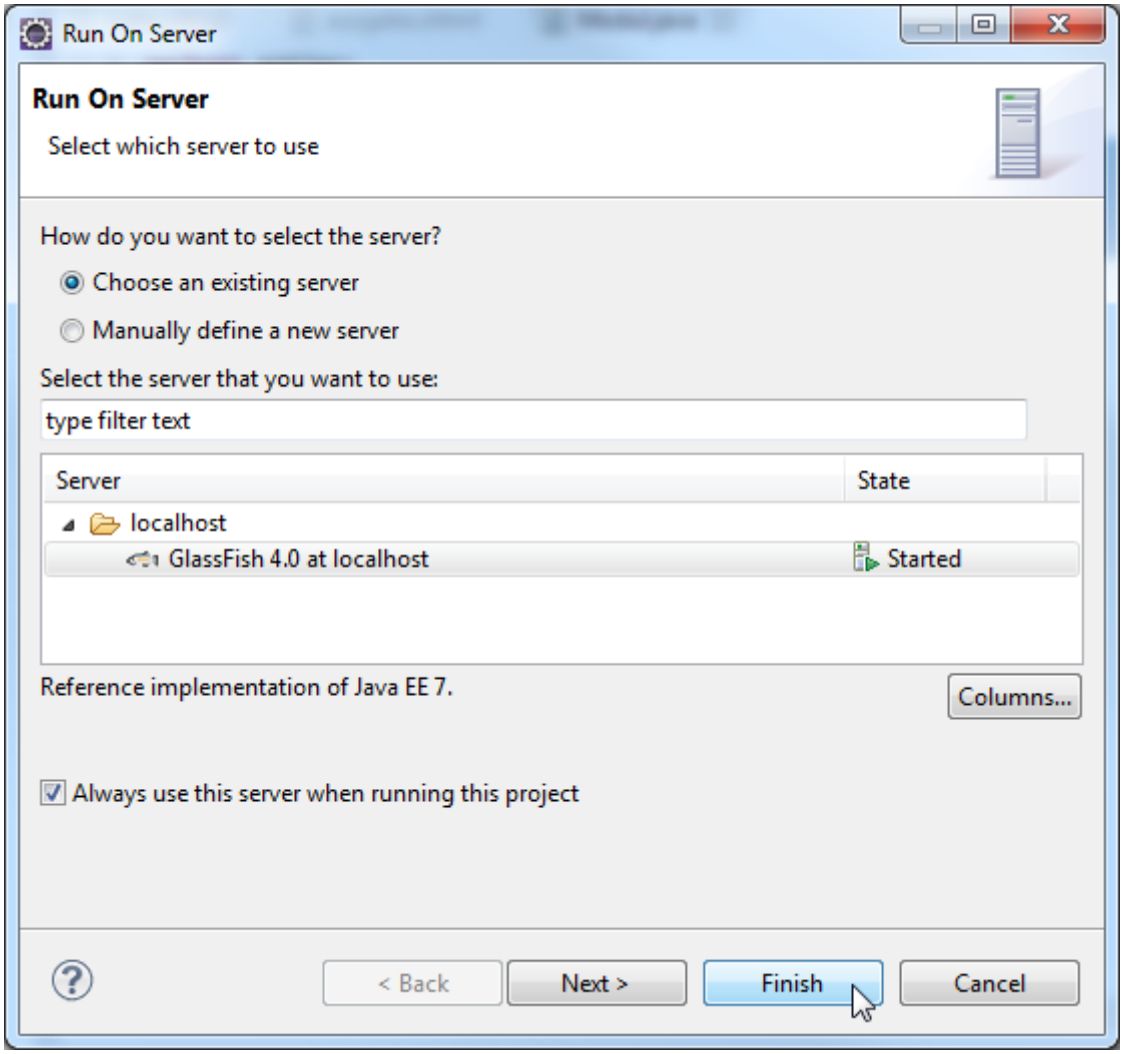
```
public Modul() {
}

public Modul(int nr, String name) {
	this.nr = nr;
	this.name = name;
}

public String uebernehmen() {
	return "./ausgabe.xhtml";
}

public String eingeben() {
	return "./index.xhtml";
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public int getNr() {
	return nr;
}

public void setNr(int nr) {
	this.nr = nr;
}

@Override
public boolean equals(Object object) {
	if (object == null || !(object instanceof Modul))
		return false;
	Modul other = (Modul) object;
	if (this.nr != other.nr || !this.name.equals(other.name))
		return false;
	return true;
}

@Override
// generieren lassen
public int hashCode() {
	int hash = 5;
	hash = 47 * hash + this.nr;
	hash = 47 * hash + (this.name != null ? this.name.hashCode() : 0);
	return hash;
}

@Override
public String toString() {
	return name + "(" + nr + ")";
```

```
        }
}
```

Generell kann man die Applikation jetzt über einen Rechtsklick auf einer XHTML-Seite und der Wahl „Run As“ mit „Run on Server“ starten.



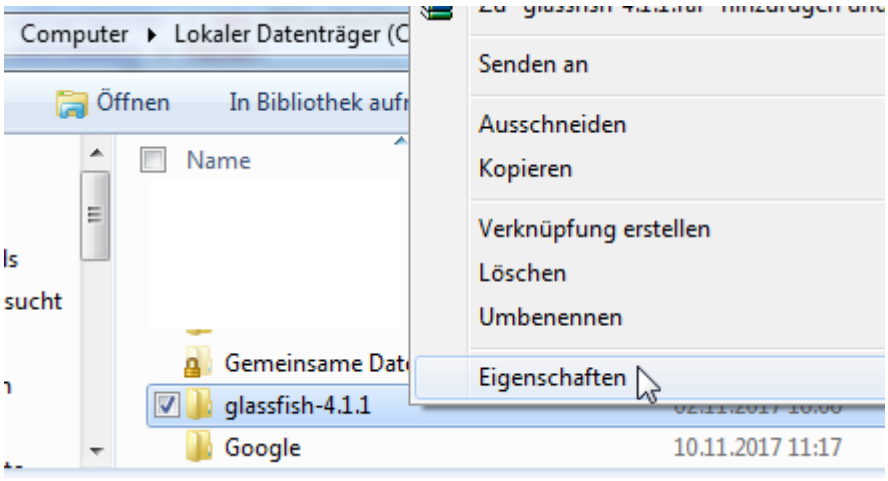
Man wird dann aufgefordert, den Server zu wählen, der hier einfach übernommen werden kann. Um weitere Nachfragen zu vermeiden, kann der Haken links-unten bei „Always use…“ gesetzt werden. Die Ausführung wird mit „Finish“ gestartet. Sollte der Server noch nicht gestartet sein, wird er jetzt automatisch hochgefahren. Nach einiger Zeit erscheint dann die Web-Seite im internen Browser.



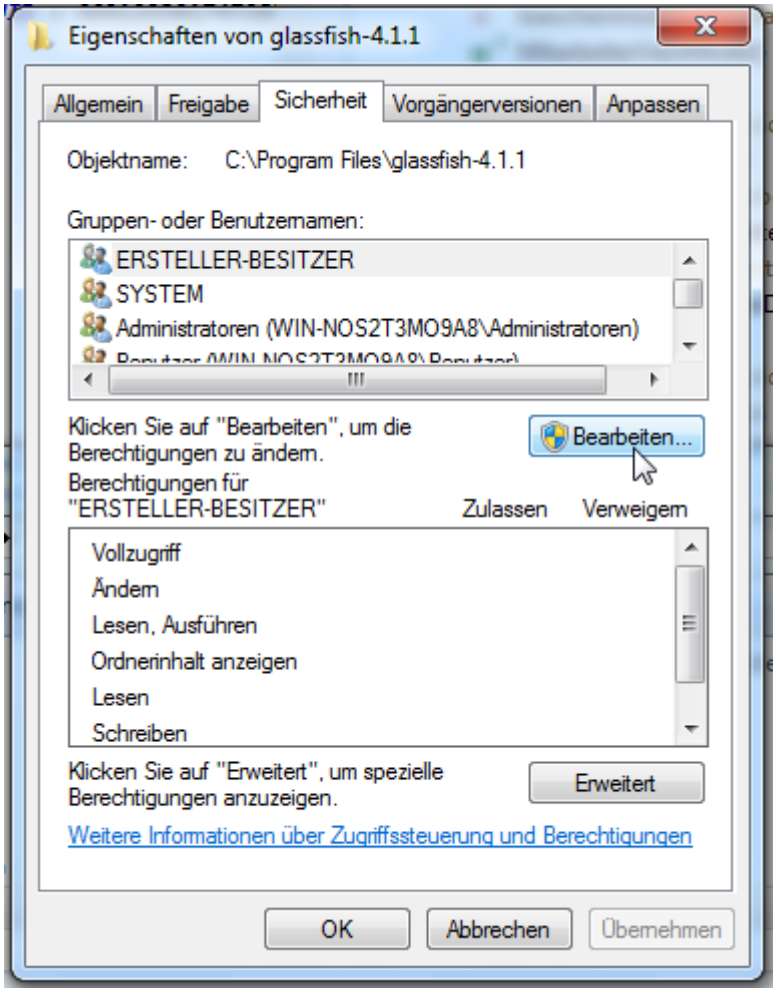
Sltte der Server nicht starten und eine Meldung der folgenden Form ausgegeben werden:

```
Launching GlassFish on Felix platform

ERROR: Unable to create cache directory
```

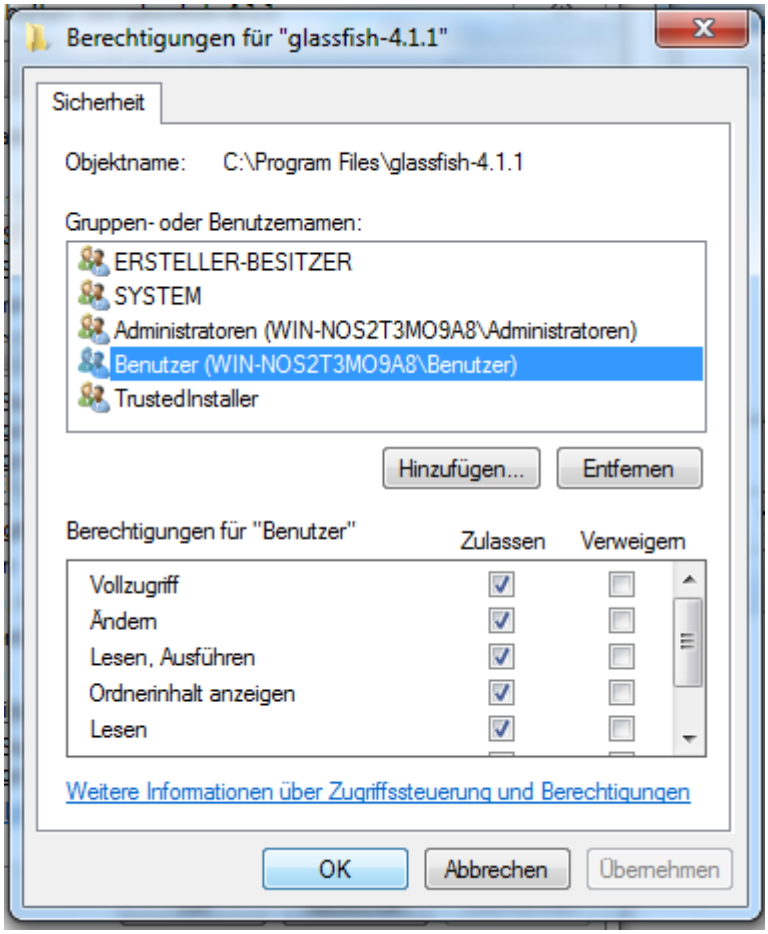
fehlen Ausführungsrechte für Eclipse. In diesem Fall wird ein Rechtsklick auf dem Installationsverzeichnis von GlassFish gemacht und „Eigenschaften“ ausgewählt.



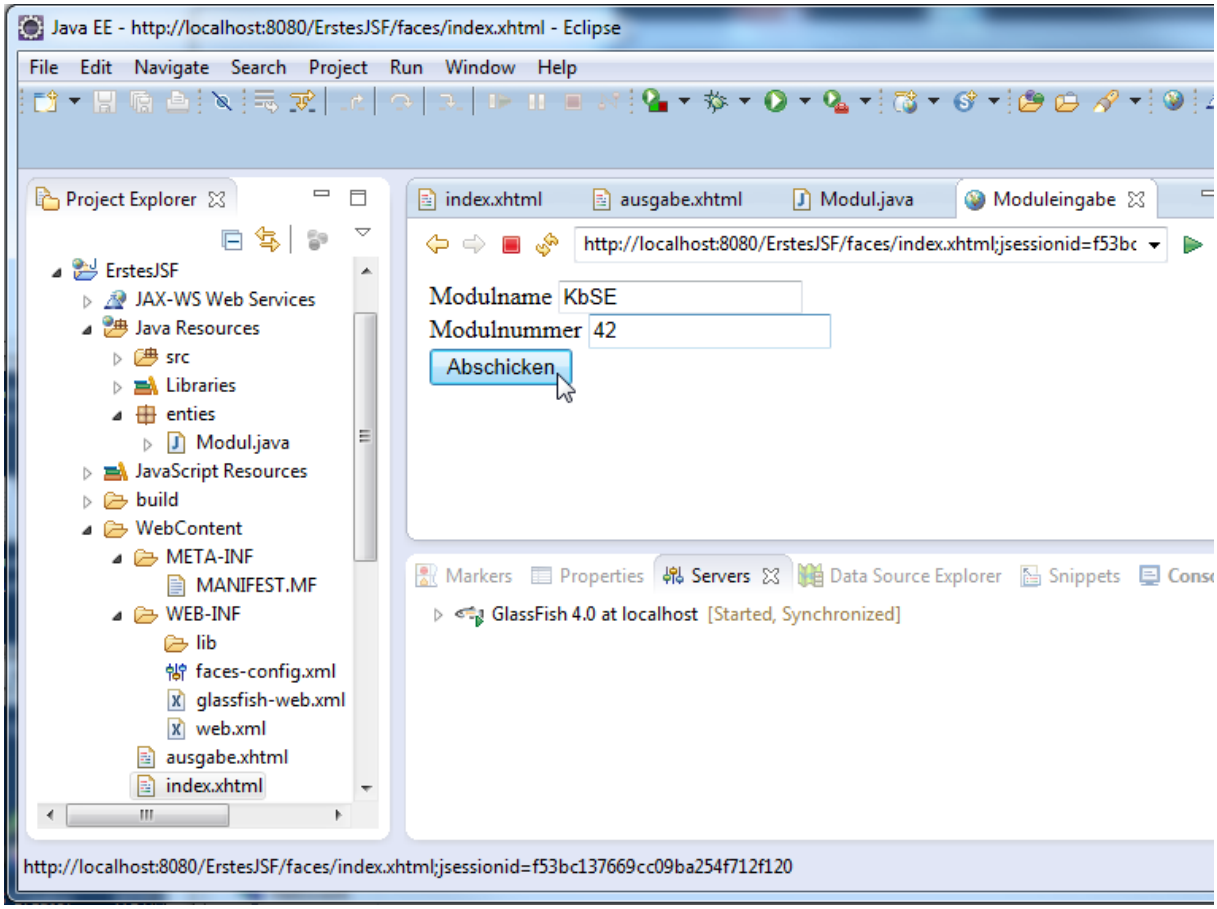
Oben wird der Reiter „Sicherheit“ angeklickt und rechts in der Mitte auf „Bearbeiten…“ geklickt.

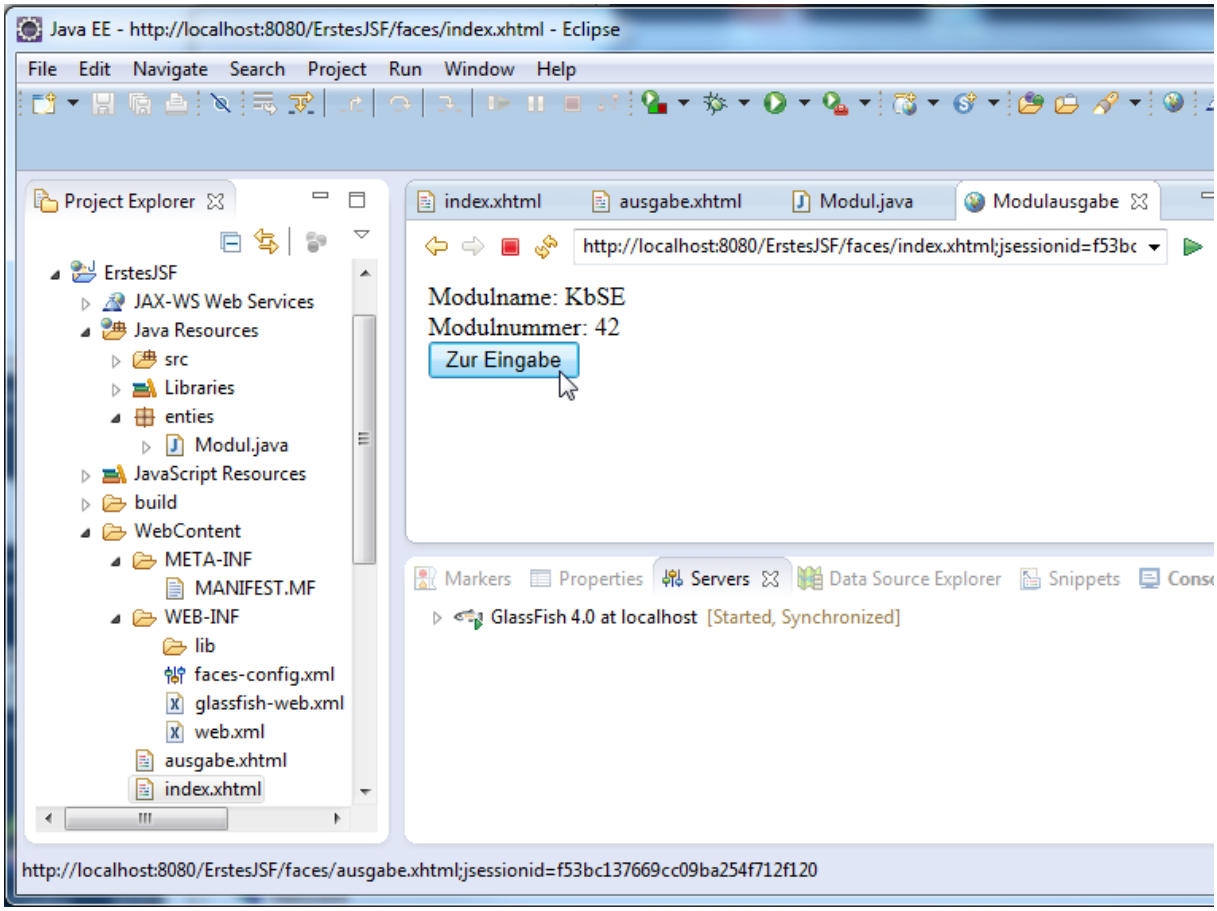


Oben wird „Benutzer …“ ausgewählt und unten dann Haken für „Zulassen“ an allen Stellen gesetzt. Die Änderungen werden mit „OK“ und „OK“ übernommen. Danach witrd erneut versucht das Projekt zu starten.



Der Browser kann wie üblich genutzt werden, um die Funktionalität zu testen. Wird der Browser nicht mehr benötigt, sollte er über das X im Browser-Reiter geschlossen werden, da bei jedem Start ein neues Browser-Fenster geöffnet wird.


Java EE - http://localhost:8080/ErstesJSF/faces/index.xhtml - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
ErstesJSF
JAX-WS Web Services
Java Resources
src
Libraries
enties
Modul.java
JavaScript Resources
build
WebContent
META-INF
MANIFEST.MF
WEB-INF
lib
faces-config.xml
glassfish-web.xml
web.xml
ausgabe.xhtml
index.xhtml
index.xhtml
ausgabe.xhtml
Modul.java
Moduleingabe
http://localhost:8080/ErstesJSF/faces/index.xhtml;jsessionid=f53bc
Modulname KbSE
Modulnummer 42
Abschicken
Markers
Properties
Servers
Data Source Explorer
Snippets
GlassFish 4.0 at localhost [Started, Synchronized]
http://localhost:8080/ErstesJSF/faces/index.xhtml;jsessionid=f53bc137669cc09ba254f712f120



Möchte man eine Seite als Startseite für die Web-Applikation festlegen, muss diese im Deployment-Decriptor web.xml festgelegt werden. Dazu wird die im Ordner WEB_INF befindliche Datei mit einem Doppelklick geöffnet und folgender Eintrag in der vorletzten Zeile ergänzt.

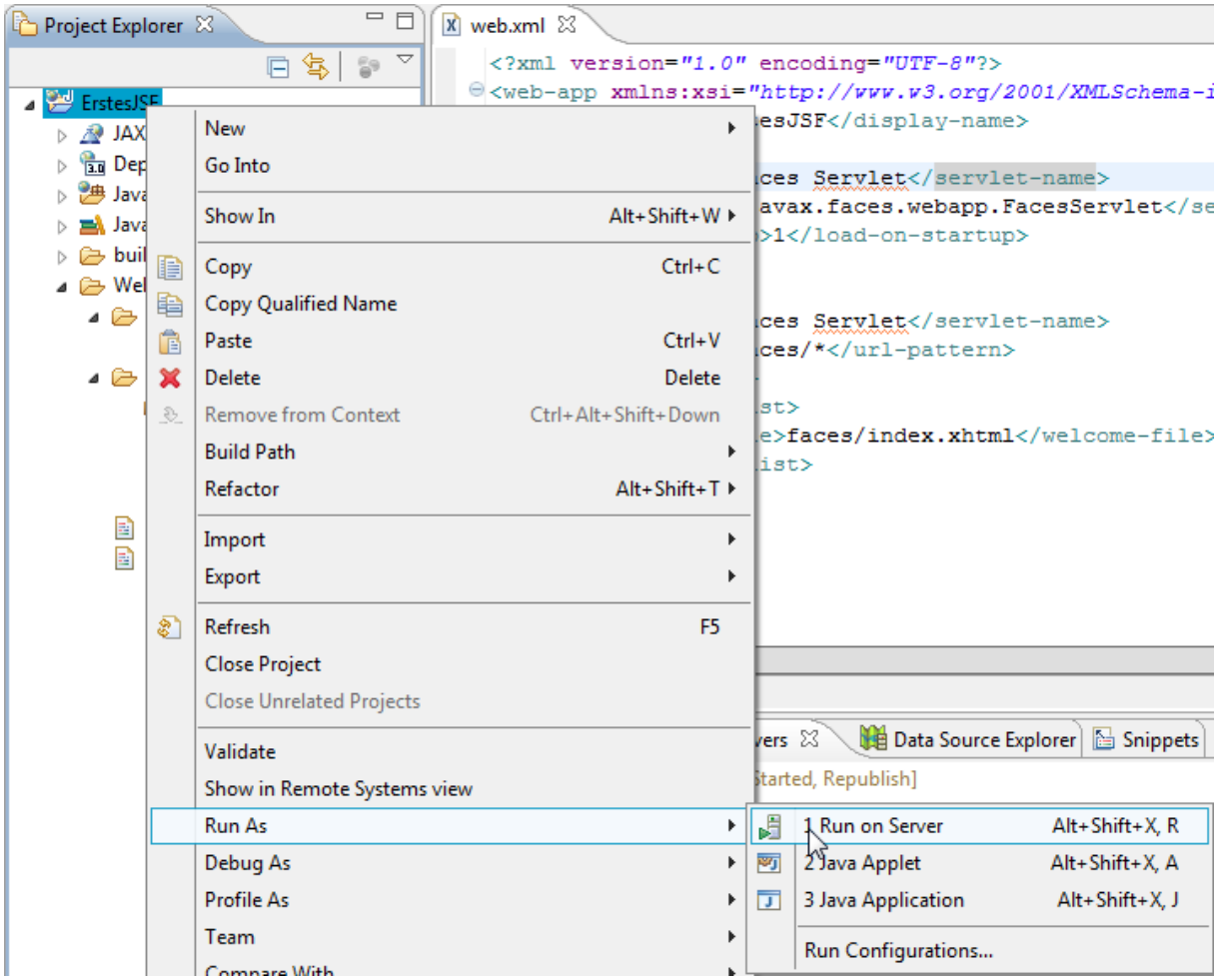
```
<welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
</welcome-file-list>
```

Die resultierende Datei ist in der folgenden Abbildung dargestellt.

web.xml

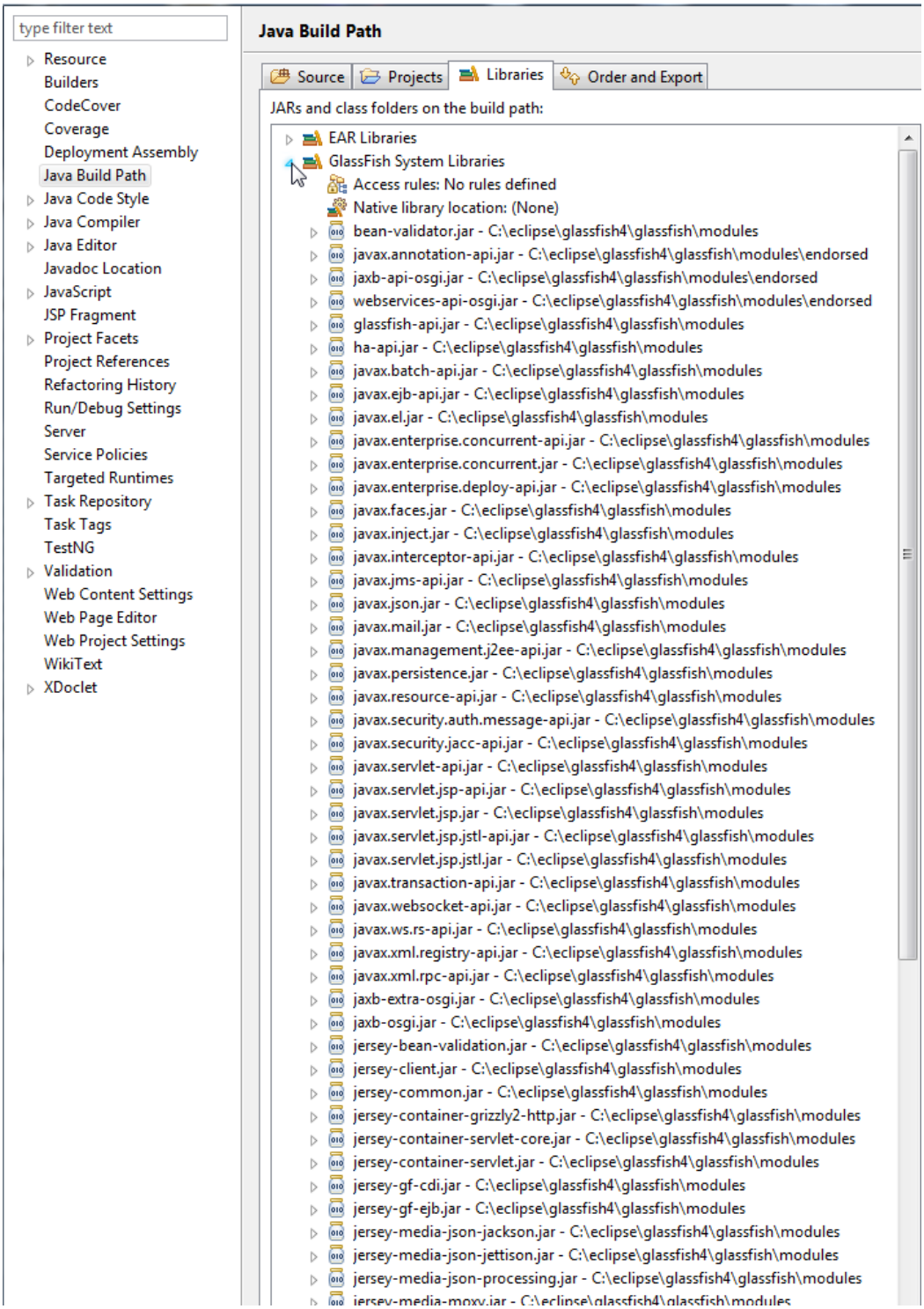
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns
  <display-name>ErstesJSF</display-name>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Nun kann die Applikation auch über einen Rechtsklick auf dem Projekt mit „Run As“ mit „Run on Server“ gestartet werden. Beim erstsen Aufruf muss wie vorher beschrieben, die Server-Konfiguration mit „Finish“ bestätigt werden.



20.3 Nutzung von Bean Validation in Eclipse JEE

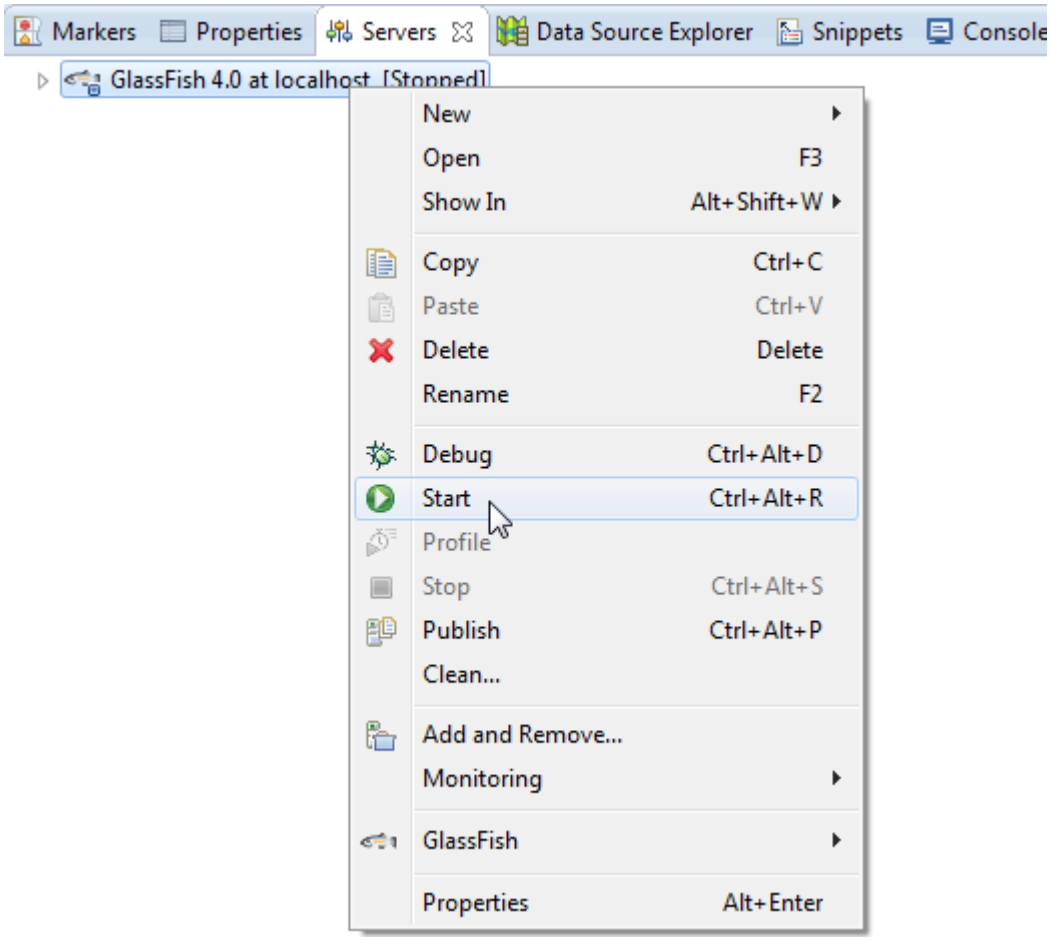
Hat man GlassFish wie beschrieben installiert, muss man keine Realisierung der Bean Validation als Bibliotheken in das Projekt ergänzen, da diese bereits mit dem Server mitgeliefert werden. Einen Überblick erhält man, wenn man für ein Projekt mit Rechtsklick die Properties, dann den Java Build Path, dann den Reiter Libraries und dann das Jar zu GlassFish genauer anschaut.



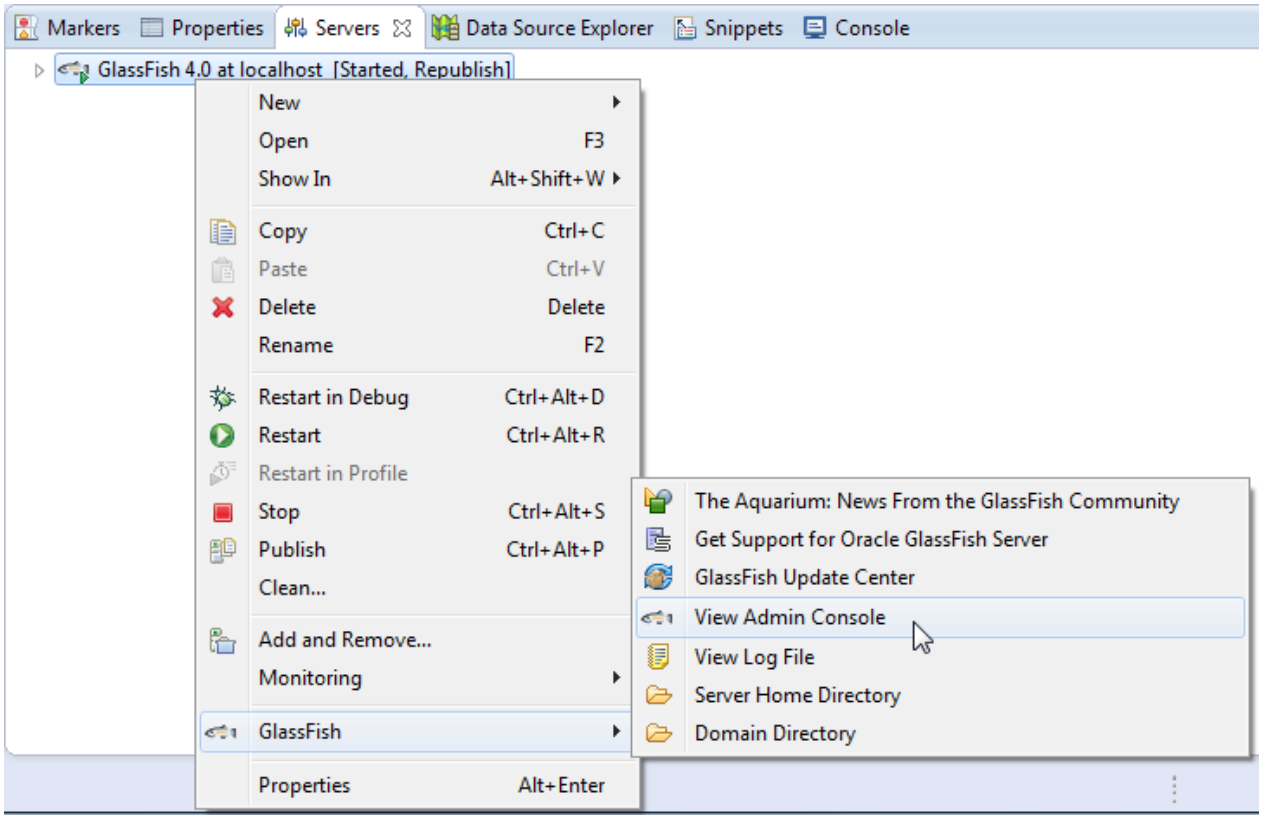
20.4 Einrichtung einer Datenbank im GlassFish

Möchte man seine Daten persistieren und soll diese Persistierung durch den Server stattfinden, muss diese Datenbank dem Server bekannt sein. Der eigentliche Zugriff erfolgt über JNDI und wird im nachfolgenden Unterkapitel behandelt.

Zunächst wird der GlassFish-Server mit einem Rechtsklick und „Start“ gestartet, falls dieser nicht bereits laufen sollte..

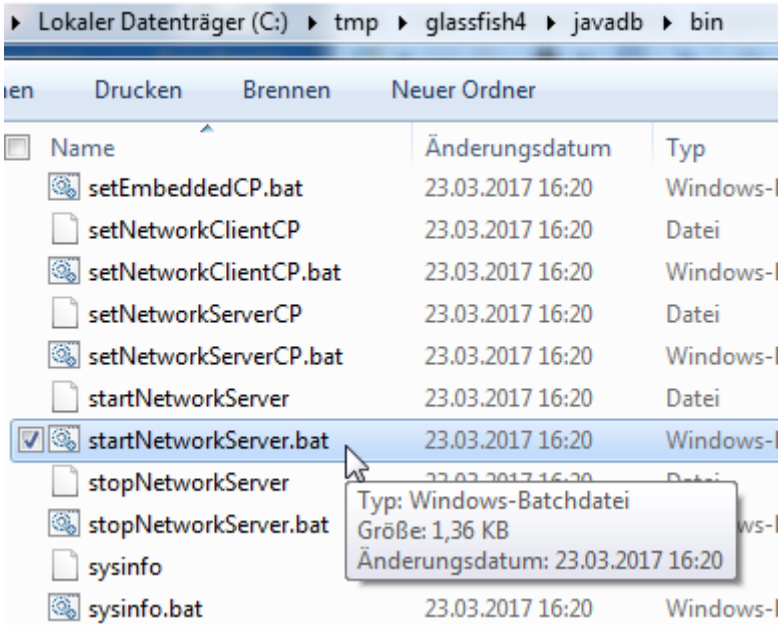


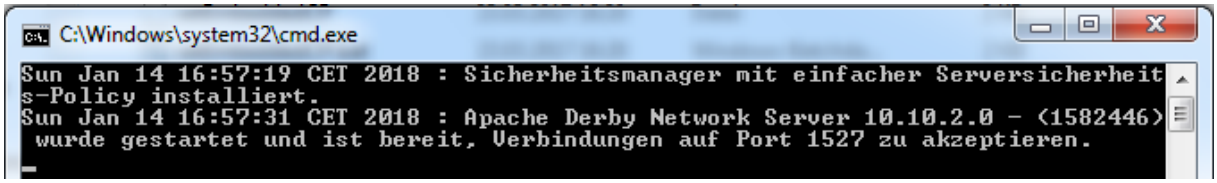
Im nächsten Schritt wird die Administrations-Console aufgerufen. Dies geschieht mit einem Rechtsklick auf dem Server, es wird „GlassFish“ und dann „View Admin Console“ gewählt. Alternativ kann in einem Browser http://localhost:4848 aufgerufen werden.



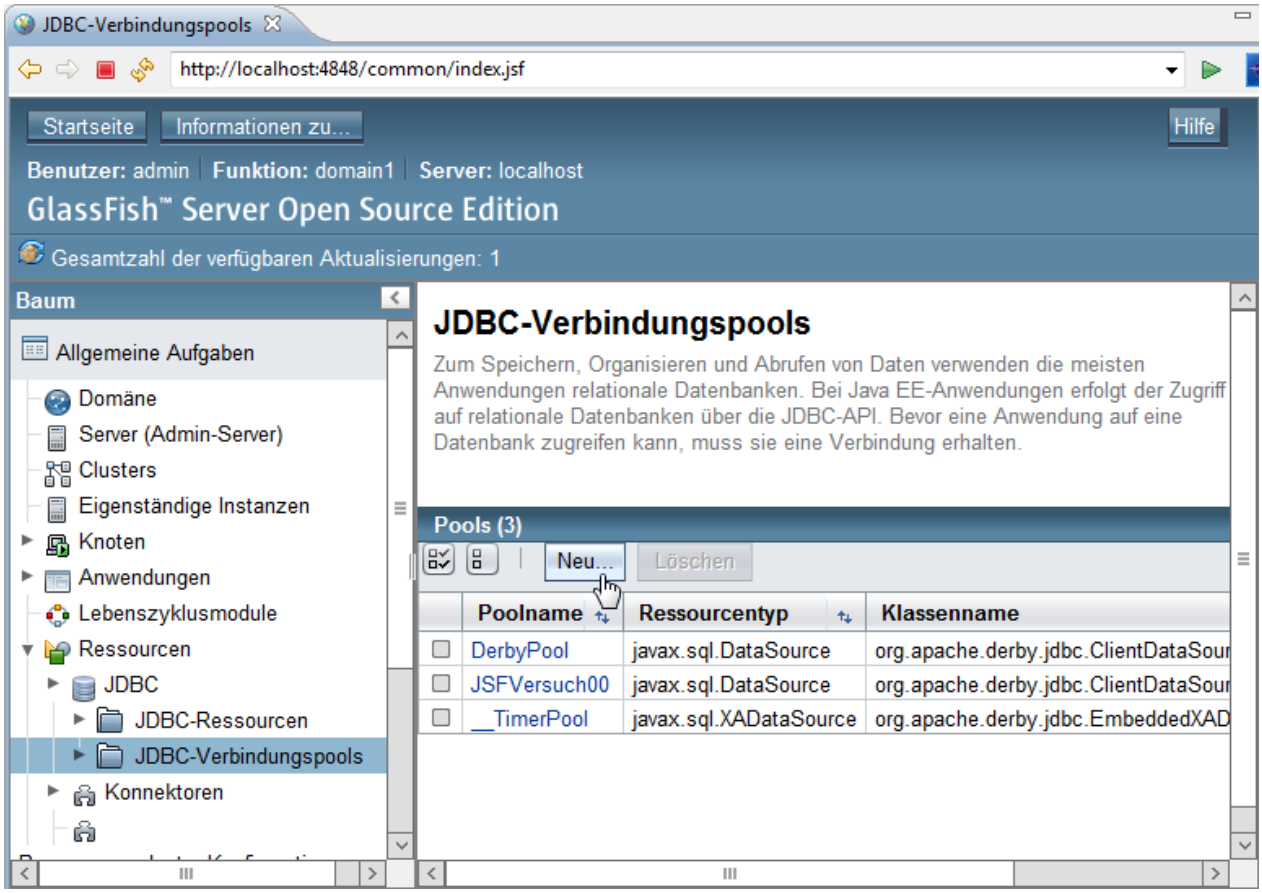
Die folgenden Schritte funktionieren leider nicht beim GlassFish 4.1.1, es muss eine aktuellere Version, z. B. 4.1.2 genutzt werden. Generell kann es in Eclipse mehrere Server geben, es ist nur zu beachten, dass entweder immer nur einer läuft oder dass ein anderer Port genutzt wird.

Weiterhin sollte die Datenbank laufen. Dies kann auch direkt geschehen, indem im Unterordner javadb/nin des GlassFish-Servers „startNewtworkServer.bat ausgeführt wird.



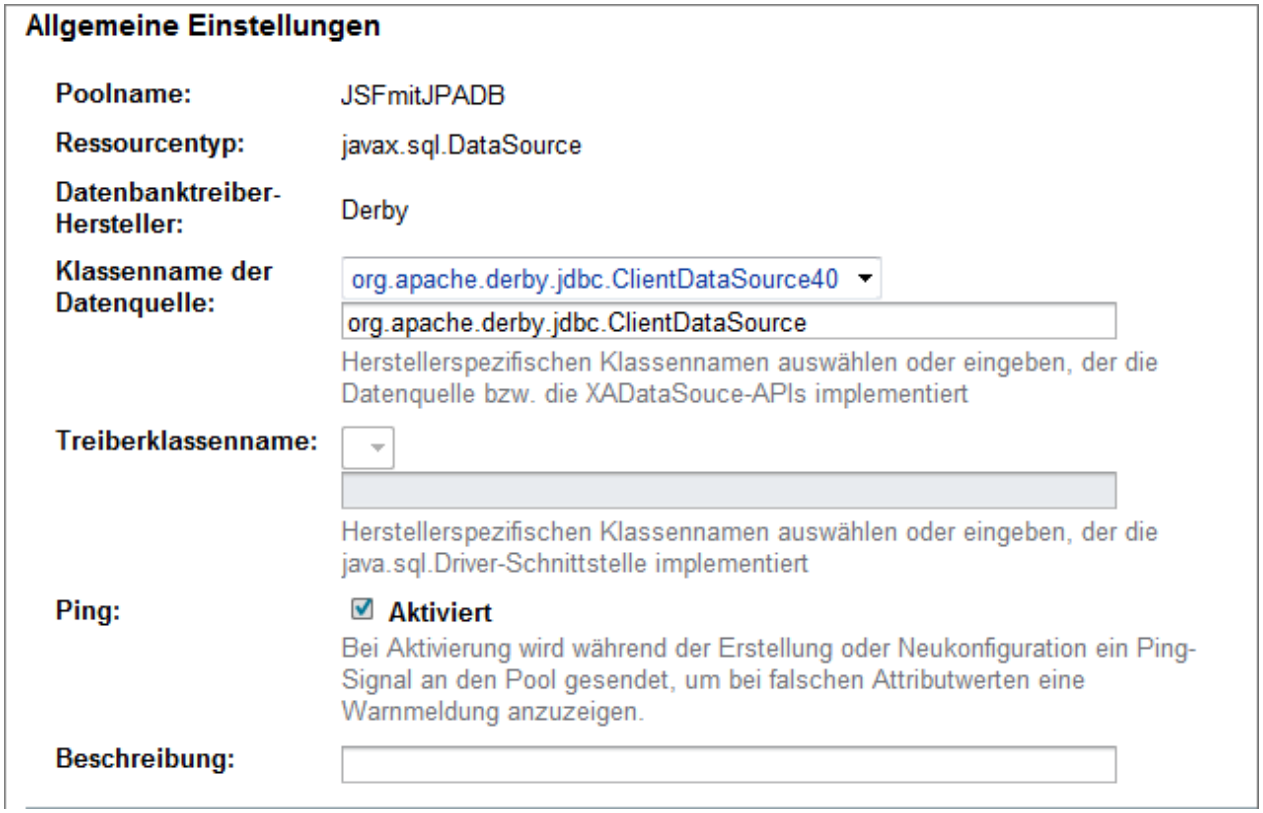


Nun soll zunächst eine Datenbank eingerichtet werden, die im folgenden Schritt über einen JNDI-Eintrag nutzbar werden soll. Dazu wird links „JDBC“ aufgeklappt, „JDBC-Verbindungspools“ gewählt und auf „Neu“ geklickt.

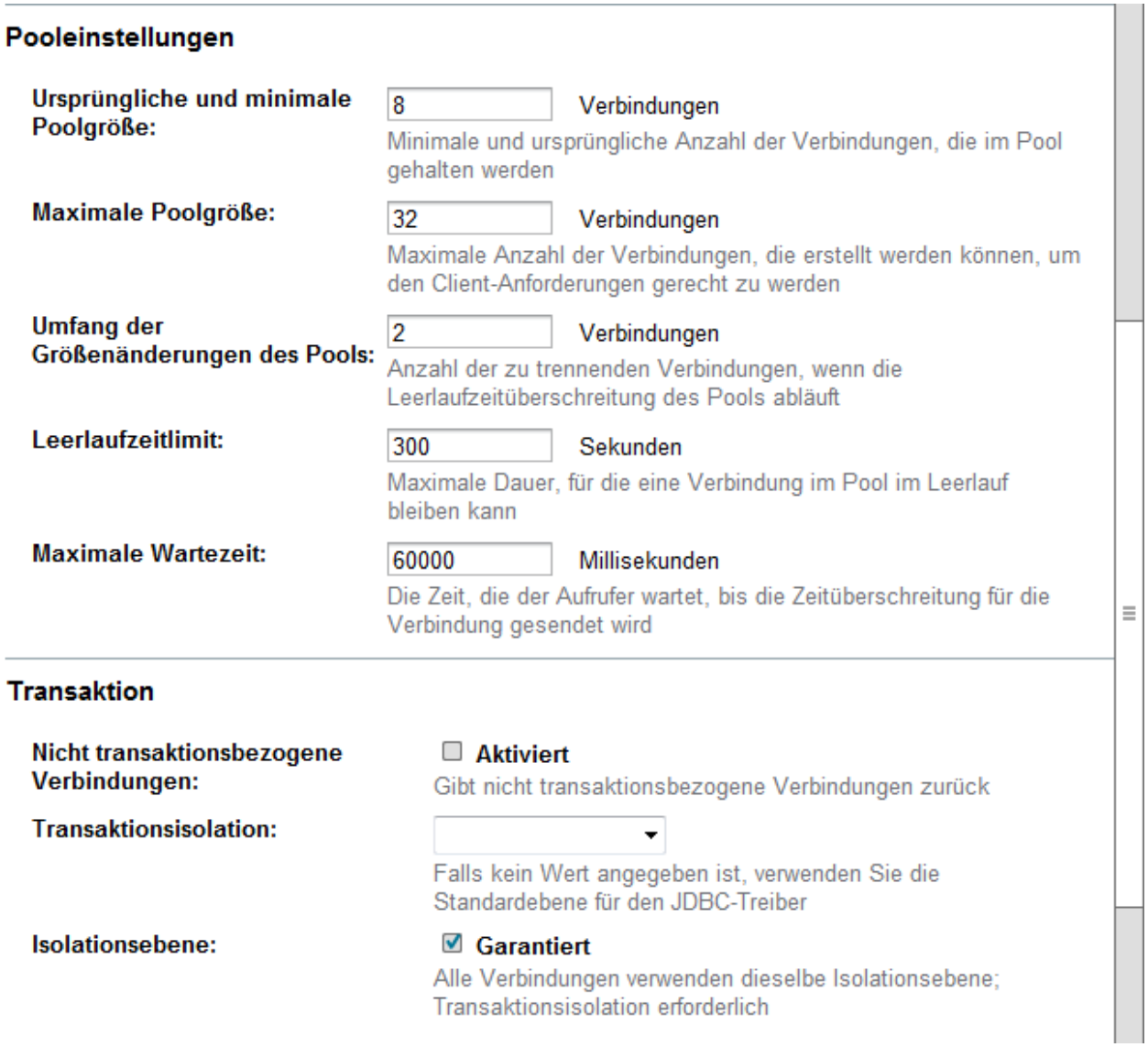


Es wird ein Poolname, der Ressourcentyp und der Datenbanktreiber-Hersteller angegeben und „Weiter“ geklickt.

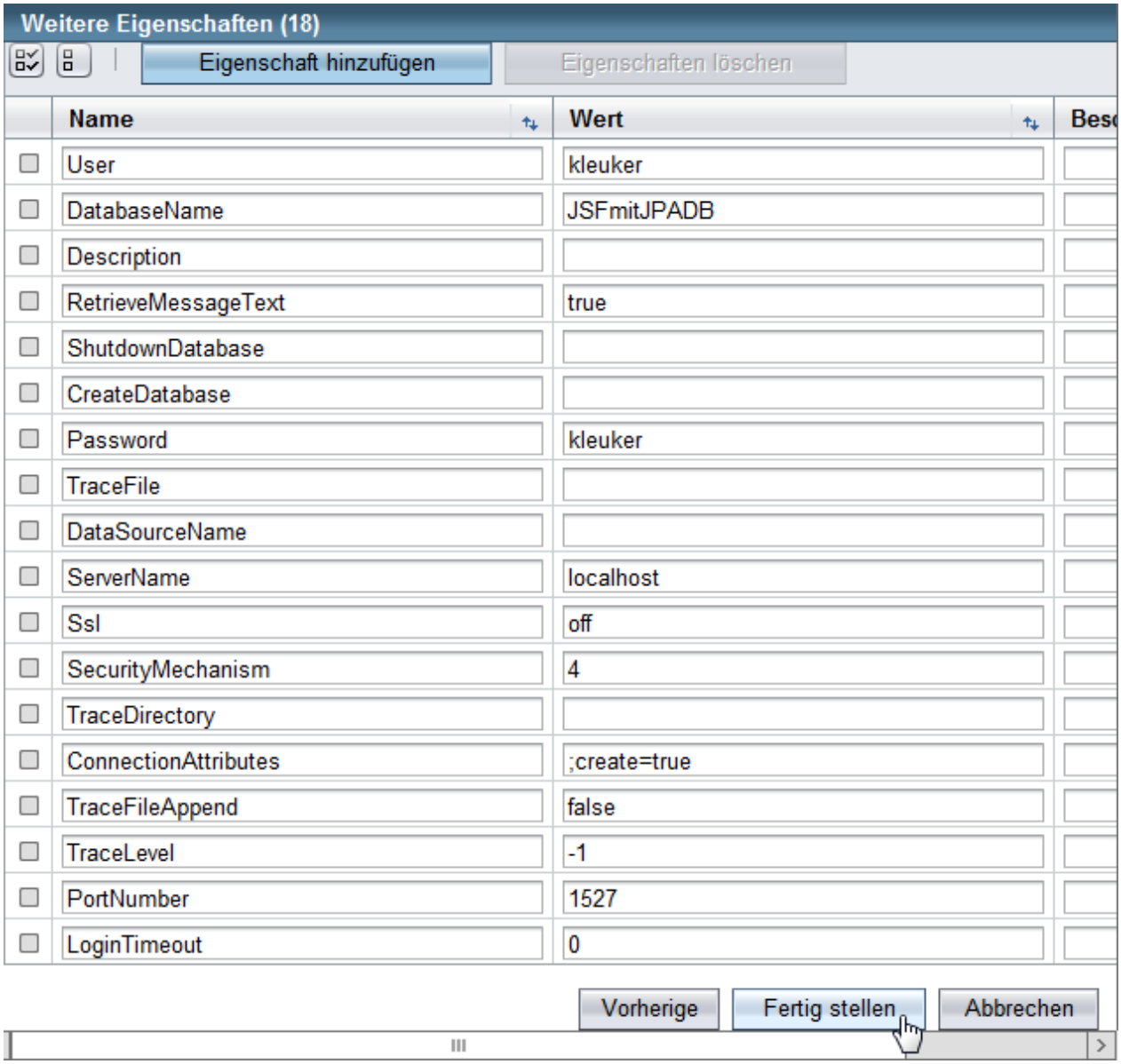
Beim Klassennamen der Datenquelle wird der angebene Name ohne die Endung „40“ eingegeben. Gegebenenfalls wird der Eintrag automatisch angepasst, dieses Anpassung wird dann genutzt. Der Haken beim „Ping“ kann gesetzt werden, da so nach der Einrichtung geprüft wird, ob eine Verbindung aufgebaut werden kann.



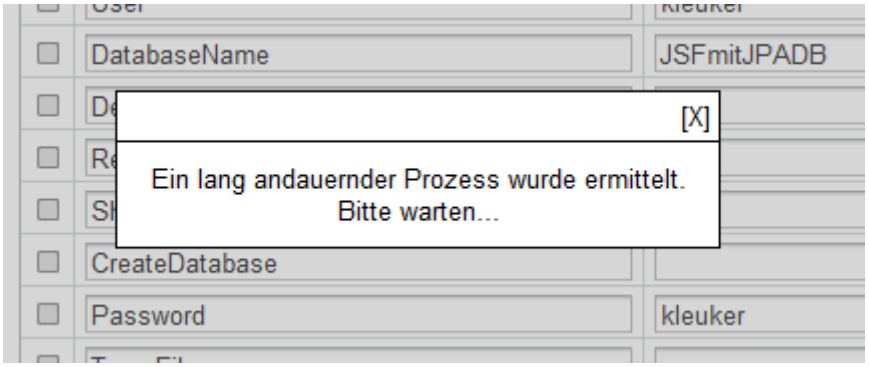
Weiter unten auf der Seite müssen keine Änderungen vorgenommen werden. Man beachte aber, dass man hier die Transaktionssteuerung für die Datenbank einstellt, die zentrale Bedeutung für das korrekte Verhalten des Gesamtsystems haben kann.



Nun wird zum Ende der Seite gescrollt und es werden die Einträge bei „User“, „Password“, „DatabaseName“ und „connectionAttributes“ (;create=true) geändert und „Fertig stellen“ gedrückt.

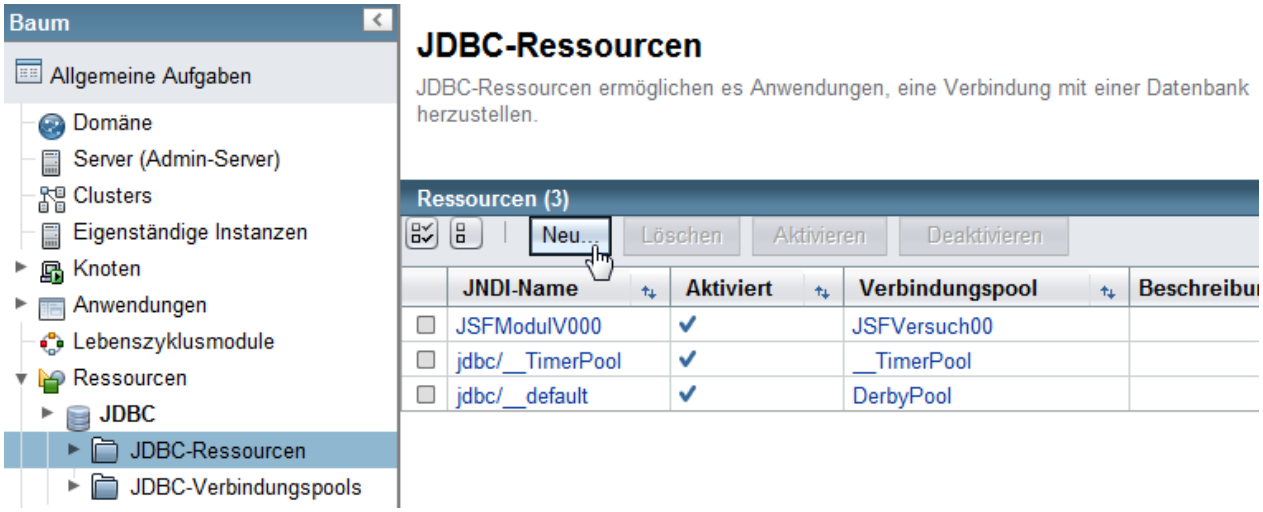


Nun wird versucht die Datenbank direkt zu starten, was mit einer Meldung angezeigt wird.



Man erhält die Meldung, dass die Verbindung geklappt hat, mit einem „Ping erfolgreich“.

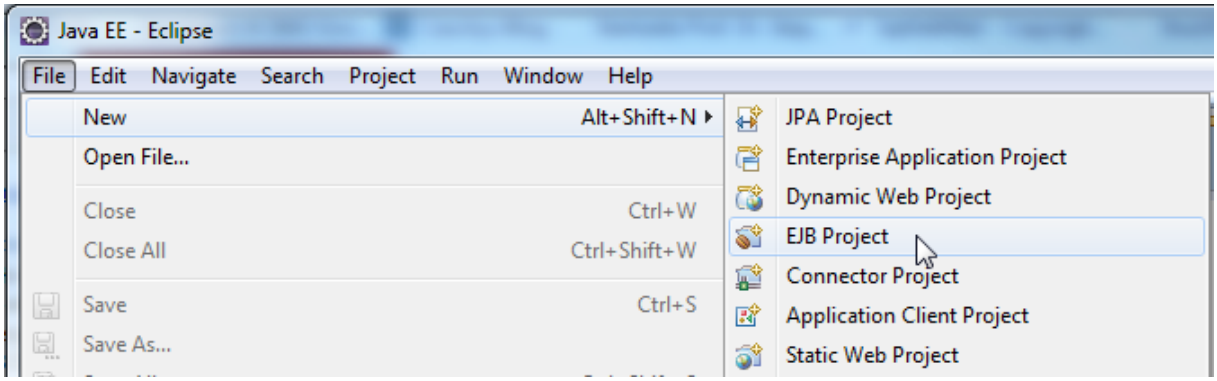
Am linken Rand wird dann JDBC-Ressourcen gewählt und auf „Neu“ geklickt.



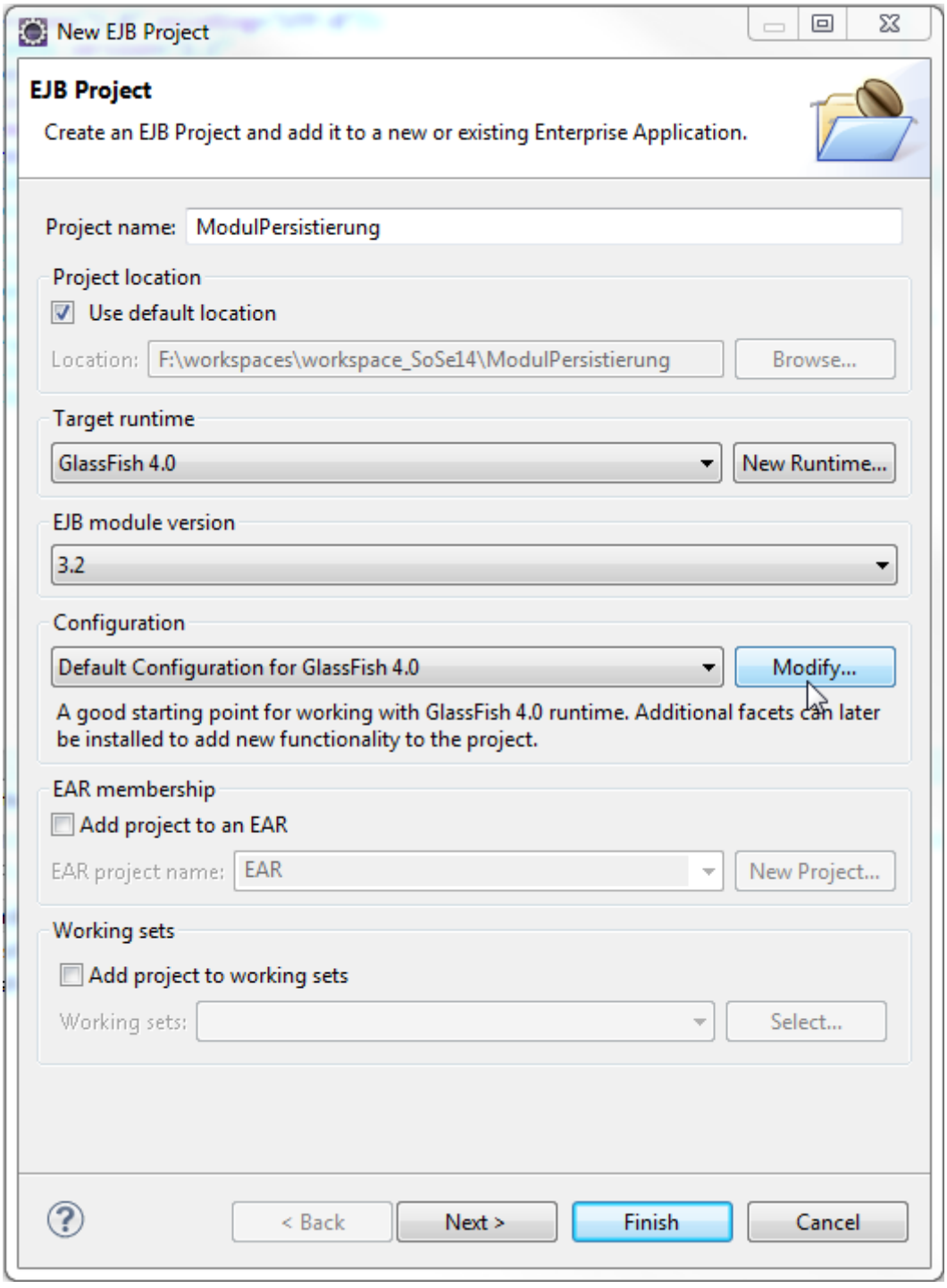
Hier wird ein JNDI-Name eingegeben, der frei wählbar ist, wobei es eine Konvention gibt, dass der Name mit „jdbc/“ anfängt. Als Verbindungspool wird unter Poolname der gerade eingerichtete Pool, also die Datenbankverbindung gewählt und auf „OK“ geklickt, was die Konfiguration auf dem Server abschließt.

20.5 Einrichtung eines EJB-Projekts als Persistenz-Schicht

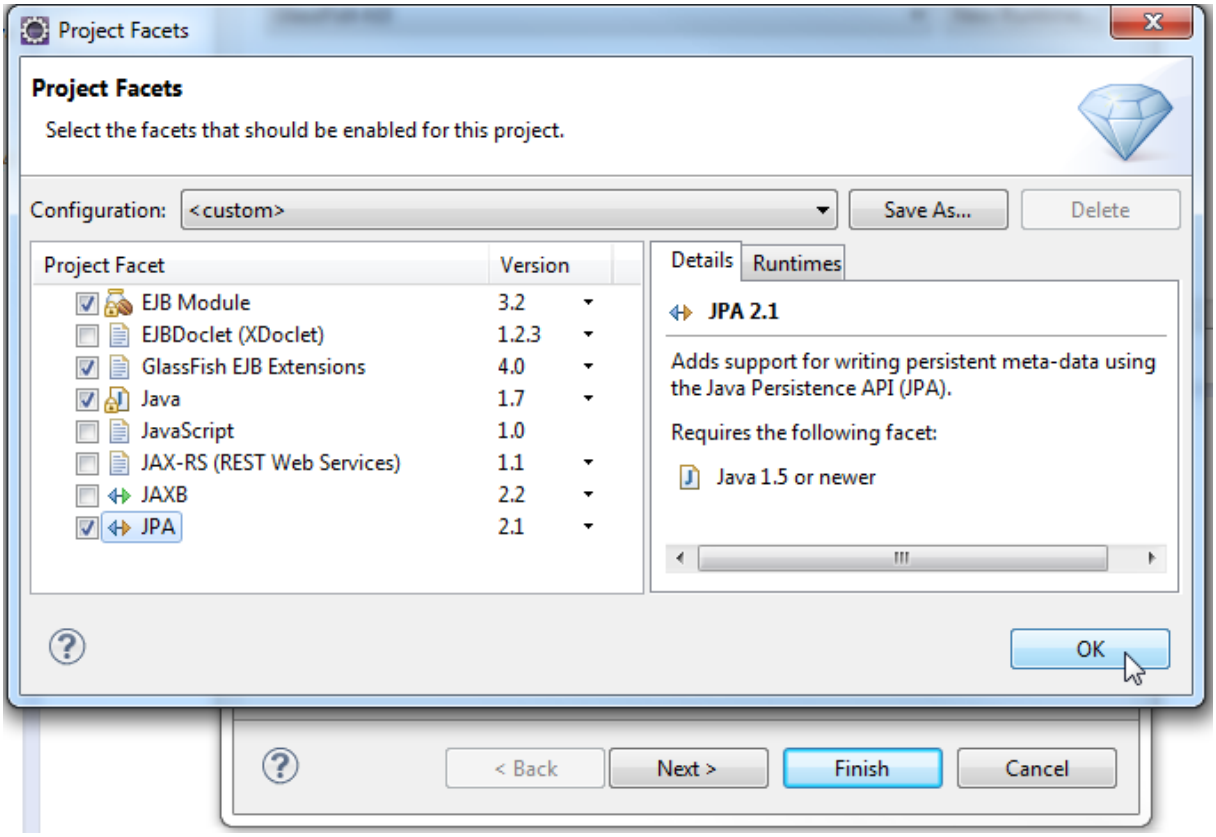
Das EJB-Projekt nutzt JPA zur Verbindung mit der Datenbank und erleichtert im Wesentlichen die systematische Transaktionssteuerung. Das entstehende Projekt kann dann in anderen Projekten zur Persistierung genutzt werden. Alternativ besteht die Möglichkeit alle Schritte in einem Dynamic Web Project zusammenzufassen und auf eine getrennte Bereitstellung des EJB-Projekts zu verzichten.



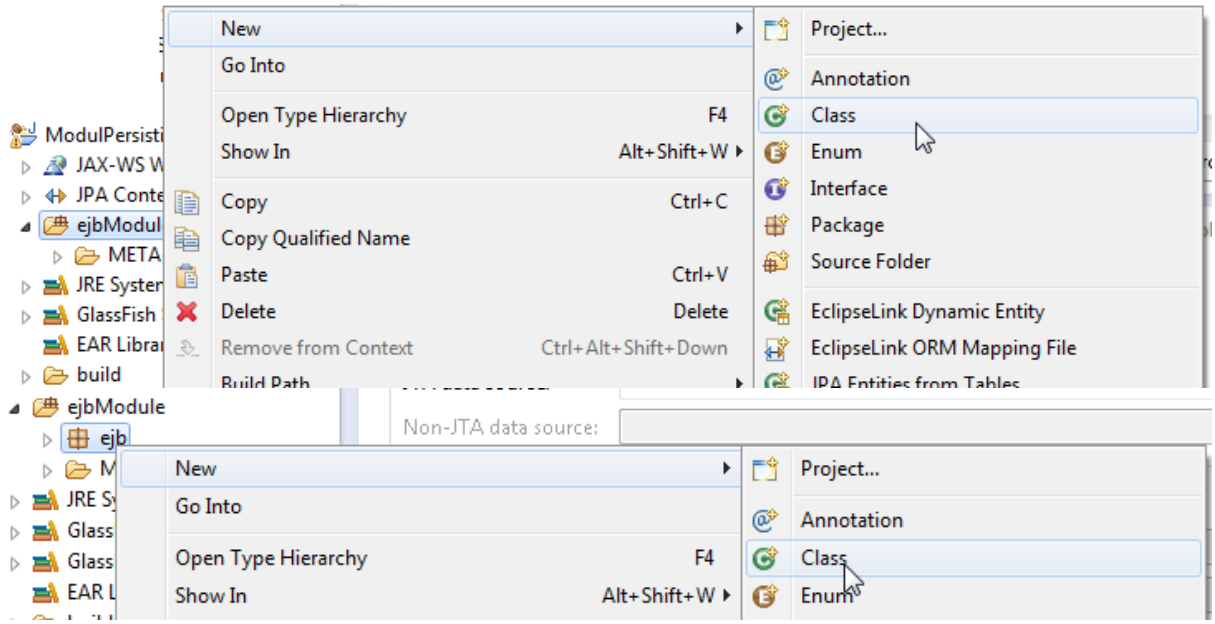
Es wird ein Projekt ModulPersistierung angelegt und rechts in der Mitte auf „Modify“ geklickt.



Links wird der Haken bei JPA gesetzt und „Ok“ und danach „Finish“ geklickt.



Nun werden die eigentliche Entitäts-Klasse bzw. die benötigten Entitätsklassen entwickelt. Es wird die Entity-Klasse als normale Java-Klasse unter ejbModule in einem Paket entities angelegt.



Die Klasse sieht wie folgt aus.

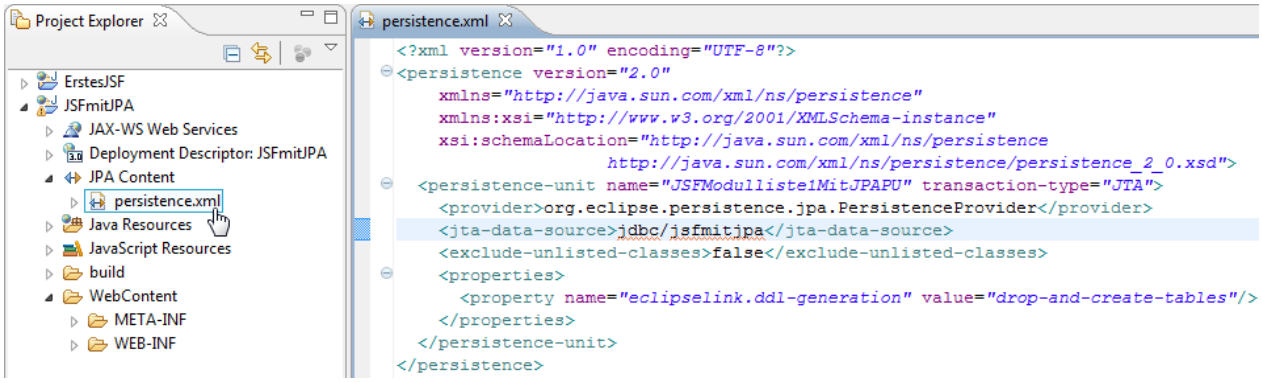
```
package entities;
```

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Version;

@Entity
public class Modul implements Serializable {

  private static final long serialVersionUID = 2L;
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
  //@Column(unique=true)
  private int nr; // so sind gleiche Nummern erlaubt
  private String name;
  @Version int version;

  public Modul() {
  }

  public Modul(int nr, String name) {
    this.nr = nr;
    this.name = name;
  }

  ///*
  public int getVersion() {
    return version;
  }

  public void setVersion(int version) {
    this.version = version;
  }
//*/

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getNr() {
    return nr;
  }
```

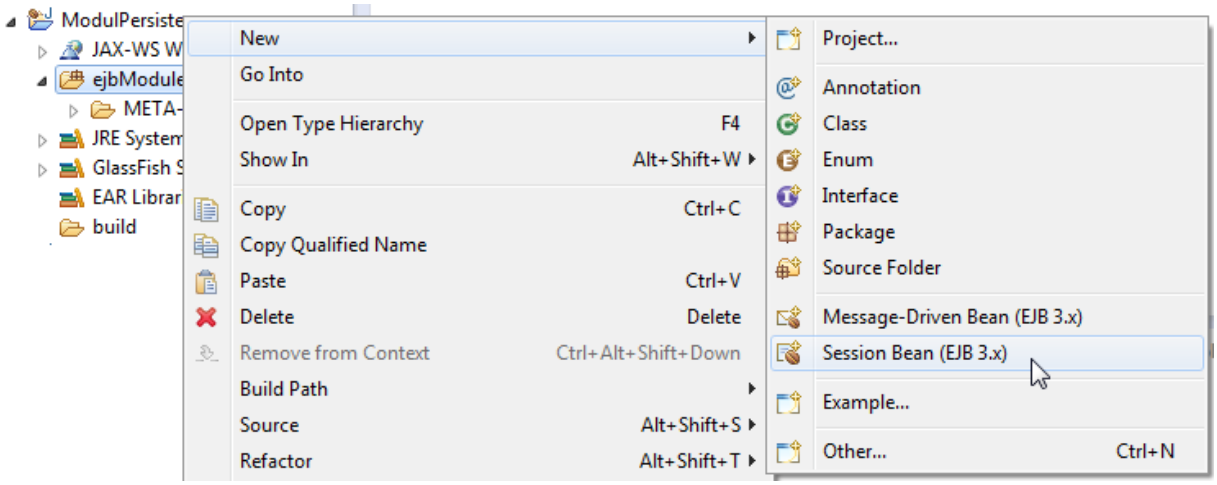
```
  public void setNr(int nr) {
    this.nr = nr;
  }

  @Override
  public int hashCode() {
    int hash = 0;
    hash += (int) id;
    return hash;
  }

  @Override
  public boolean equals(Object object) {
   // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof Modul)) {
      return false;
    }
    Modul other = (Modul) object;
    if (this.id != other.id) {
      return false;
    }
    return true;
  }

  @Override
  public String toString() {
    return "entities.Moudul[id=" + id + "]";
  }
}
```

Nun wird im Projektbaum auf der linken Seite „JPA Content“ aufgeklappt und persistence.xml geöffnet. Hier muss jetzt die im vorherigen Kapitel erstellte JNDI-Verbindung eingegeben werden. Eine einfache Möglichkeit besteht darin, folgende Einträge zu nutzen. Die Einstellung ist für die Entwicklung sinnvoll, da am Start immer wieder eine neue leere Datenbank angelegt wird.

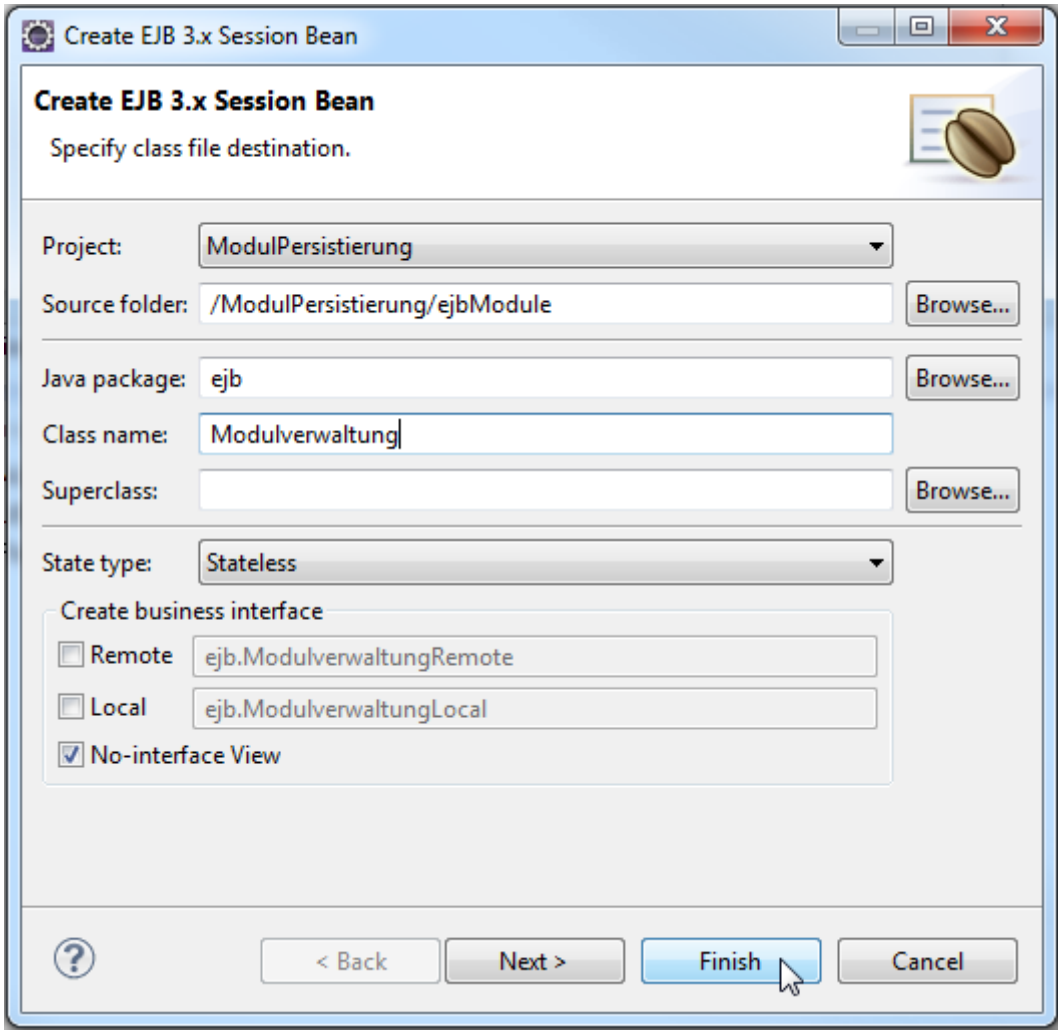
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
         xmlns="http://xmlns.jcp.org/xml/ns/persistence"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
         http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
 <persistence-unit name="JSFModulliste1MitJPAPU" transaction-type="JTA">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <jta-data-source>jdbc/jsfmitjpa</jta-data-source>
  <exclude-unlisted-classes>false</exclude-unlisted-classes>
  <properties>
   <property name="eclipselink.ddl-generation" value="drop-and-create-tables"/>
  </properties>
 </persistence-unit>
</persistence>
```



Die eigentliche Verbindung zur Datenbank wird über eine Stateless-SessionBean verwaltet. Zur Erzeugung wird ein Rechtsklick auf „ejbModule“ gemacht und „New > Session Bean (EJB 3.x)“ ausgewählt.



Außer dem „Java package“ und dem „Class name“ ist nichts zu ändern.

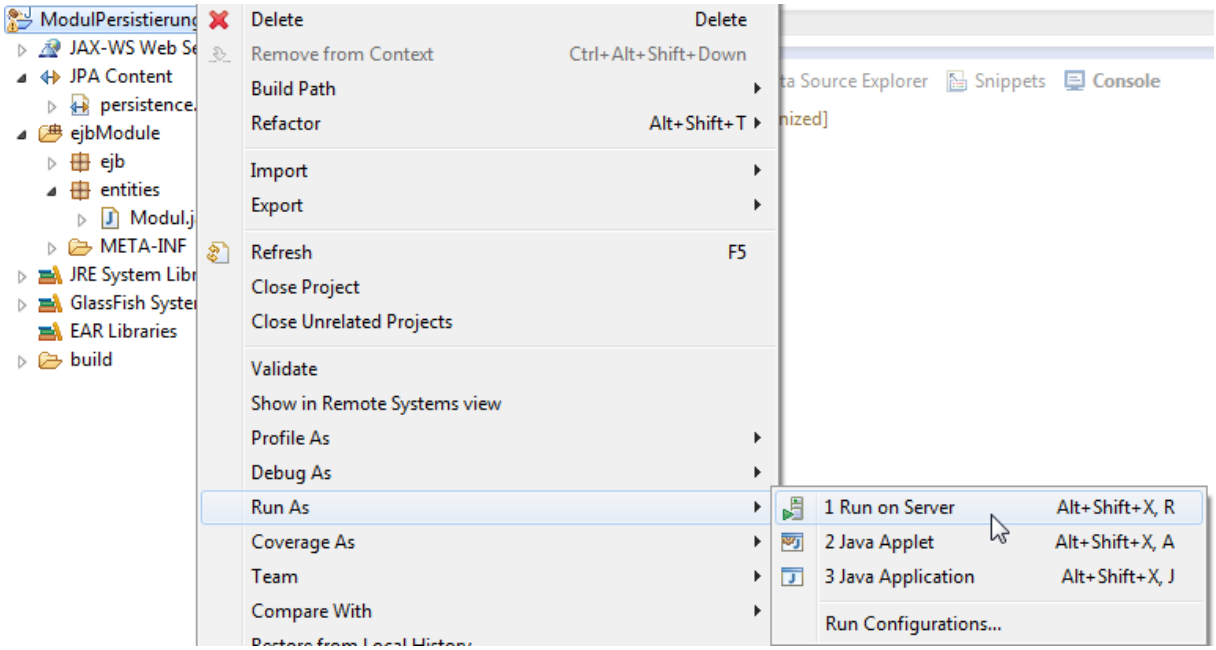


Der folgende Code realisiert die EJB.

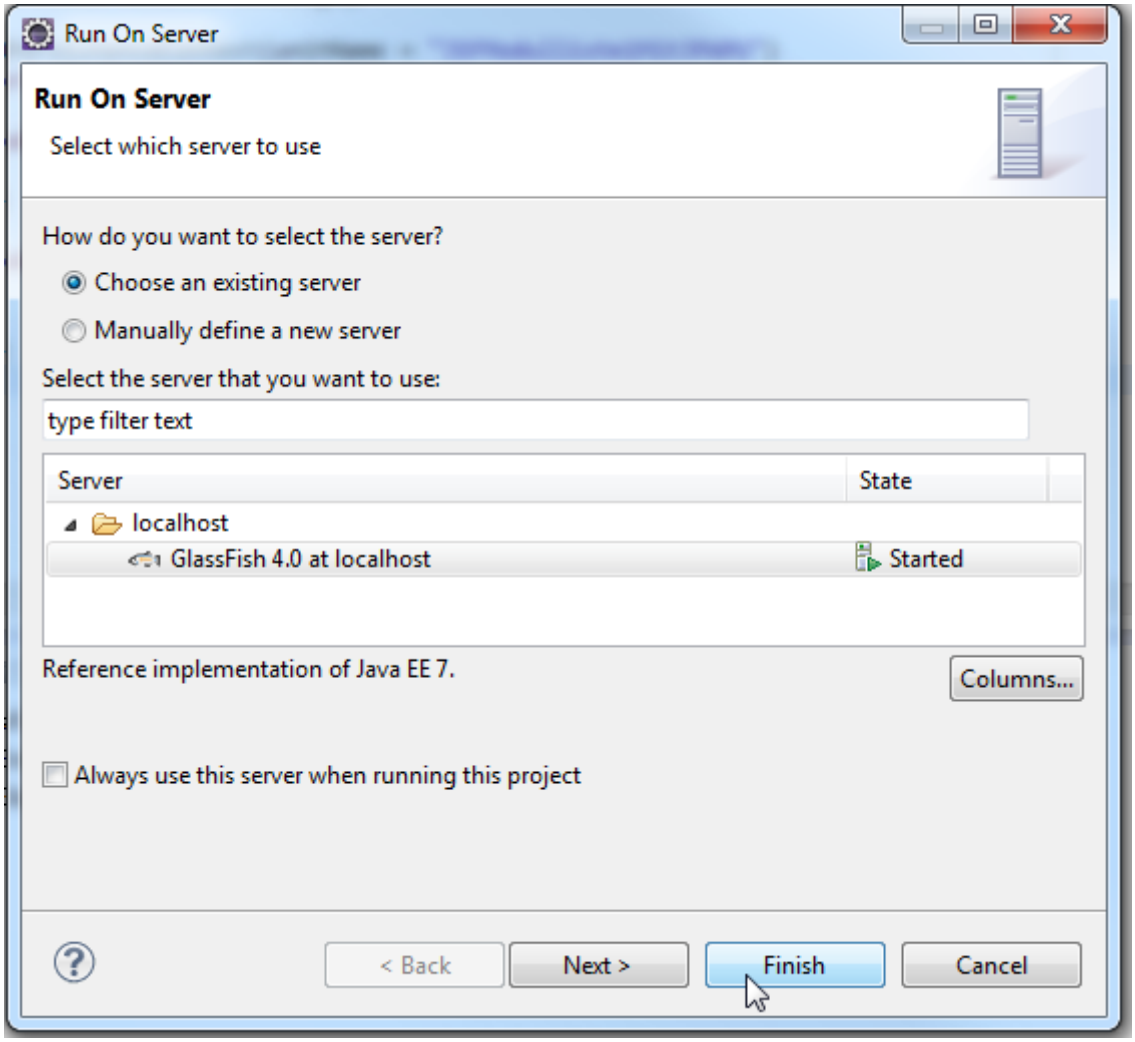
```
package ejb;

import entities.Modul;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class Modulverwaltung {

    @PersistenceContext(unitName = "vlJSFModullisteMitEJBPU")
    private EntityManager em;

    public void persist(Object object) {
        em.persist(object);
    }

    public List<Modul> getModule() {
```

```
        return em.createQuery("SELECT m FROM Modul m", Modul.class)
                .getResultList();
    }
}
```

Man kann die EJB auch auf dem Server laufen lassen, wobei dies ohne weitere Tests wenige Erkenntnisse bringt. Dazu wird das Projekt mit einem Rechtsklick angeklickt und „Run As > Run on Server“ ausgewählt.

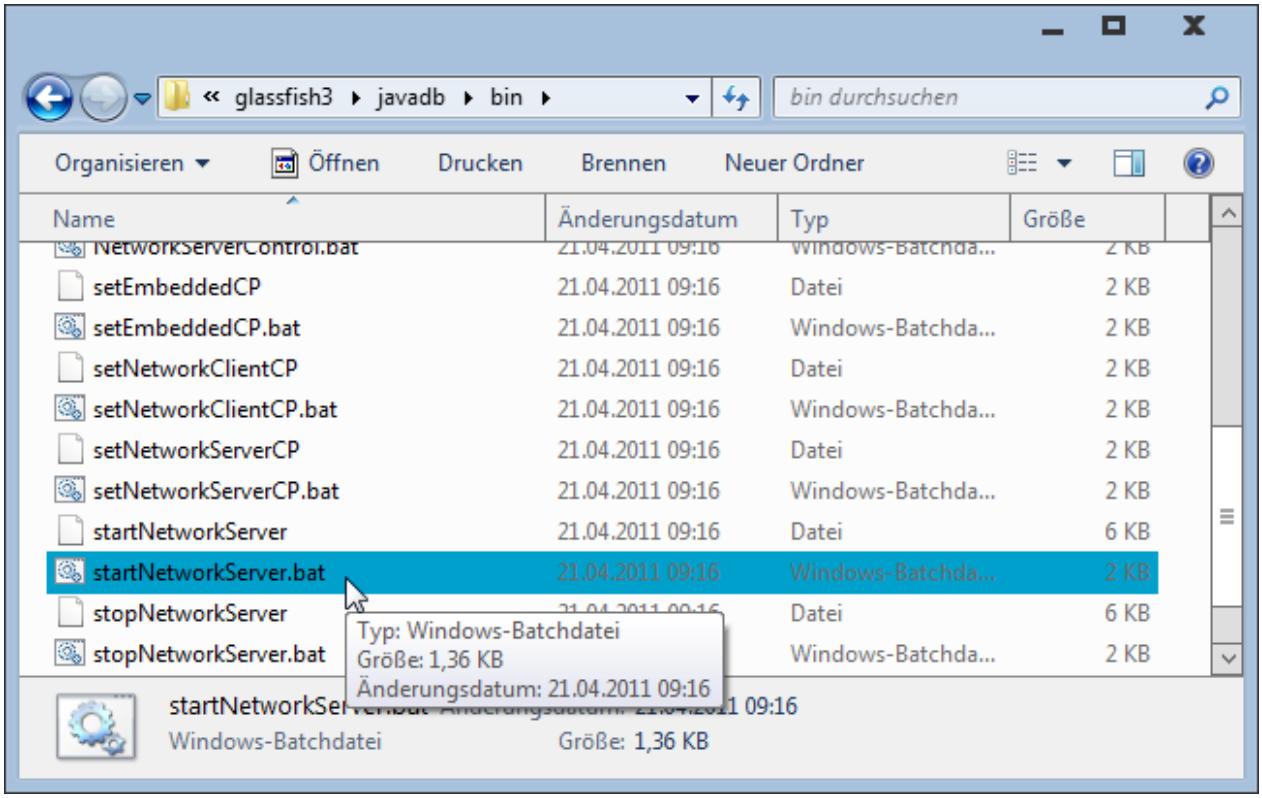


Beim ersten Start muss man den Server noch auswählen. Im konkreten Fall wird einfach „Finish“ geklickt.

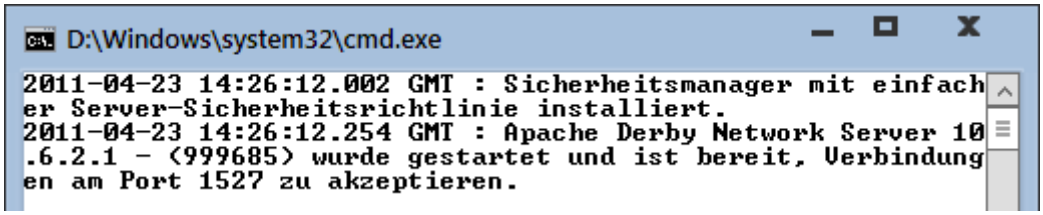


20.6 Nutzung von JSF mit EJB in Eclipse JEE

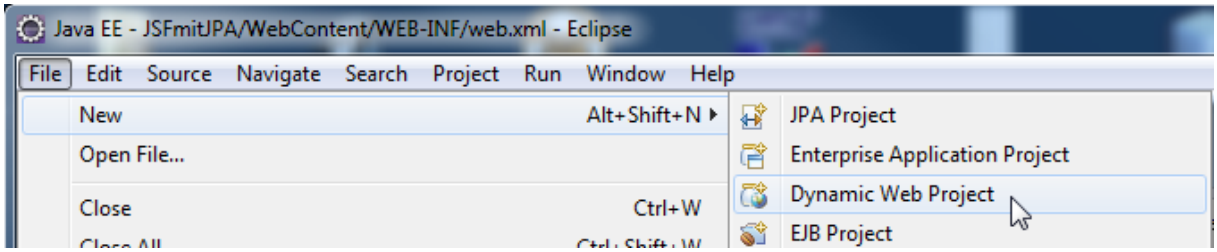
Zunächst wird der Datenbankserver gestartet, dies geschieht über die Start-Skripts im javadb/bin Unterordner im Ordner glassfish.



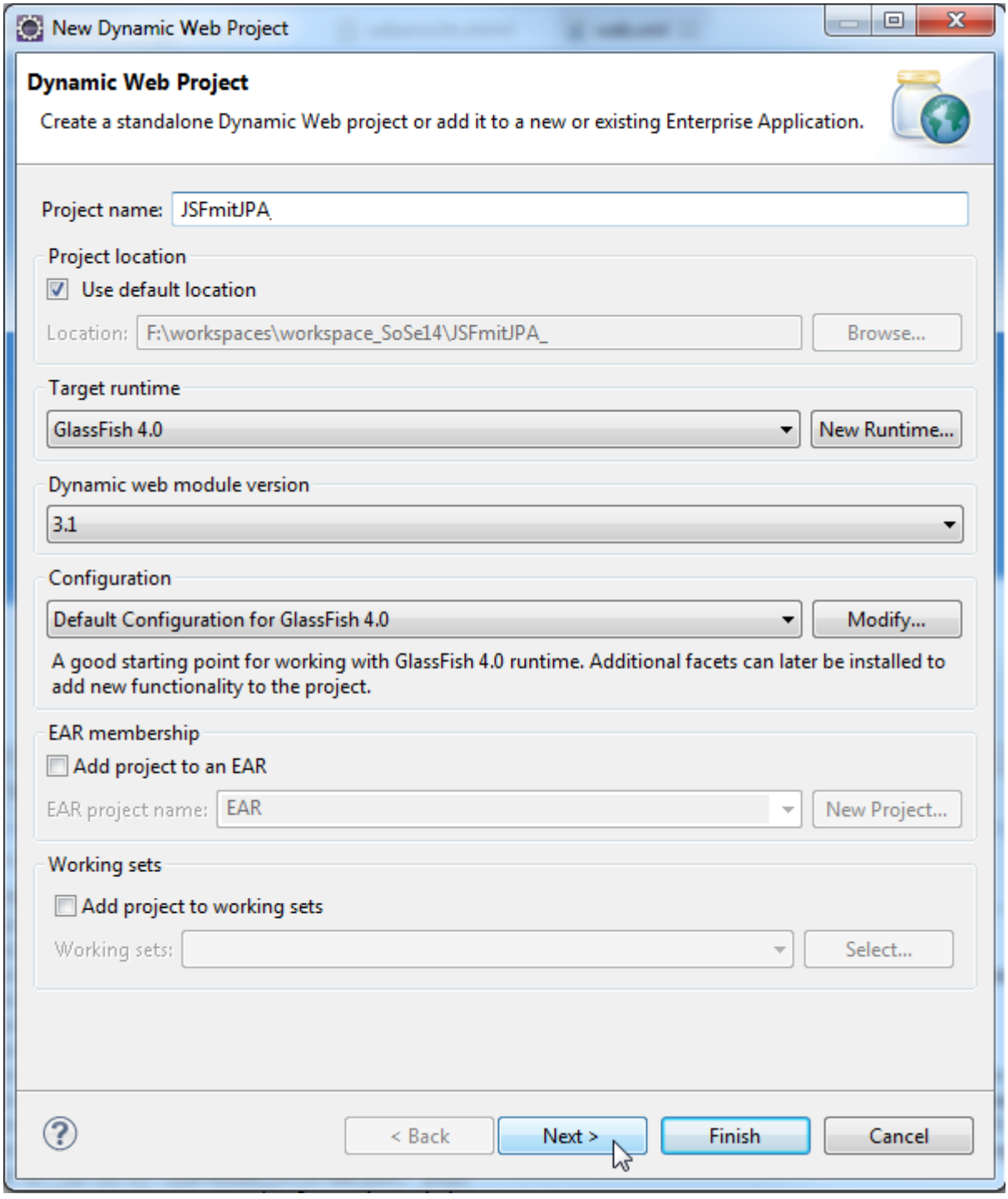
Das sich öffnende Fenster darf minimiert, aber nicht geschlossen werden.



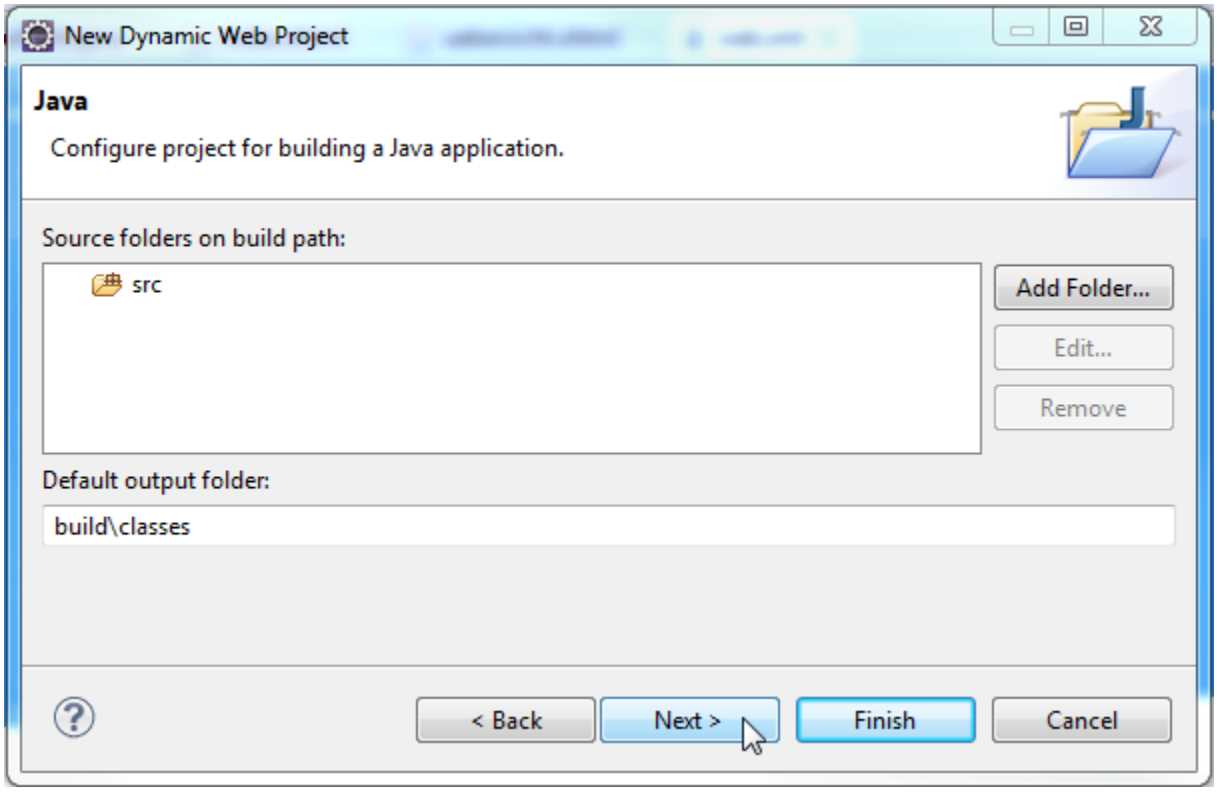
Nun wird ein Dynamic Web Project, hier „JSFmitJPA“, angelegt und „Next>“ geklickt



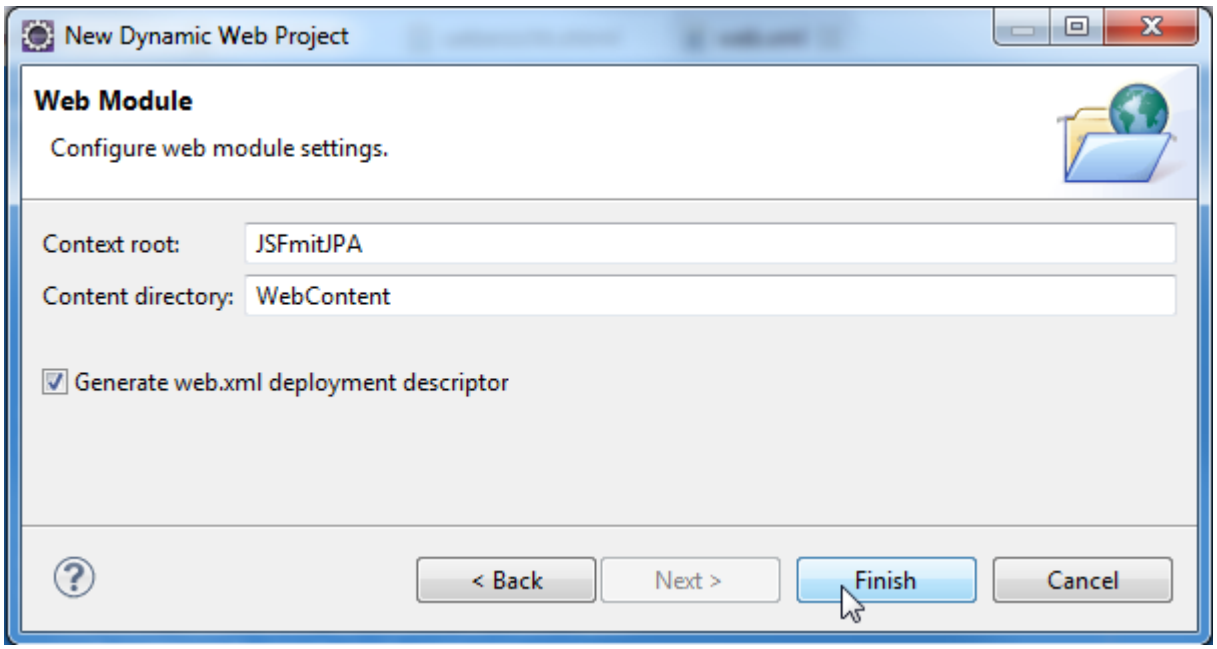
.



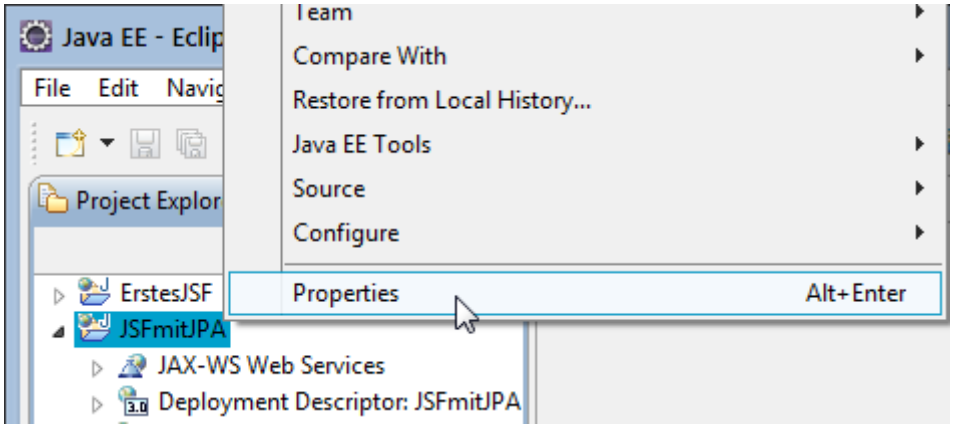
Es wird wieder „Next>“ geklickt.



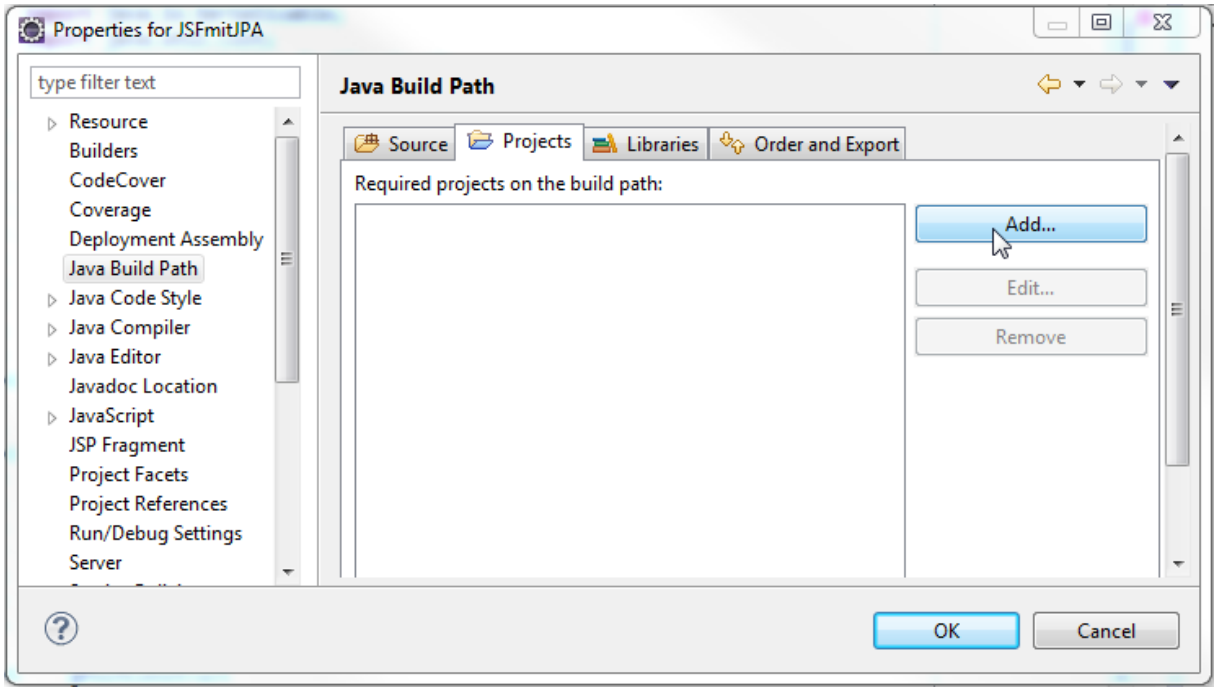
Hier wird der Haken bei „Generate web.xml deployment descriptor“ hesetzt und „Finish“ geklickt.



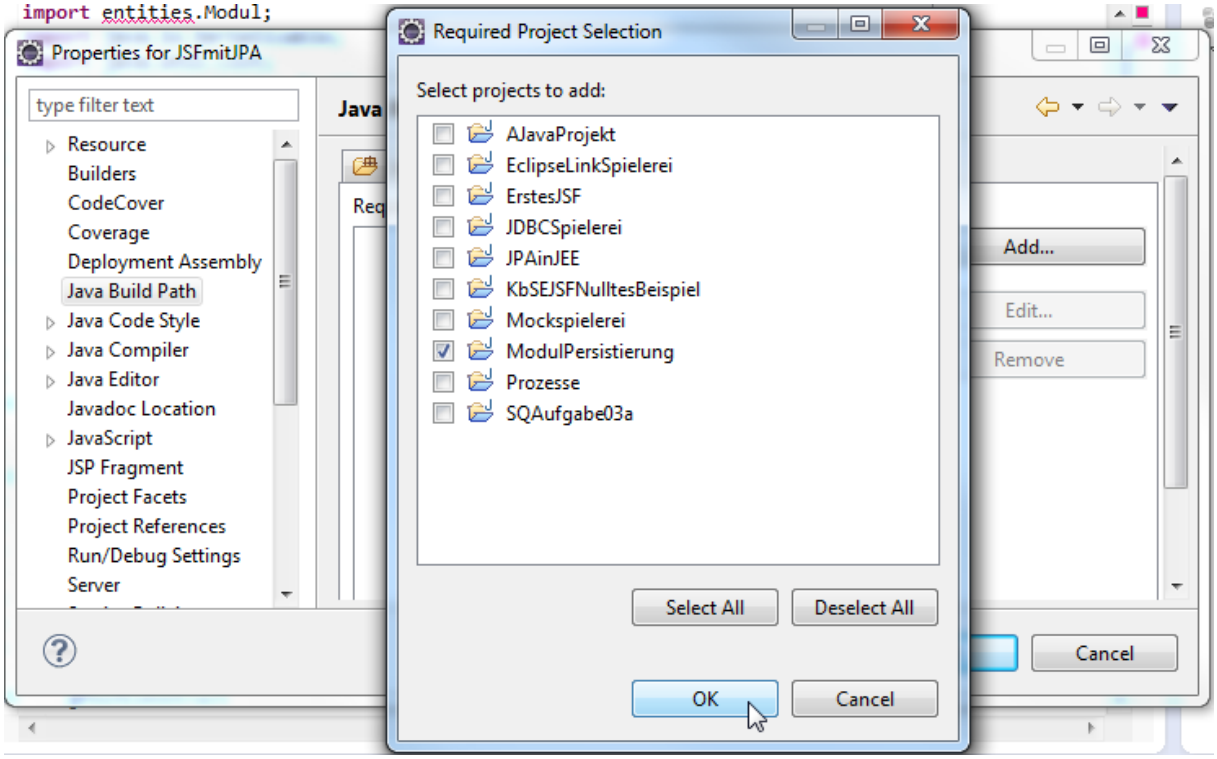
Danach wird ein Rechtklick auf dem Projekt gemacht und „Properties“ gewählt.



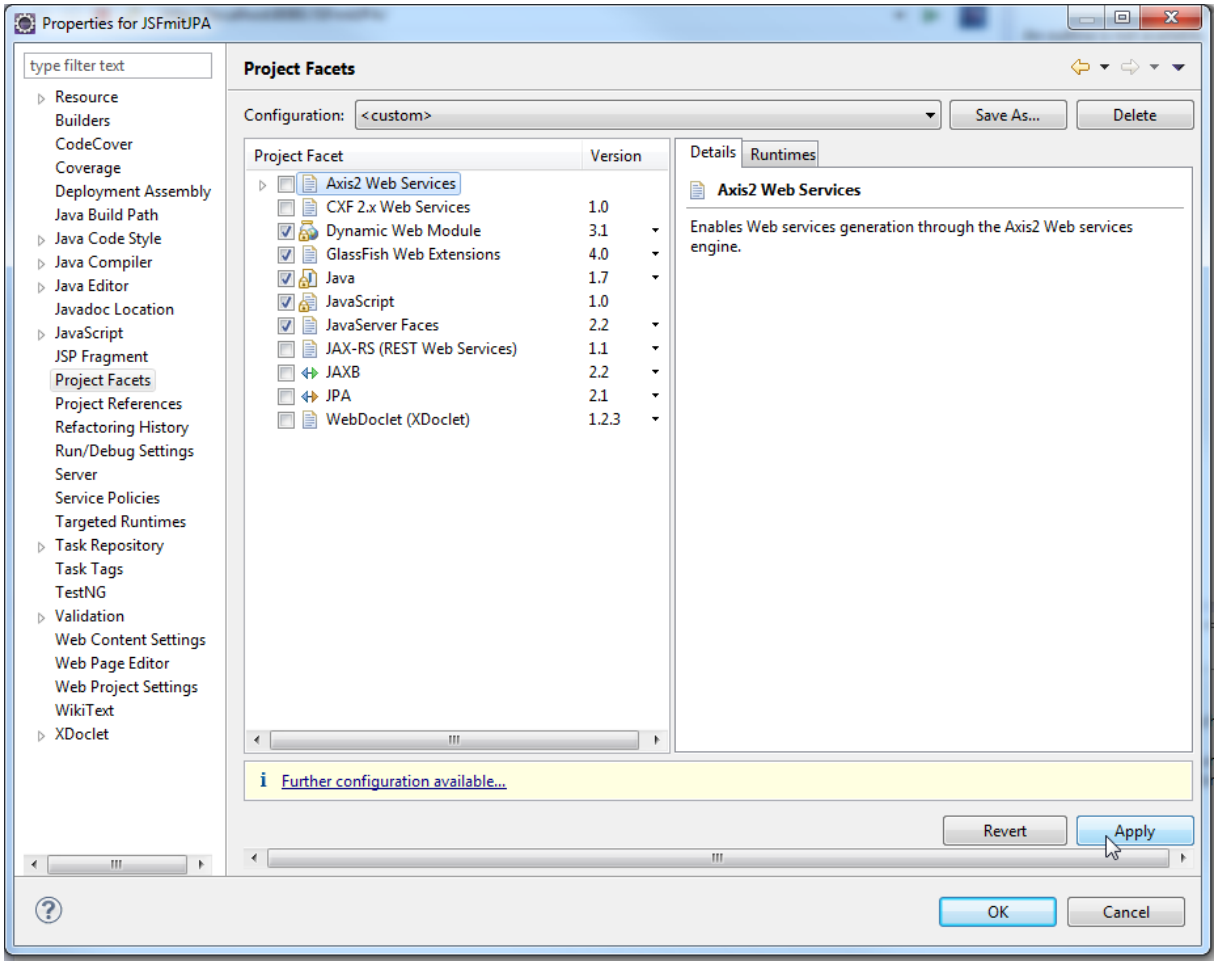
Auf der linken Seite wird „Java Build Path“ angeklickt und links „Add…“ geklickt.



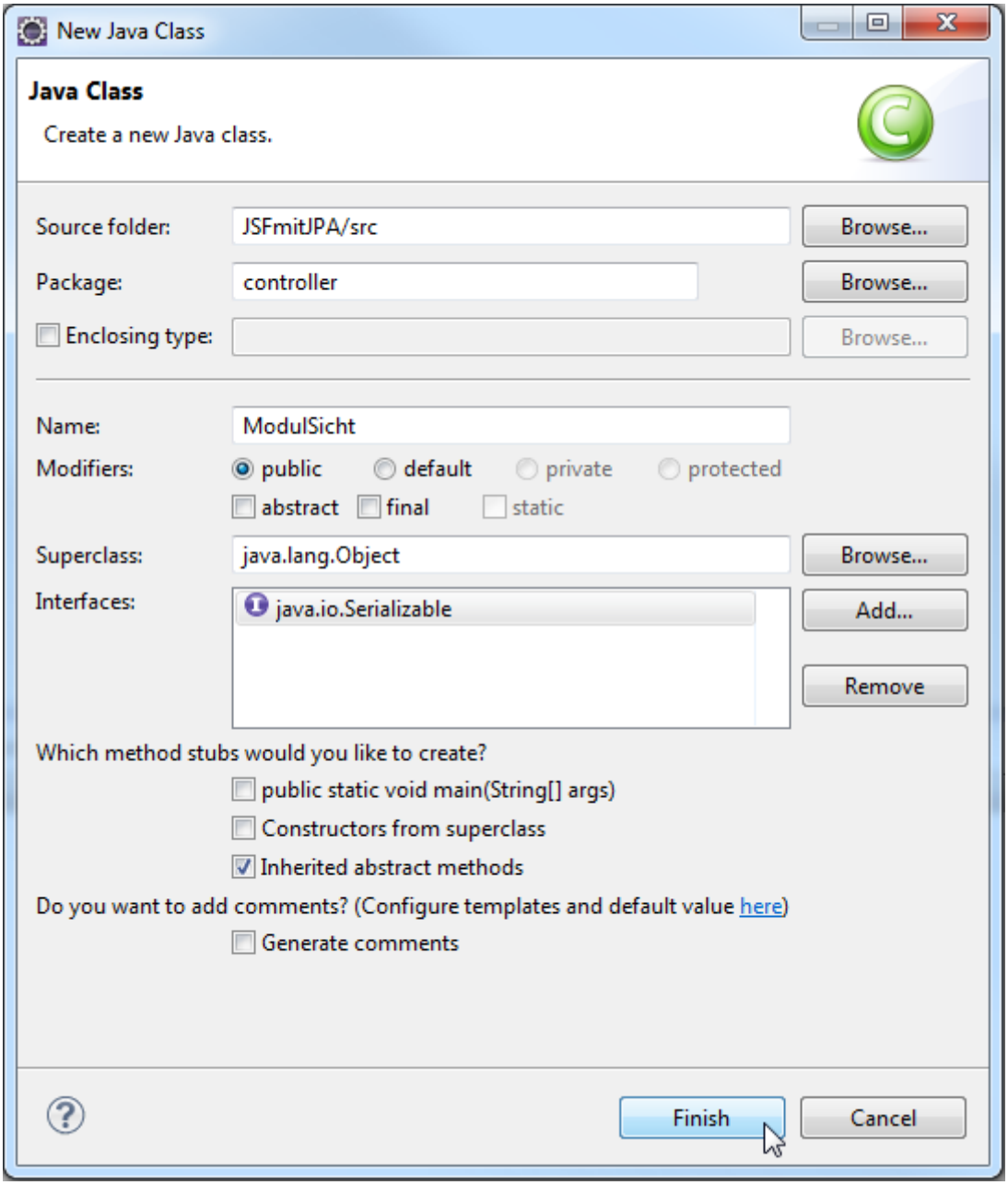
Es wird das im vorherigen Unterkapitel erstellte Projekt ModulPersistierung ausgewählt und alles mit „OK“ bestätigt.



Weiterhin wird „Project Facets“ angeklickt und „JavaServer Faces“ ausgewählt und mit „Apply“ und „OK“ bestätigt.



Nun wird ein Seiten-Controller zur Anzeige der Entitäts-Objekte geschrieben, der als ManagedBean realisiert wird.



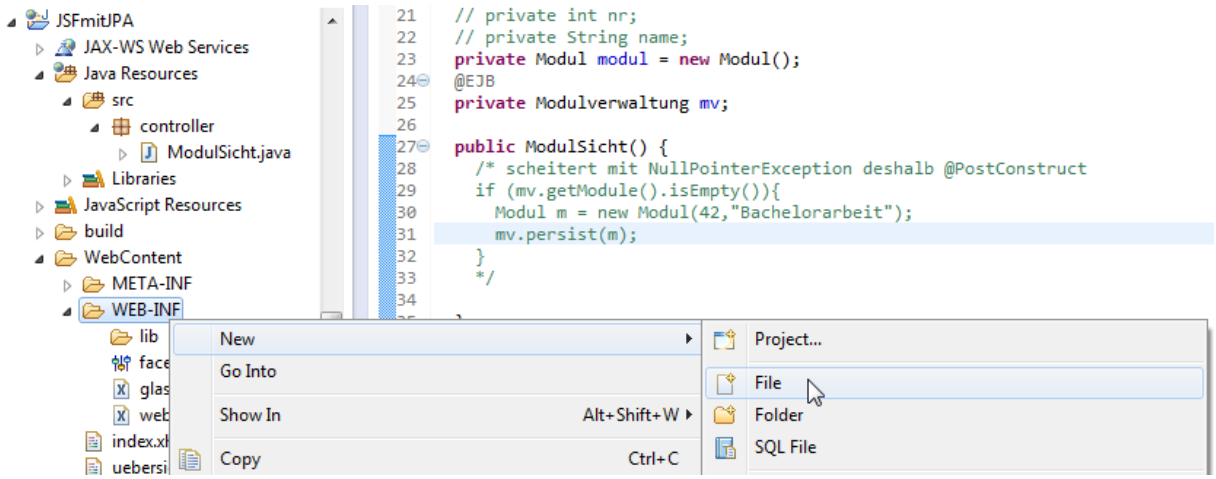
Die Klasse hat und folgenden Inhalt.

```
package controller;

import ejb.Modulverwaltung;
import entities.Modul;
import java.io.Serializable;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.ejb.EJB;
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named("module")
@RequestScoped
```

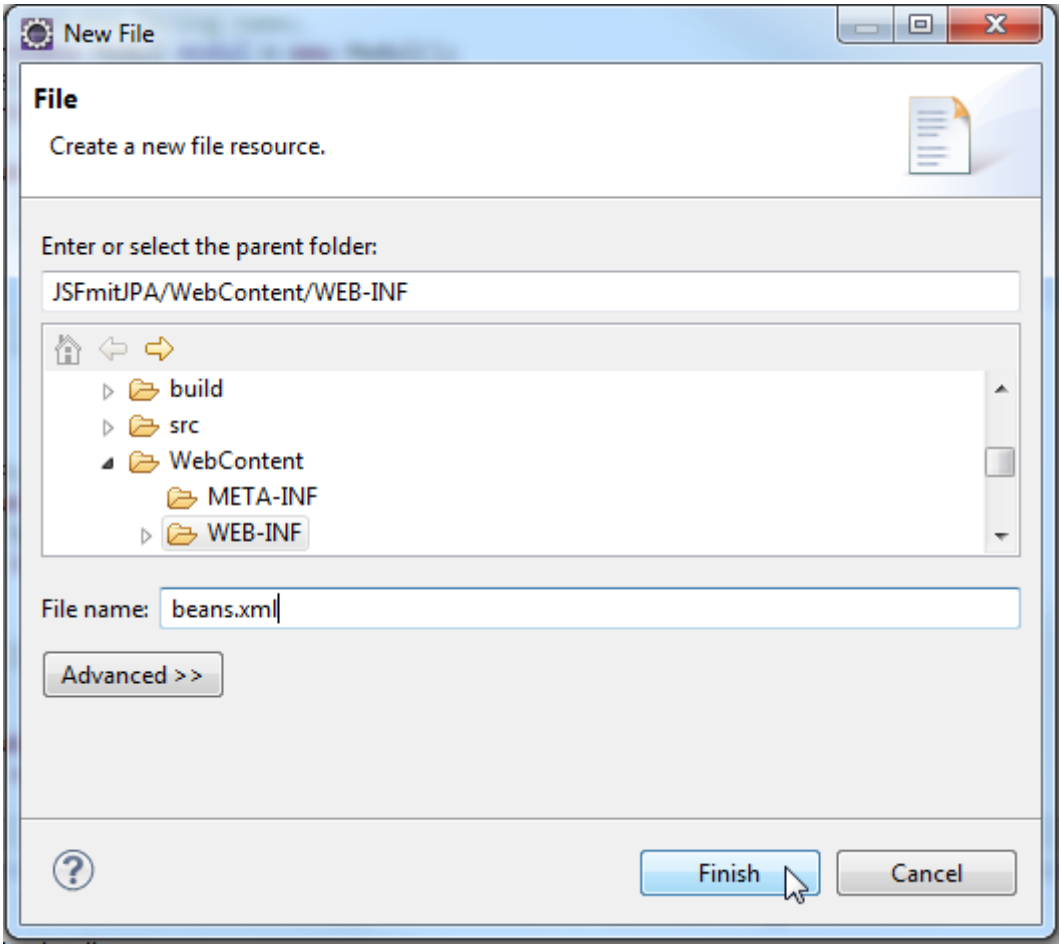
```
public class ModulSicht implements Serializable {

    private static final long serialVersionUID = 1L;
    //private int nr;
    //private String name;
    private Modul modul = new Modul();
    @EJB
    private Modulverwaltung mv;

    public ModulSicht() {
        /* scheitert mit NullPointerException deshalb @PostConstruct
         if (mv.getModule().isEmpty()){
           Modul m = new Modul(42,"Bachelorarbeit");
           mv.persist(m);
         }
         */
    }

    @PostConstruct
    public void modulGarantieren() {
        if (mv.getModule().isEmpty()) {
            Modul m = new Modul(42, "Bachelorarbeit");
            mv.persist(m);
        }
    }

    public List<Modul> getModule() {
        return mv.getModule();
    }

    public String uebernehmen() {
        try {
            mv.persist(modul);
            // modul = new Modul() nicht notwendig
        } catch (Exception e) {
        }
        return "./uebersicht.xhtml";
    }

    public String eingeben() {
        return "./index.xhtml";
    }

    public String anzeigen() {
        return "./uebersicht.xhtml";
    }

    public Modul getModul() {
        return modul;
    }

    public void setModul(Modul modul) {
        this.modul = modul;
    }

    @Override
    public int hashCode() {
```

```
        int hash = 3;
        hash = 67 * hash + (this.modul != null ? this.modul.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final ModulSicht other = (ModulSicht) obj;
        if (this.modul != other.modul && (this.modul == null
                                          || !this.modul.equals(other.modul))) {
            return false;
        }
        return true;
    }
}
```

Damit mit Hilfe von CDI die ManagedBean, genauer die mit @Named annotierte Bean gefunden wird, muss im Verzeichnis WEB-INF eine Datei beans.xml angelegt werden. Man kann dies über die Erstellung einer XML-Datei erledigen oder direkt die Datei, wie hier gezeigt erstellen. Nach einem Rechtsklick auf dem Verzeichnis WEB-INF wird „New > File“ ausgewählt.



Der Name muss beans.xml lauten.



Theoretisch kann die Datei leer sein, sinnvoll ist aber der folgende Inhalt.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       bean-discovery-mode="all">

</beans>
```

Bei Problemen sollte das „annotated“ in der letzten Zeile durch „all“ ersetzt werden.

Danach werden im Ordner WebContent zwei Seiten zur Eingabe von Modulen und zur Ausgabe aller Module erstellt. Die Datei index.xhtml hat folgenden Inhalt.

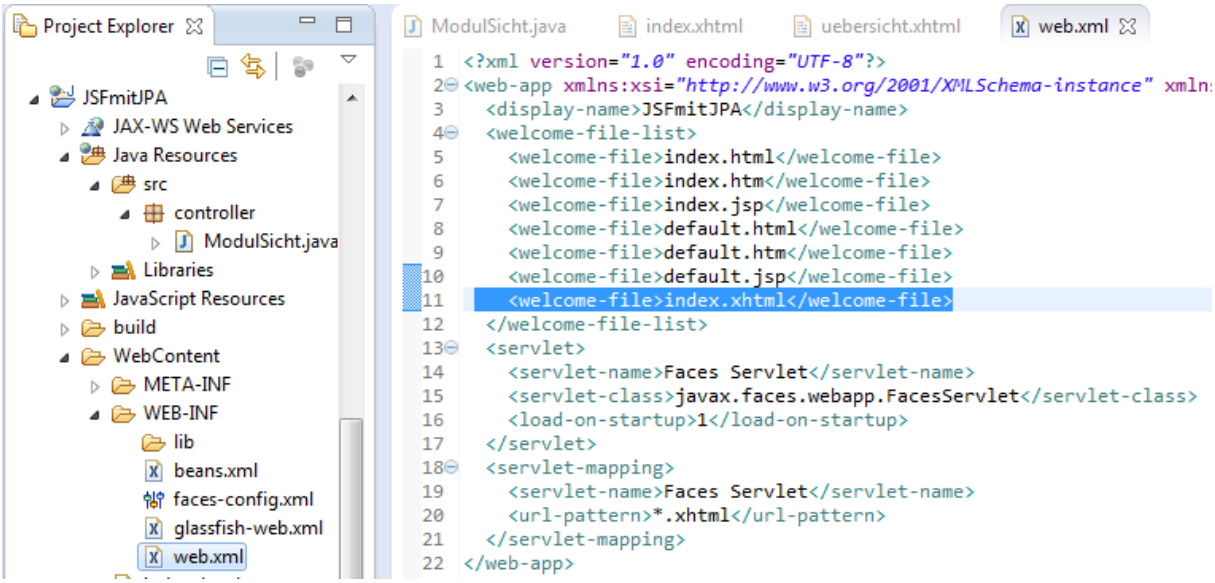
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Moduleingabe</title>
    </h:head>
    <h:body>
      <h:form id="f1">
        <h:messages globalOnly="true"/>
```

```
        <h:panelGrid columns="3" >
          <h:outputLabel for="mname" value="Modulname "/>
          <h:inputText id="mname" value="#{module.modul.name}"/>
          <h:message for="mname" />
          <h:outputLabel for="mnr" value="Modulnummer "/>
          <h:inputText id="mnr" value="#{module.modul.nr}"/>
          <h:message for="mnr"/>
          <h:commandButton value="Abschicken" action="#{module.uebernehmen}"/>
        </h:panelGrid>
        <h:commandLink action="#{module.anzeigen}" >
          <h:outputText value="Zur Modulübersicht"/>
        </h:commandLink>
      </h:form>
    </h:body>
</html>
```

Die Datei uebersicht.xhtml hat folgenden Inhalt.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <h:head>
    <title>Modulübersicht</title>
  </h:head>
  <h:body>
    <h:form id="f2">
      <h:messages globalOnly="true"/>
      <h:panelGrid rendered="#{!empty module.module}">
        <h:dataTable value="#{module.module}" var="m" border="8" frame="box" >
          <h:column >
            <f:facet name="header">
              <h:outputText value="Nummer" />
            </f:facet>
            <h:outputLabel value="#{m.nr}"/>
          </h:column>
          <h:column>
            <f:facet name="header">
              <h:outputText value="Modulname" />
            </f:facet>
            <h:outputLabel value="#{m.name}"/>
          </h:column>
        </h:dataTable>
      </h:panelGrid>
      <h:commandLink action="#{module.eingeben}" >
        <h:outputText value="Zur Moduleingabe"/>
      </h:commandLink>
    </h:form>
  </h:body>
</html>
```

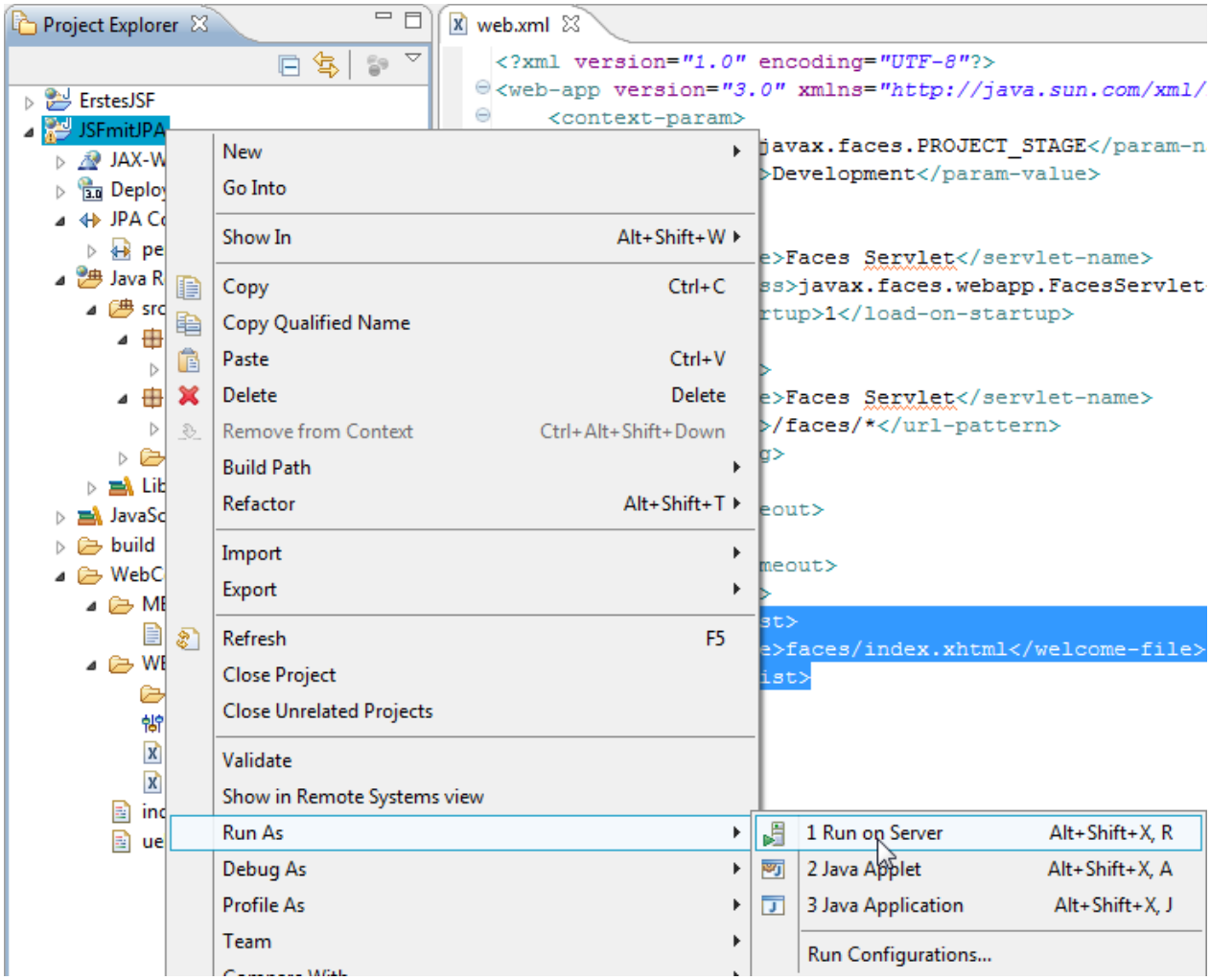
Abschließend muss in der Datei web.xml im Ordner WEB-INF noch die Startseite eingetragen werden. Weiterhin zeigt die folgende Abbildung auf der linken Seite den gesamten Projektaufbau.



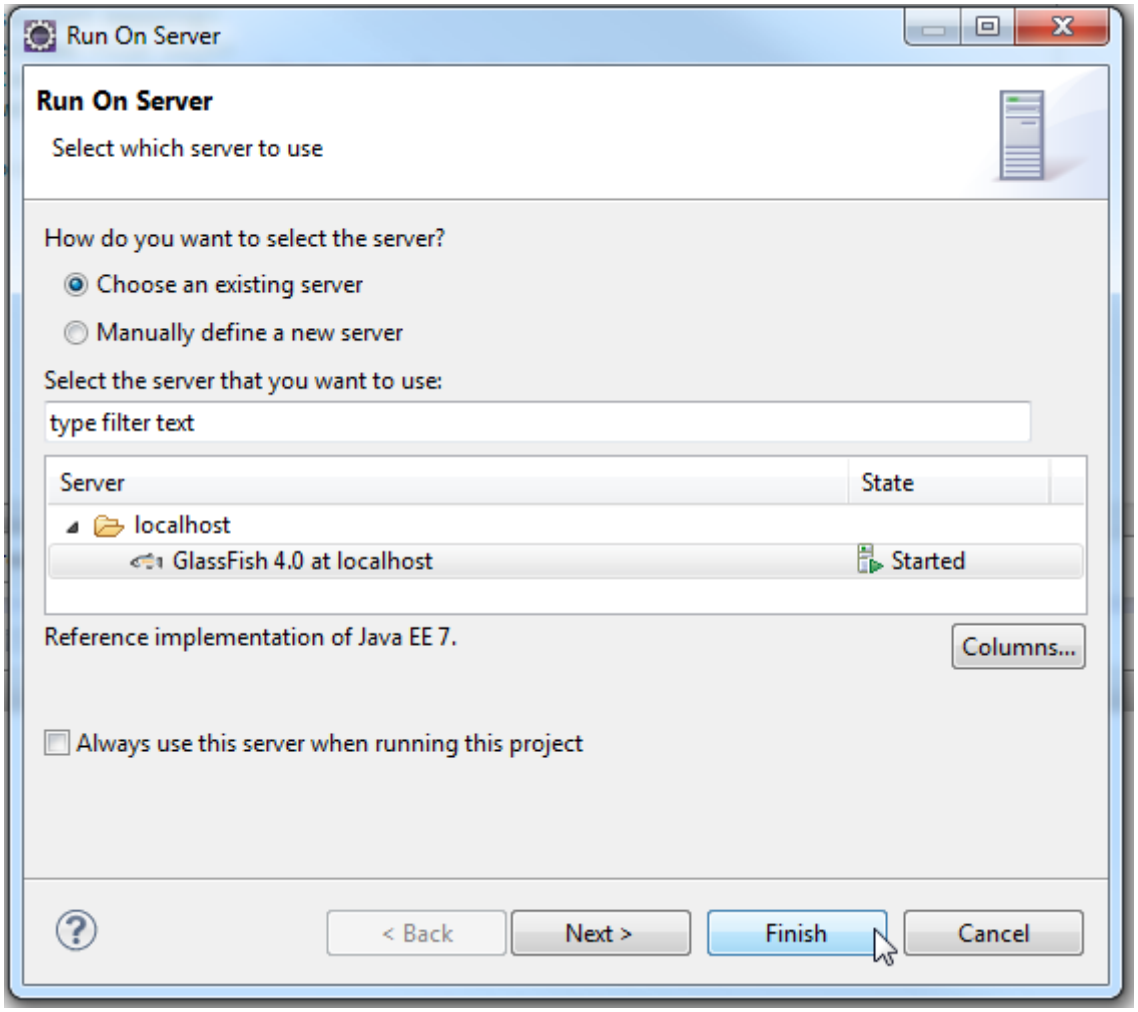
Falls web.xml nicht generiert wurde, sieht der Inhalt wie folgt aus.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>JSFmitJPA</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>index.xhtml</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>
</web-app>
```

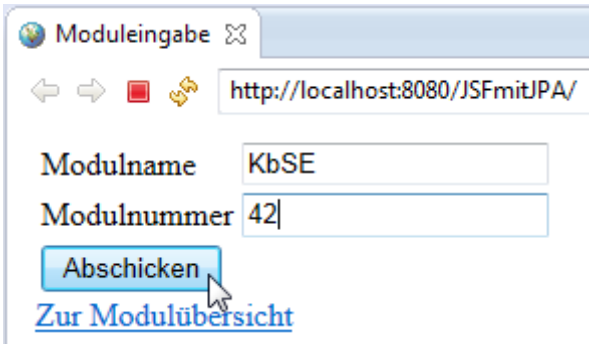
Die Applikation kann dann über einen Rechtsklick auf dem Projekt mit „Run As“ und „Run on Server“ gestartet werden.

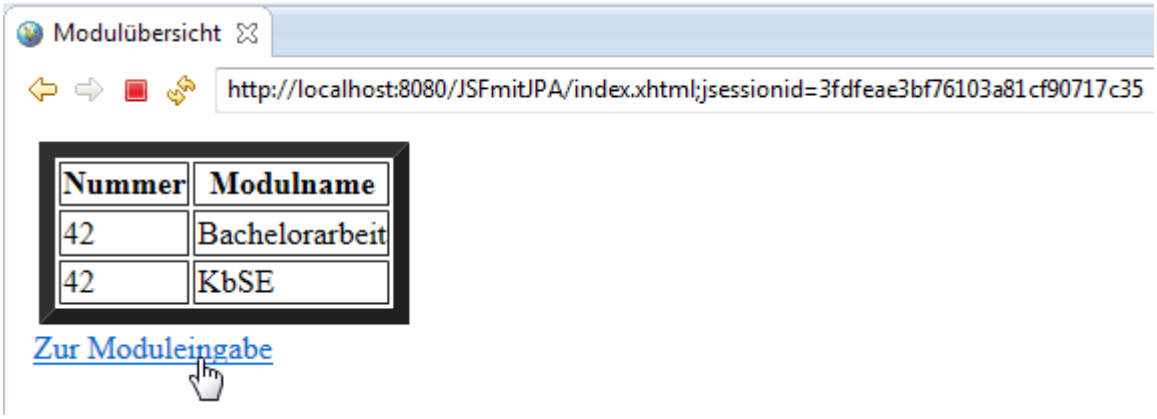


Die folgende Konfigurationsseite wird mit „Finish“ geschlossen.

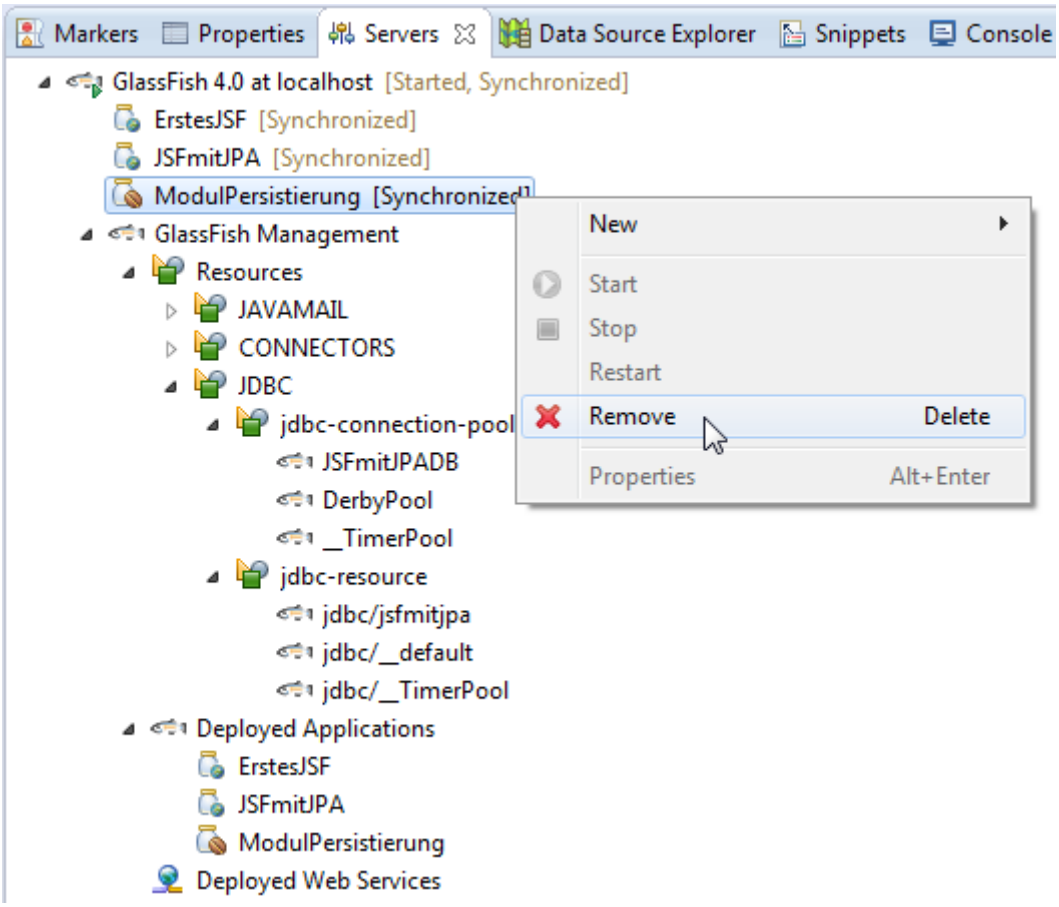


Folgende Seitenfolge ist dann möglich.

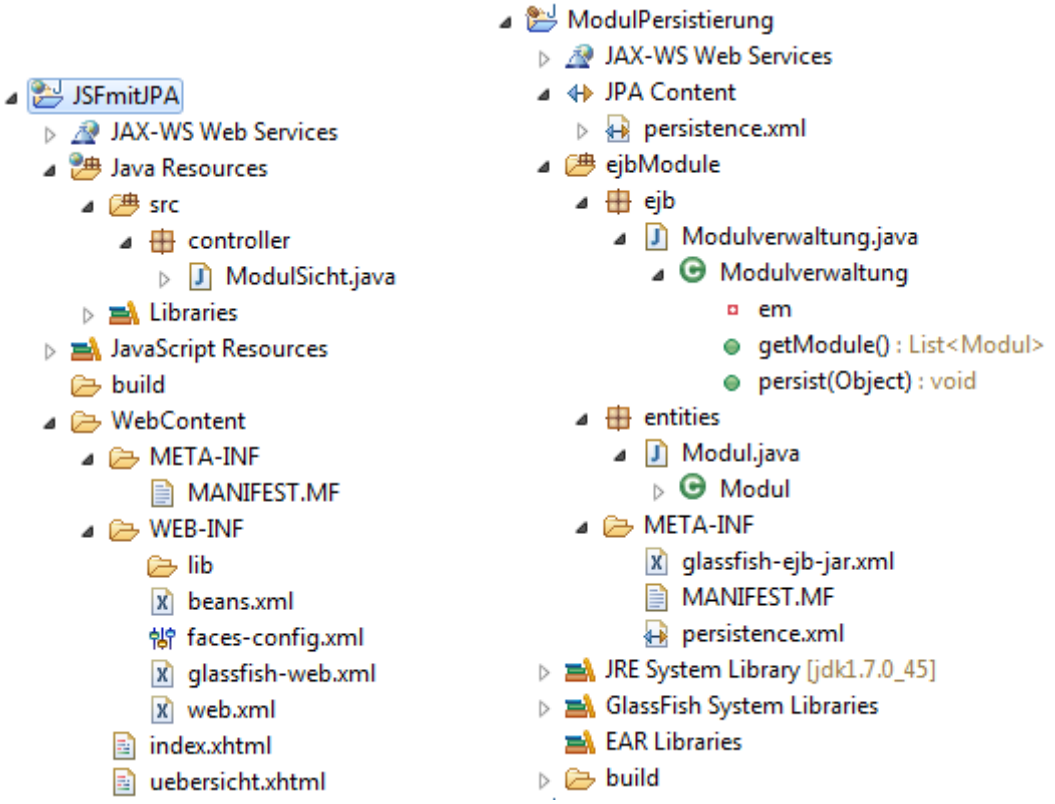




Im Reiter „Servers“ sieht man die laufenden Applikationen, die man auch über einen Rechtsklick entfernen kann.

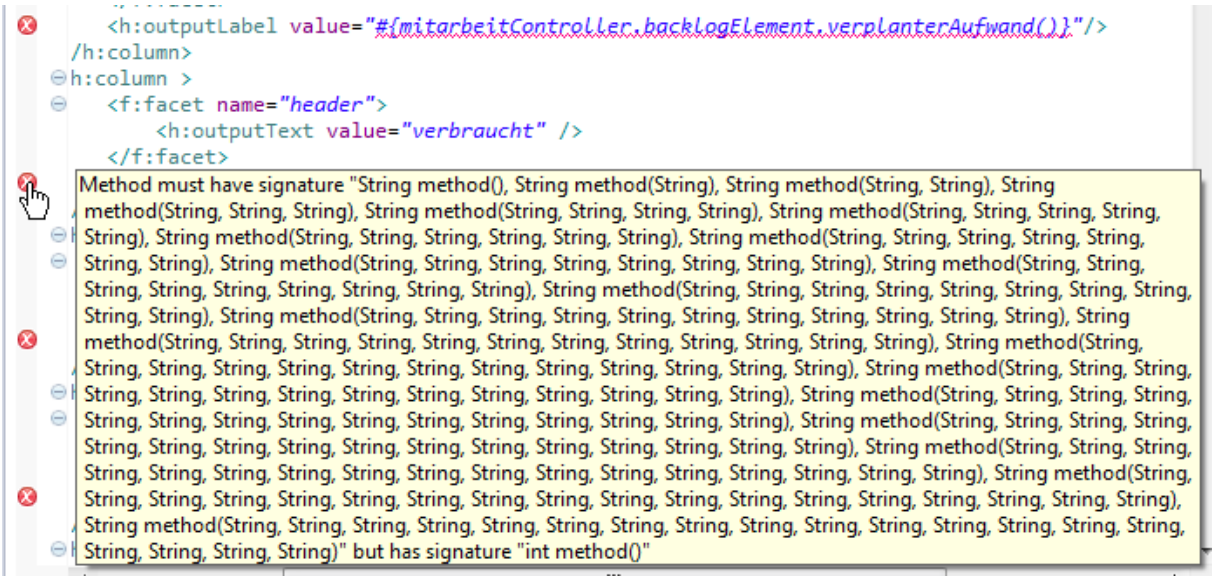


Das gesamten Projekte haben folgenden Aufbau.

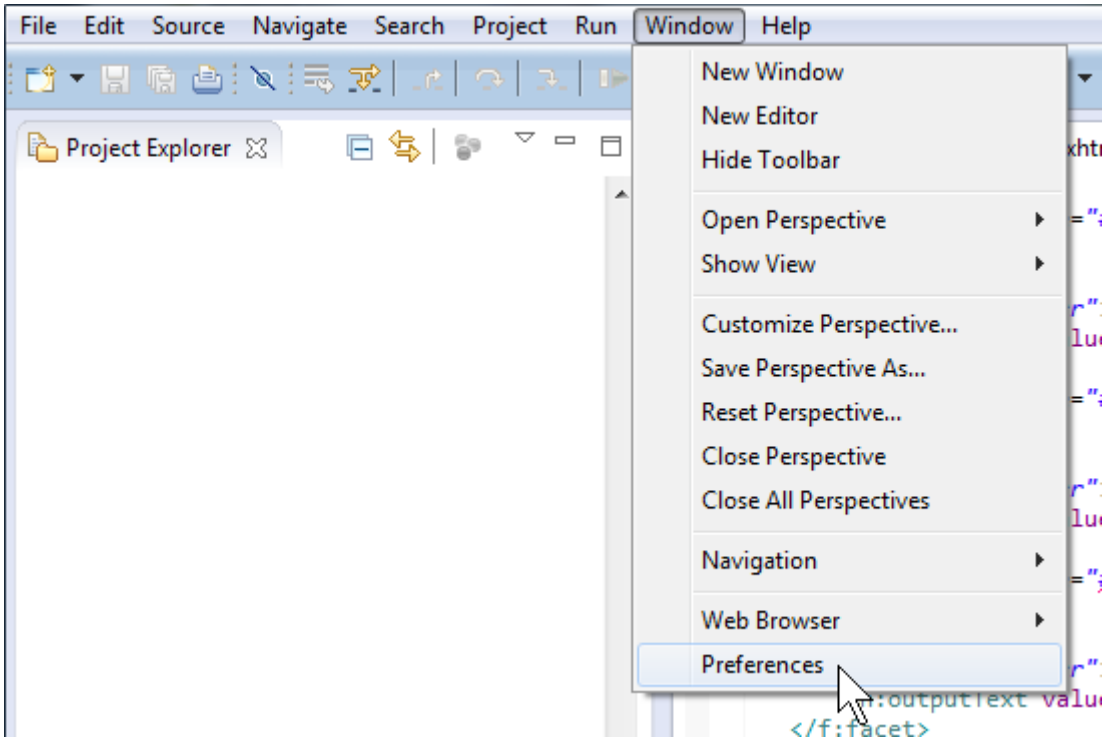


20.7 Probleme in Eclipse mit der Expression Language

In XHTML-Seiten werden ab und zu Fehler bei Ausdrücken in der Unified Expression Language (EL) angezeigt, die keine sind und auch z. B. von NetBeans nicht als solche angezeigt werden. Die folgende Abbildung zeigt zwei dieser Probleme nacheinander und die erschreckend sinnlose Fehlermeldung, da ein Rückgabewert vom Typ int natürlich erlaubt ist.

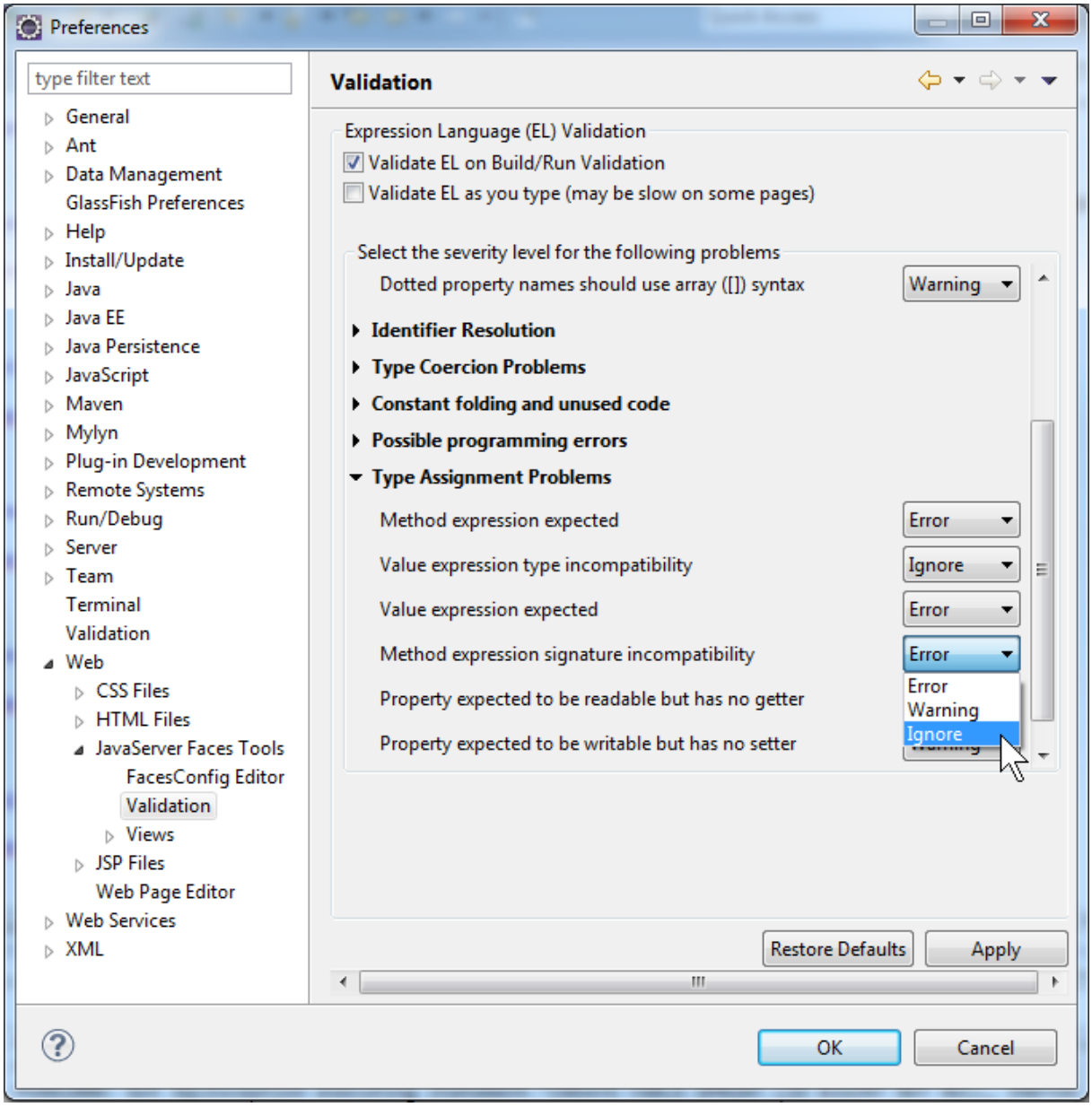


Der einzig passende Workaround ist, diesen Teil der Kompatibilitätsprüfung auszuschalten. Dazu wählt man oben „Window > Preferences“.

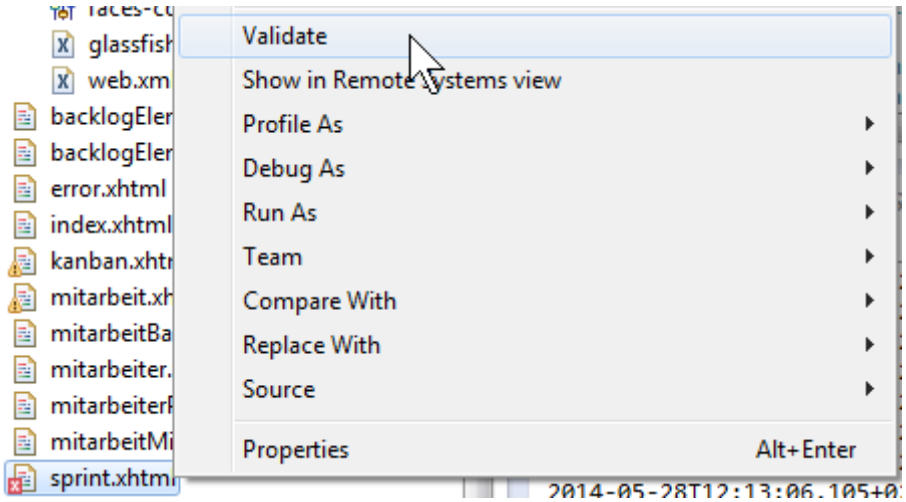


Dann wird links „Web“ und dach „JavaServerFaces Tools“ aufgeklappt und „Validation“ ausgewählt. Dann wird rechts das Feld „Type Assignment Problems“ audgeklappt und der Wert bei „Method expression signature incompatibility“ auf „Igore“ gesetzt und die Schritte „OK“ abgeschlossen.

Bei verwandten Problemen kann es sinnvoll sein, weitere Validierungsregeln auszuschalten.



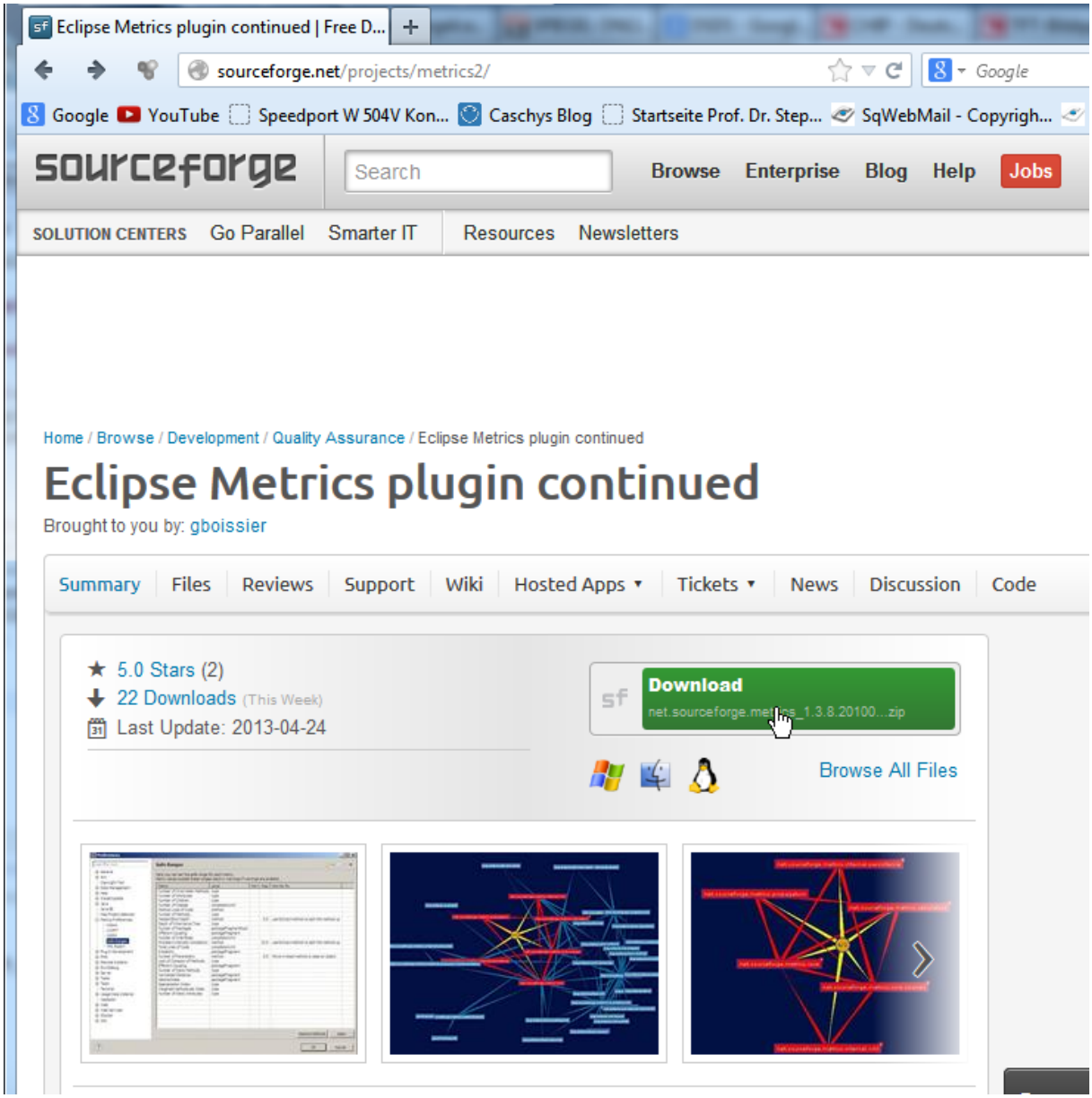
Man beachte, dass man die Validierung auch von Hand starten kann. Dazu wird ein Rechtsklick auf der Datei gemacht und dann „Validate“ ausgewählt.


glassfish
web.xm
backlogEler
backlogEler
error.xhtml
index.xhtml
kanban.xhtr
mitarbeit.xh
mitarbeitBa
mitarbeiter.
mitarbeiterF
mitarbeitMi
sprint.xhtml
Validate
Show in Remote Systems view
Profile As
Debug As
Run As
Team
Compare With
Replace With
Source
Properties
Alt+Enter

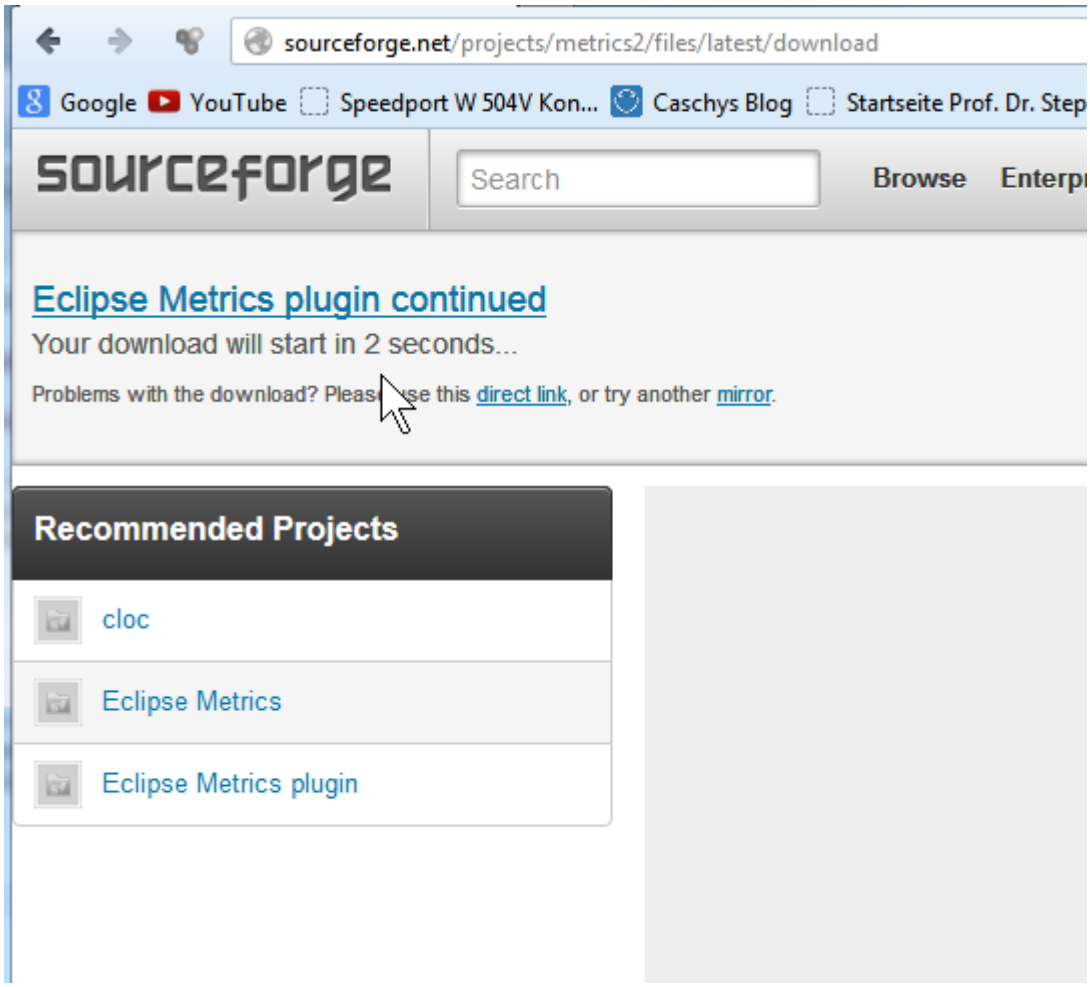
21 Metrics2 (4.11.0)

Mit Metrics 2 besteht die Möglichkeit sehr unterschiedliche Metriken für Java-Quellcode zu berechnen. Die bekannteste ist die zyklomatische Zahl nach McCabe mit der die Anzahl von Verzweigungen für einzelne Methoden und durchschnittlich für ganze Projekte berechnet werden kann. Dieser Wert ist ein guter Indikator für die Lesbarkeit des Programms; je niedriger der Wert desto besser. Wie für alle Metriken gilt, dass sie hilfreiche Indikatoren sein können, aber niemals eine echte Qualitätssicherung ersetzen. Da die Berechnung der Metriken aber kaum Zeit in Anspruch nimmt, stellen sie immer einen Mehrwert im Werkzeugkasten der QS-Werkzeuge dar.

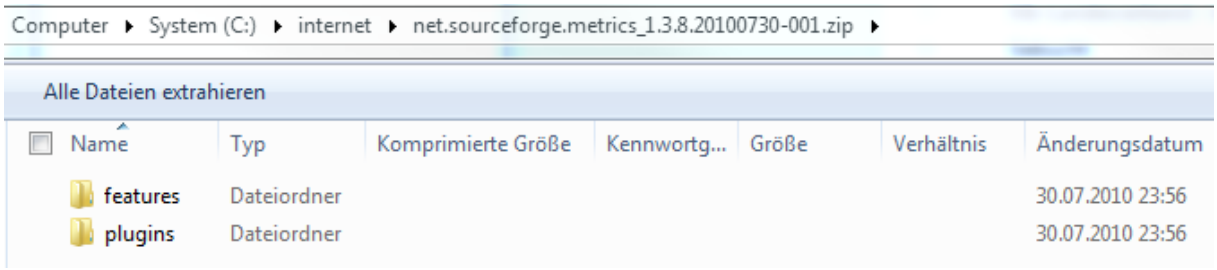
Metrics 2 kann von der Webseite http://sourceforge.net/projects/metrics2/ mit einem Klick auf das Download-Feld geladen werden, der Start des Downloads kann etwas dauern.



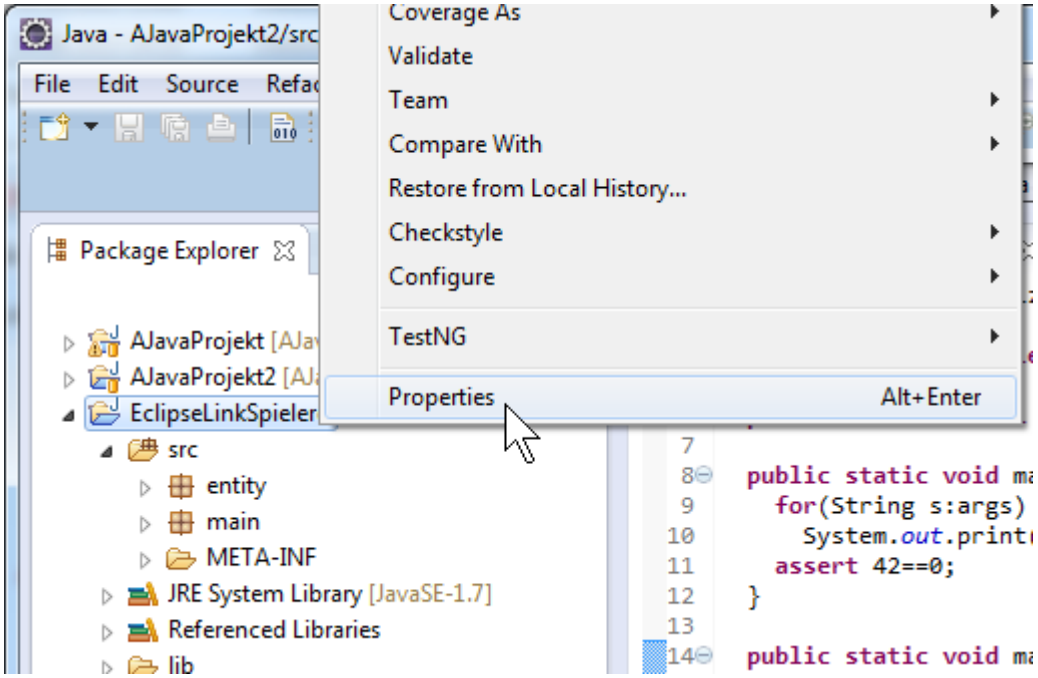
Es dauert etwas, danach startet der Download.



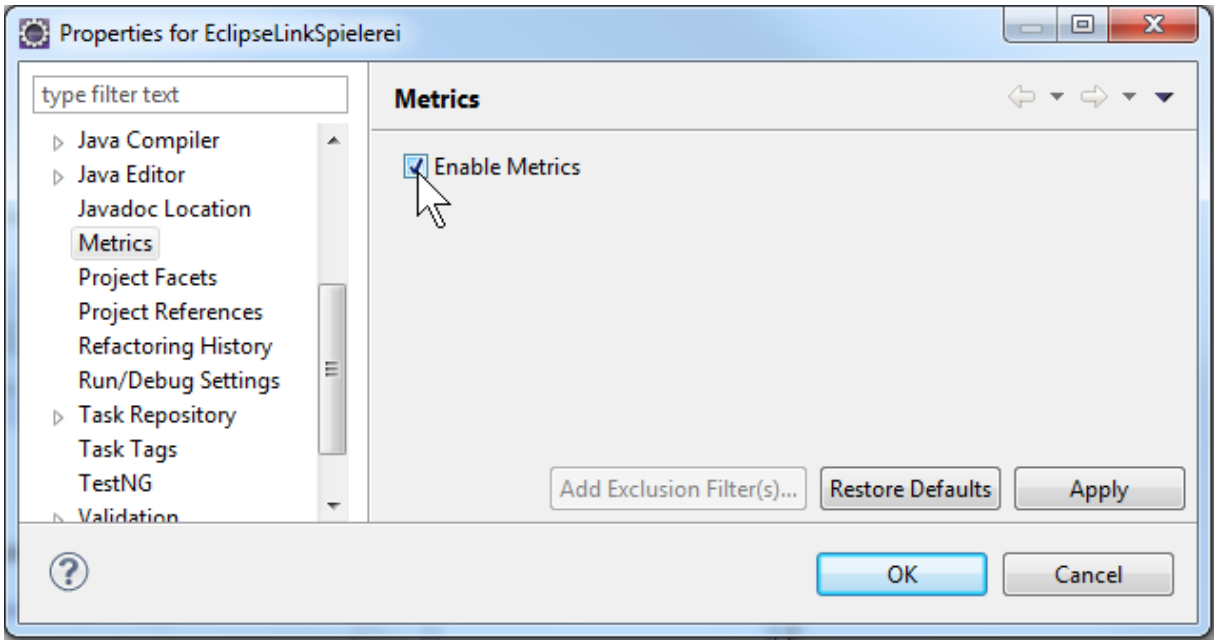
Die Installation erfolgt durch Auspacken. Man beachte, dass das Zip-Verzeichnis zwei Ordner features und plugins enthält deren Inhalt in die jeweiligen Eclipse-Ordner kopiert werden müssen.



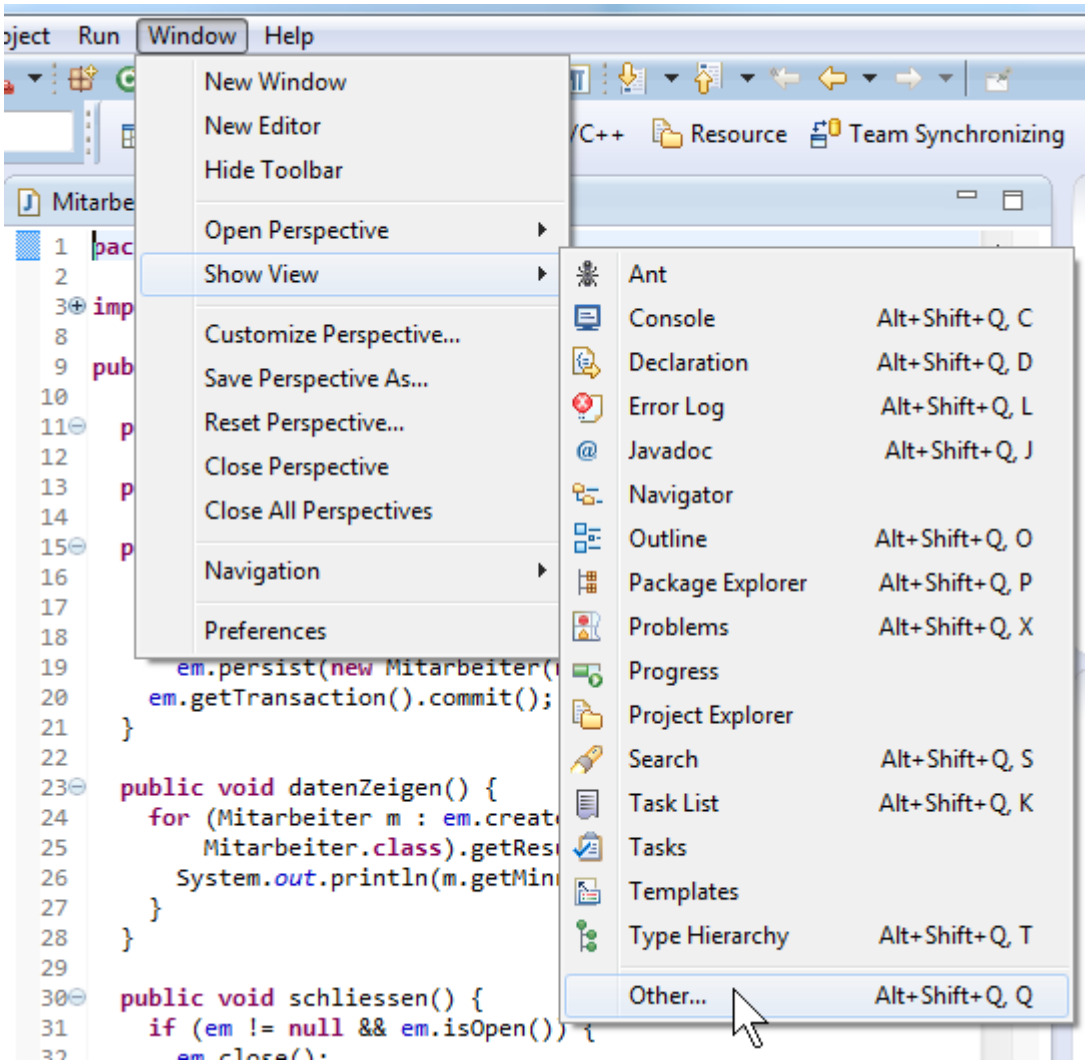
In Eclipse muss die Metrik-Berechnung aktiviert werden. Nach einem Rechtsklick auf dem Projekt wird „Properties“ gewählt.



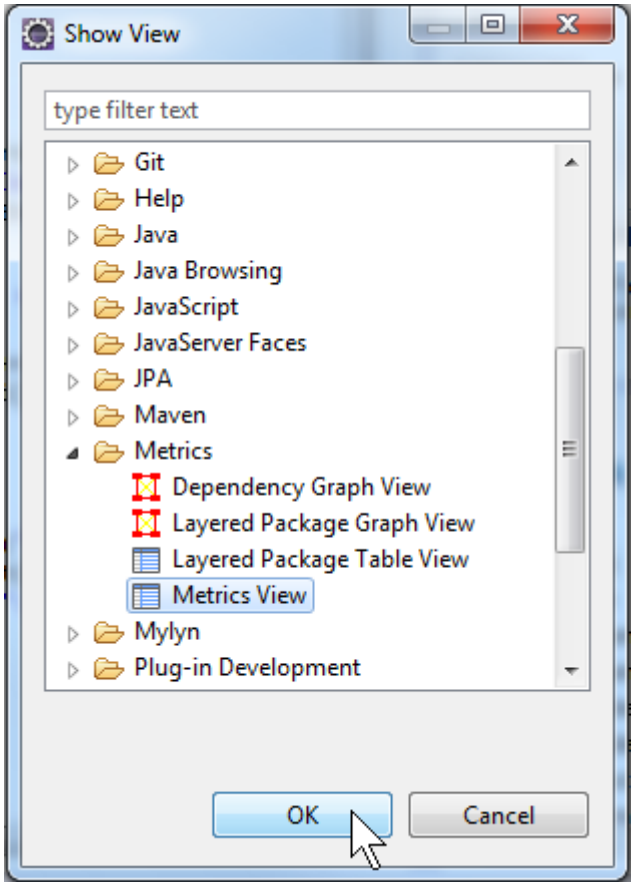
Unter dem Punkt „Metrics“ wird der Haken zum „enablen“ gesetzt.



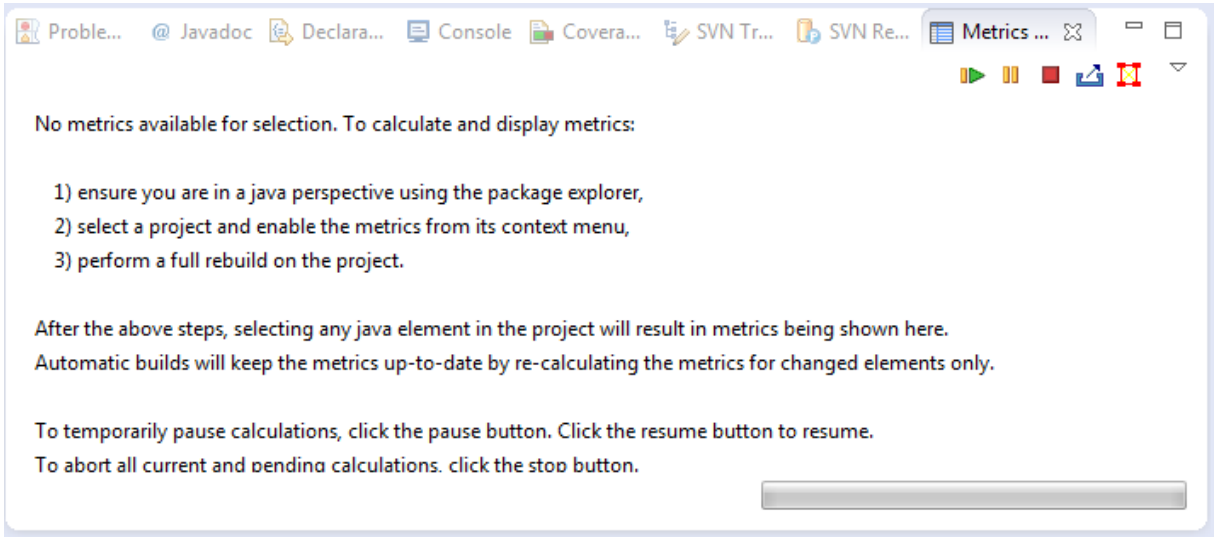
Um sich die Ergebnisse anzeigen lassen zu können, muss ein passender View angezeigt werden. Dazu wird im oberen Balken auf „Window“ geklickt und „Show View“ sowie „Other“ gewählt.



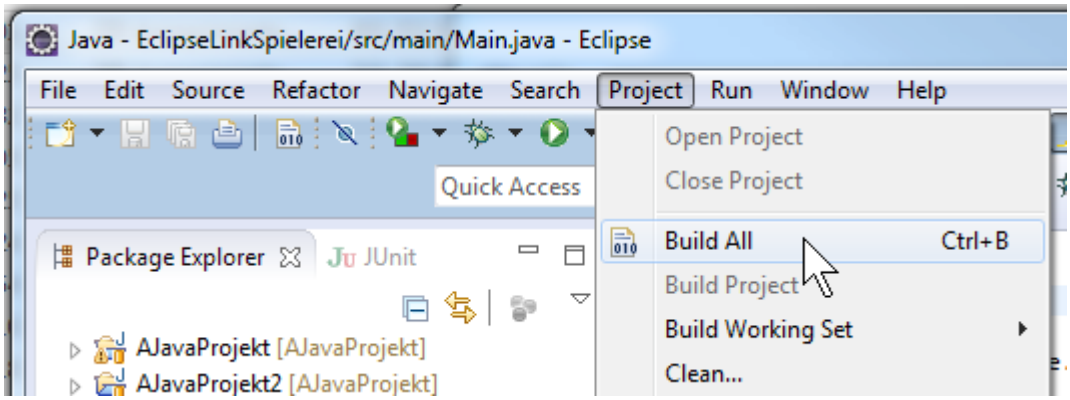
Unter „Metrics“ werden mehrere interessante Views angeboten, besonders wichtig ist der jetzt ausgewählte „Metrics View“.



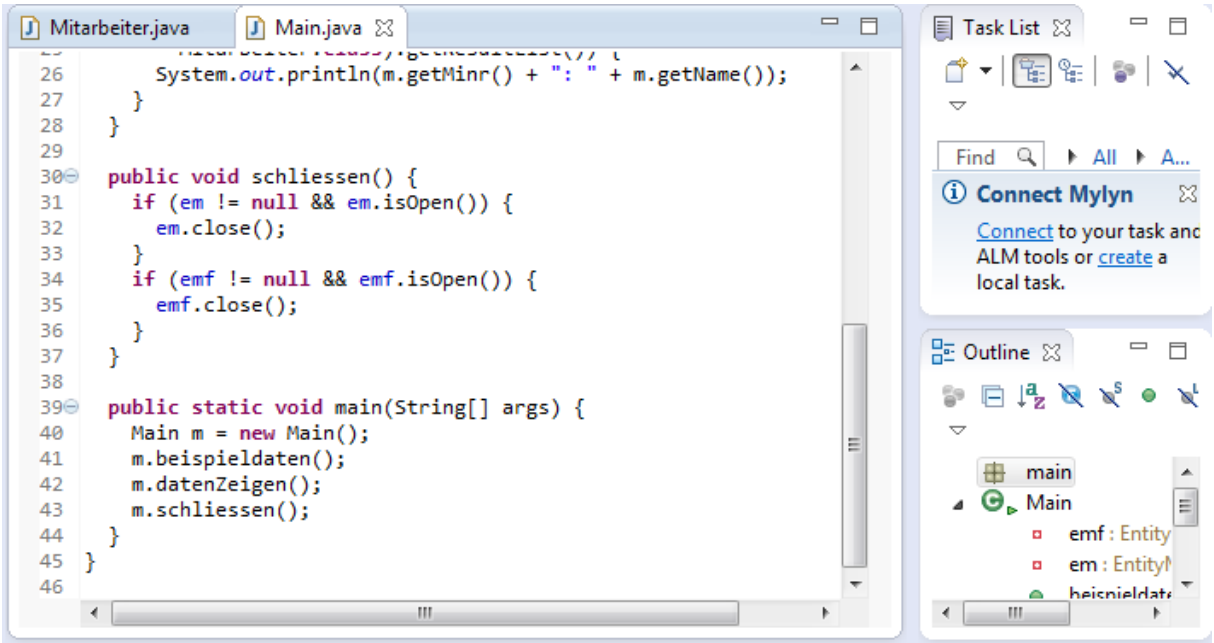
Falls die Metriken noch nicht angezeigt werden, reicht es oft aus, dass Projekt einmal zusammenzufalten und dann wieder zu öffnen.



Alternativ kann auch in der oberen Leiste unter „Project“ der Punkt „Build all“ ausgewählt.



Die resultierenden Ergebnisse sind sehr detailliert und können bis auf Methodenebene ausgeklappt werden.



Proble... Javadoc Declara... Console Covera... SVN Tr... SVN Re... Metrics ...

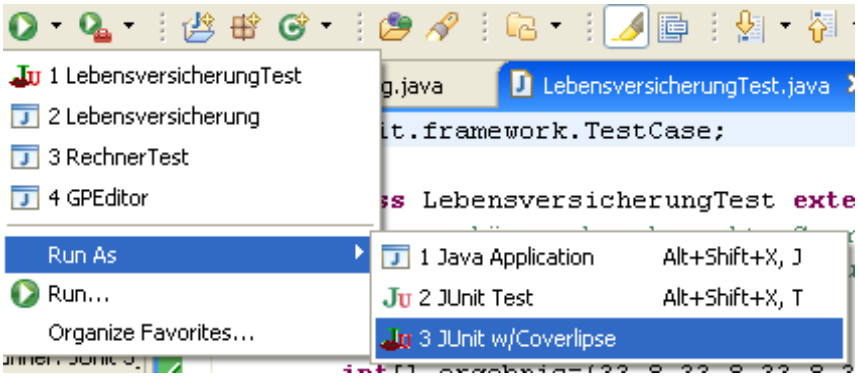
Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum
McCabe Cyclomatic Complexity (avg/max per		2,5	1,5	5	/EclipseLinkSpielerei/src/main/M
Main		2,5	1,5	5	/EclipseLinkSpielerei/src/main/M
schliessen	5				
beispieldaten	2				
datenZeigen	2				
main	1				
Number of Parameters (avg/max per method)		0,25	0,433	1	/EclipseLinkSpielerei/src/main/M
Nested Block Depth (avg/max per method)		1,5	0,5	2	/EclipseLinkSpielerei/src/main/M
Depth of Inheritance Tree (avg/max per type)		1	0	1	/EclipseLinkSpielerei/src/main/M
Weighted methods per Class (avg/max per typ	10	10	0	10	/EclipseLinkSpielerei/src/main/M
Number of Children (avg/max per type)	0	0	0	0	/EclipseLinkSpielerei/src/main/M
Number of Overridden Methods (avg/max per	0	0	0	0	/EclipseLinkSpielerei/src/main/M
Lack of Cohesion of Methods (avg/max per typ		0,5	0	0,5	/EclipseLinkSpielerei/src/main/M
Number of Attributes (avg/max per type)	2	2	0	2	/EclipseLinkSpielerei/src/main/M
Number of Static Attributes (avg/max per type	0	0	0	0	/EclipseLinkSpielerei/src/main/M
Number of Methods (avg/max per type)	3	3	0	3	/EclipseLinkSpielerei/src/main/M
Number of Static Methods (avg/max per type)	1	1	0	1	/EclipseLinkSpielerei/src/main/M
Specialization Index (avg/max per type)		0	0	0	/EclipseLinkSpielerei/src/main/M
Number of Classes	1				
Number of Interfaces	0				
Total Lines of Code	37				
Method Lines of Code (avg/max per method)	19	4,75	0,829	6	/EclipseLinkSpielerei/src/main/M

22 Kurzhinweise für weitere Werkzeuge in Eclipse

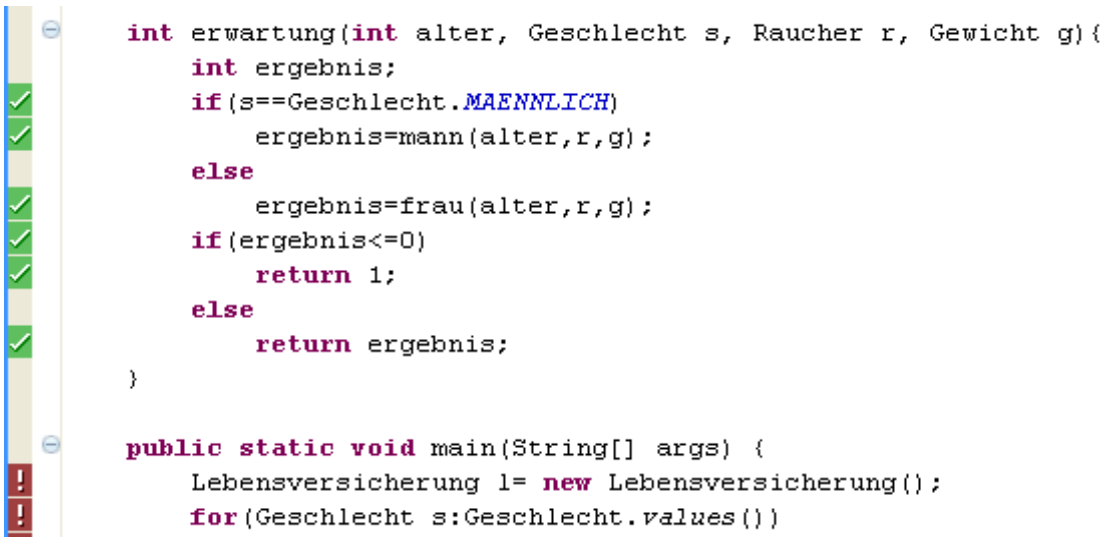
Die folgenden Werkzeuge können, wenn in Eclipse installiert, eventuell während eines SW-Projekts hilfreich sein. Sie sind nur als ein Angebot zum Ausprobieren zu sehen. Generell ist es sinnvoll, über weitere Werkzeuge nachzudenken, die dann aber auf individuellen Rechnern installiert werden müssen. Weiterhin 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.

22.1 Coverlipse (alt)

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“.



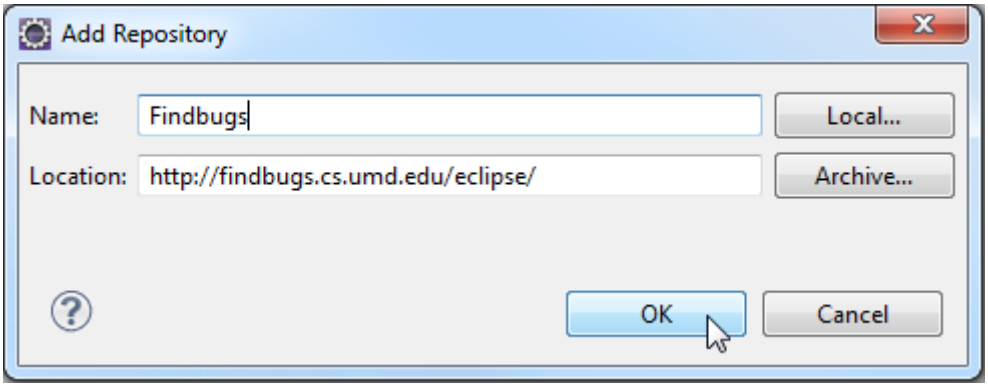
Im Java-Quellcode werden dann Markierungen gesetzt, die zeigen, ob die Zeile ausgeführt wurde.



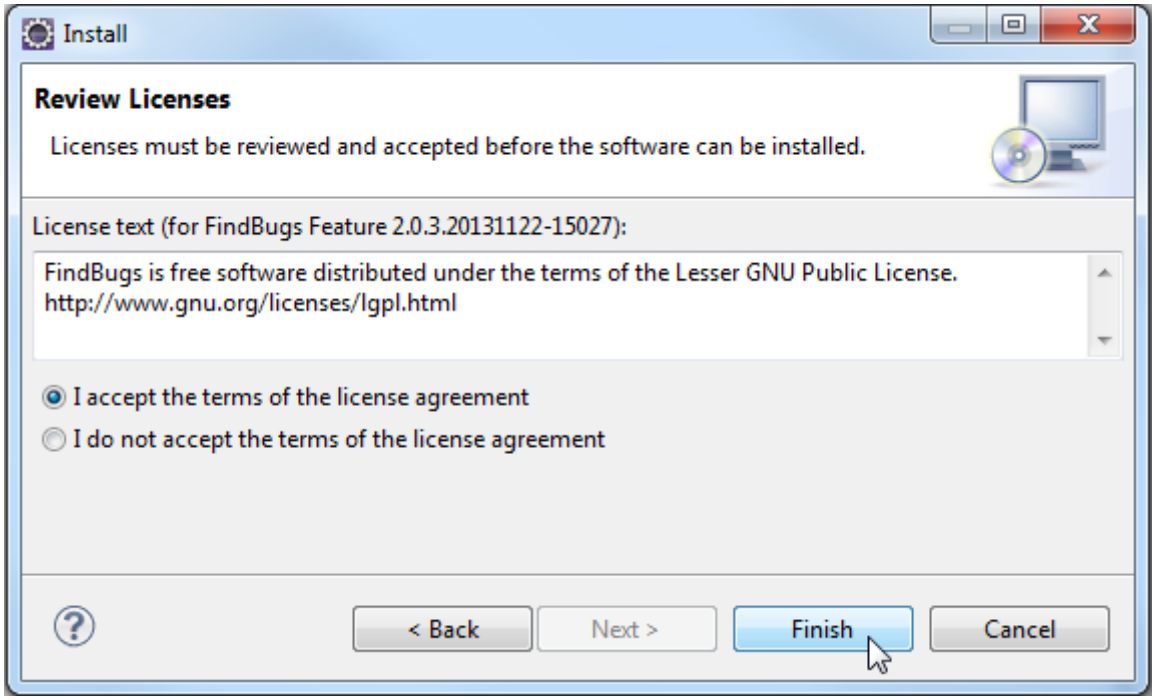
22.2 FindBugs (4.3.1)

Mit FindBugs (http://findbugs.sourceforge.net/) 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.

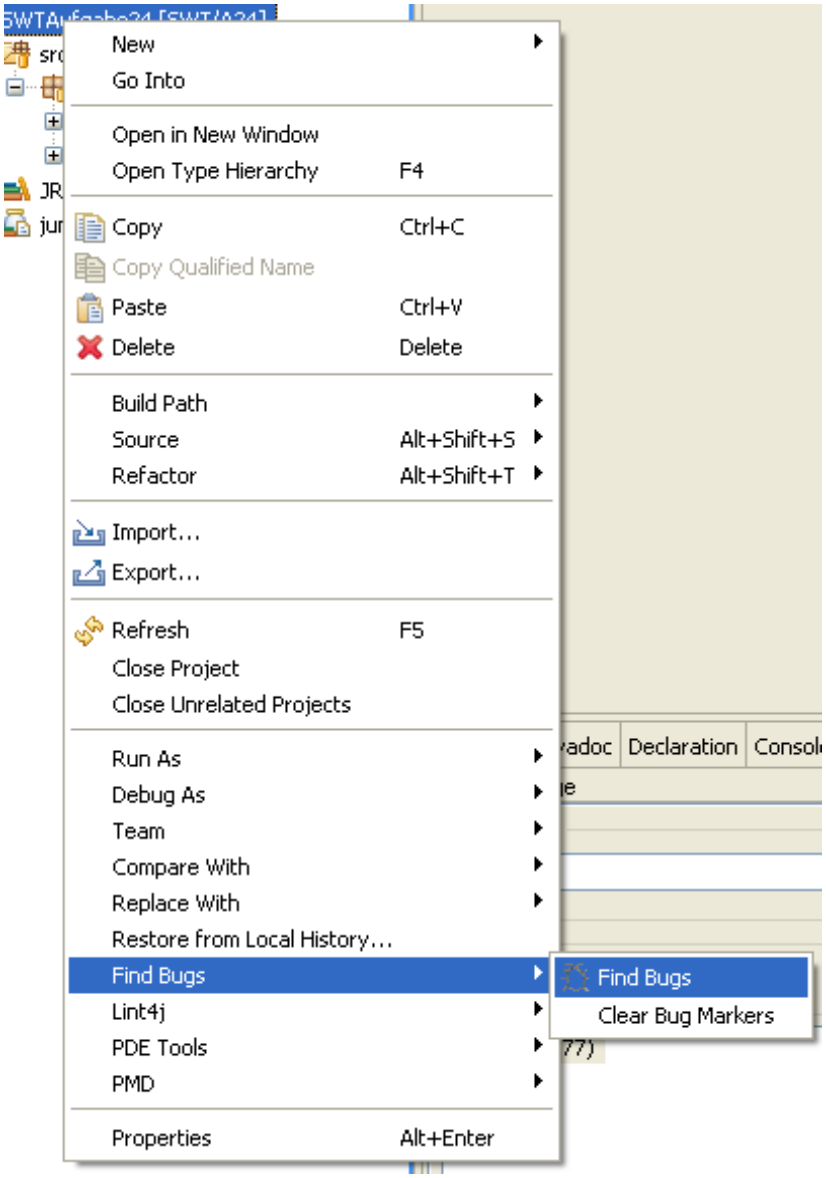
Findbugs kann über eine Update-Seite installiert werden, wie es in Kapitel 9 beschrieben ist. Die Adresse lautet http://findbugs.cs.umd.edu/eclipse/ .



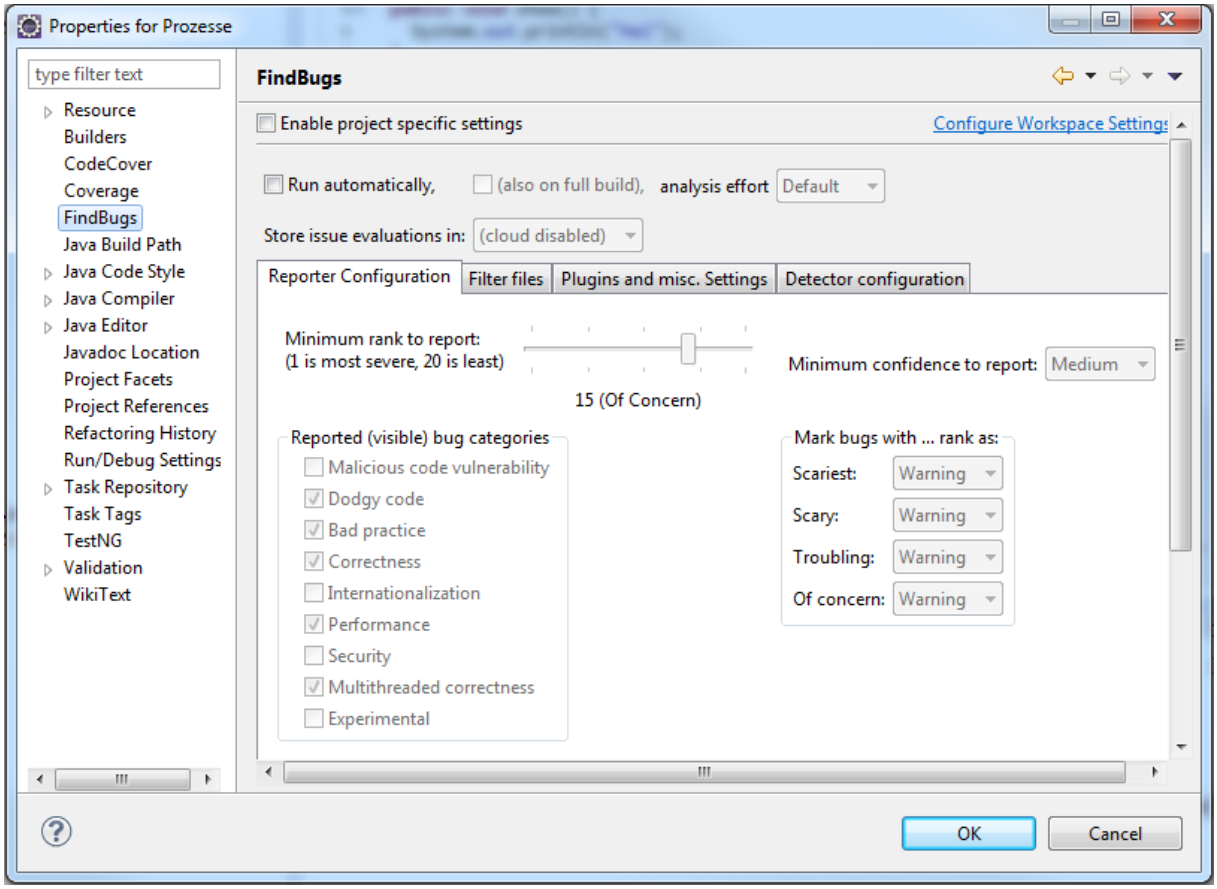
Man beachte, dass hier nicht die Eclipse sondern die Lesser GNU Public License genutzt wird.



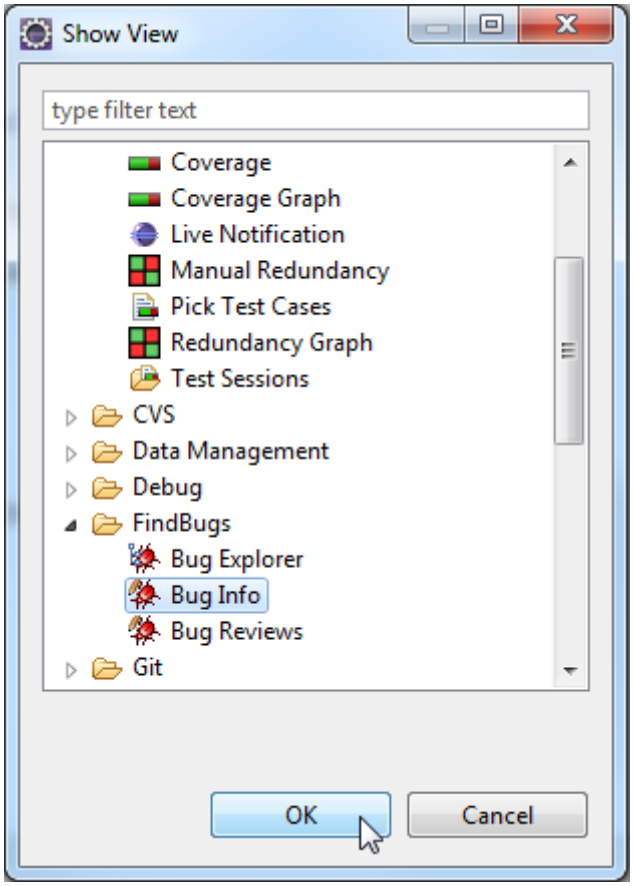
FindBugs wird mit einem Rechtklick auf dem Projekt unter „Find Bugs“ mit „Find Bugs“ gestartet. Potenzielle Fehler werden als Warnung markiert. Um diese Markierungen wieder zu entfernen, wird der Punkt „Clear Bug Markers genutzt“.



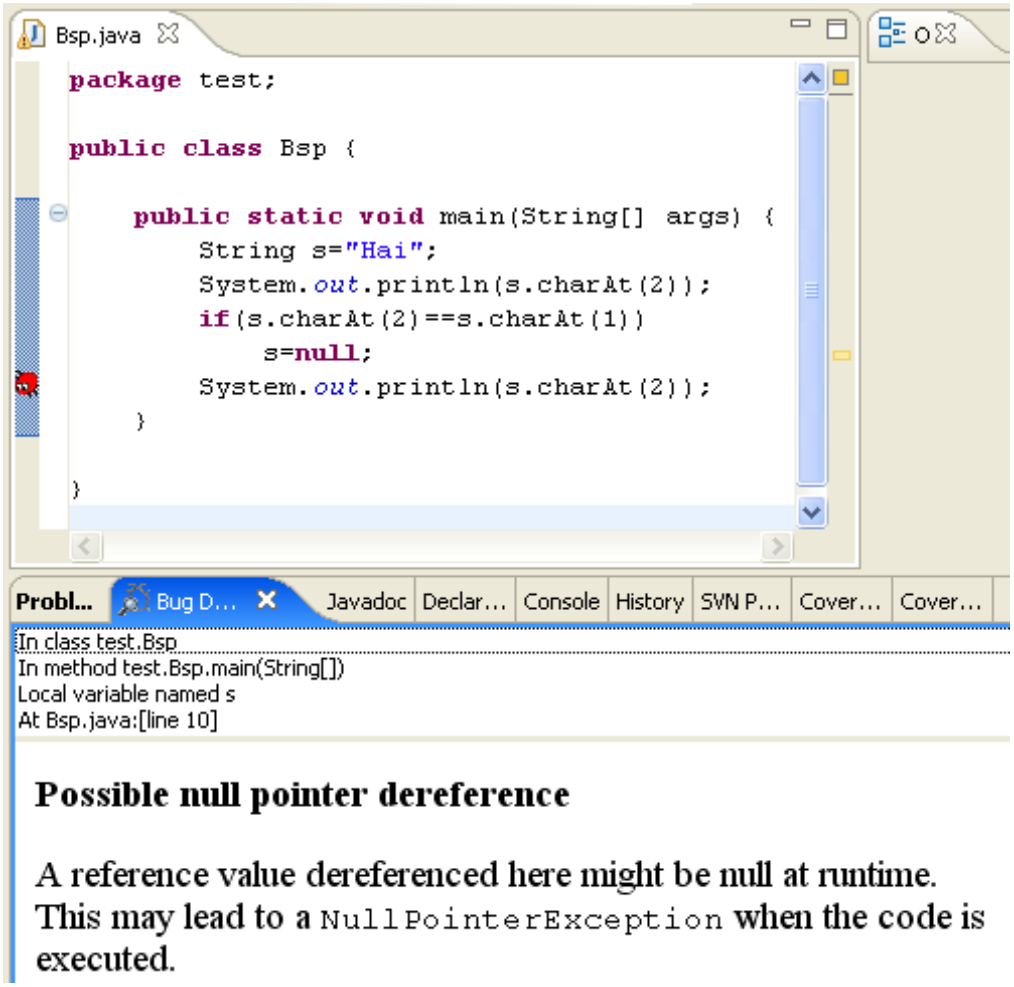
Die genaue Einstellung, was FindBugs untersuchen soll, wird unter „Properties" eingestellt. Zur Einstellung steht der folgende Dialog, hier für ein Beispielprojekt Prozesse, zur Verfügung.



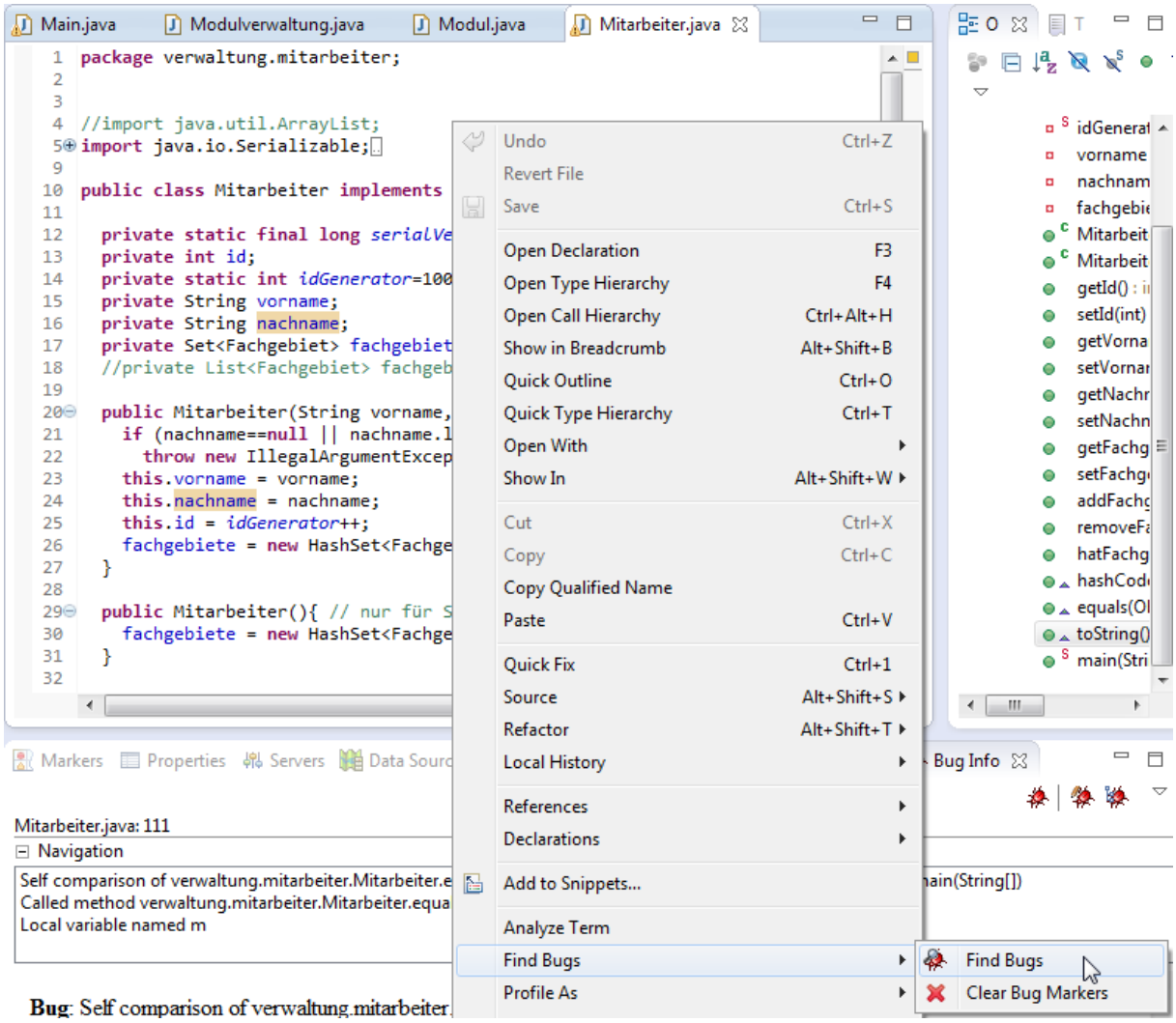
Unter „Window > Show View > Other…“ steht ein View „Bug Info“ zur Verfügung, der einen generellen Kommentar zum potenziellen Bug liefert.



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

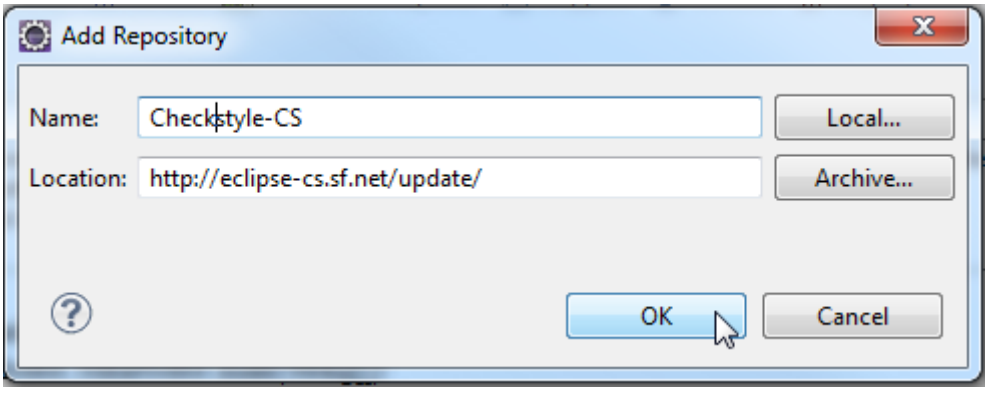


Man kann FindBugs auch direkt im Java-Editor aufrufen, indem man einen Rechtsklick in einem leeren Bereich des Editors macht und „FindBugs > FindBugs“ auswählt.

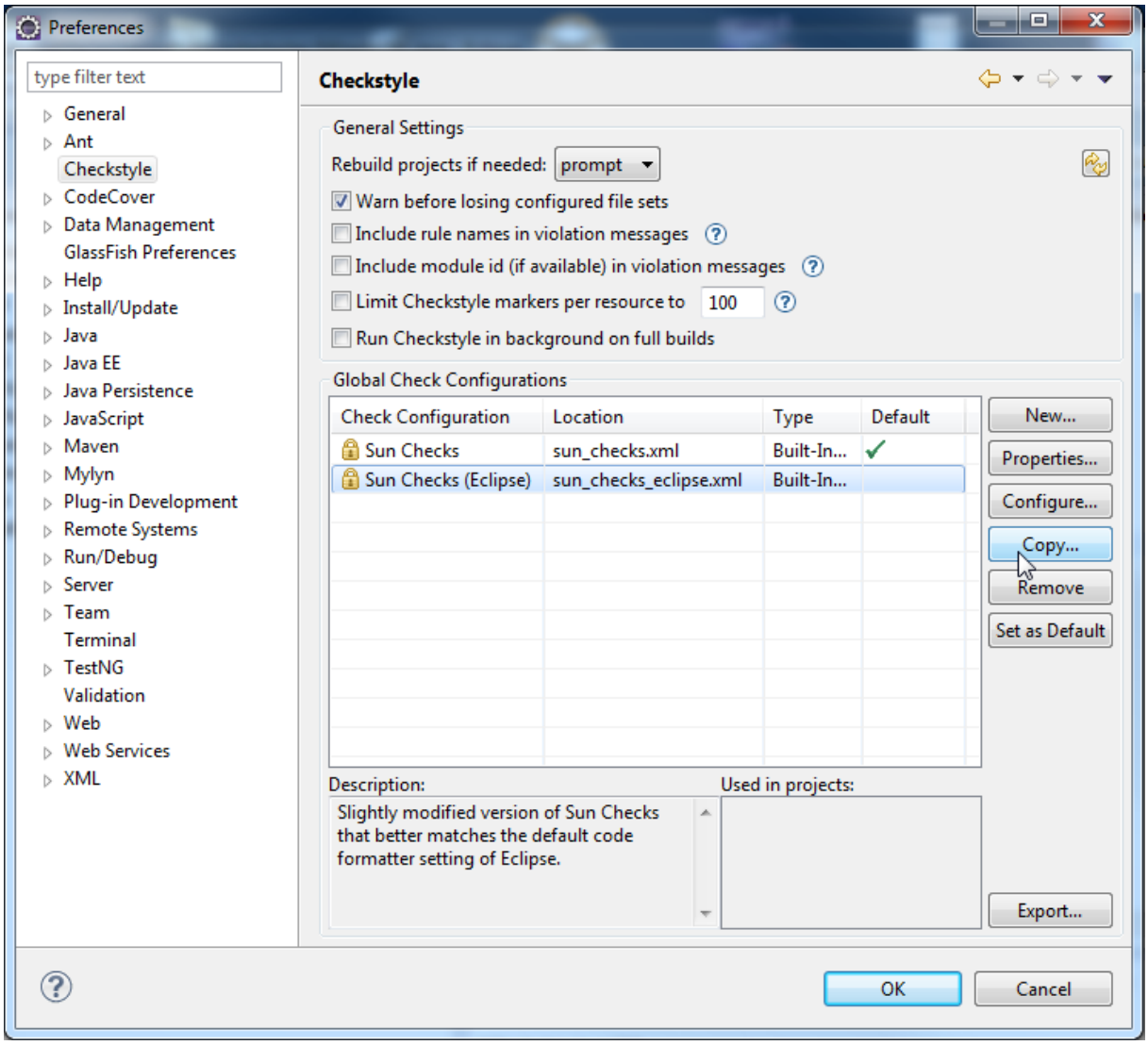


22.3 Checkstyle (4.3.1)

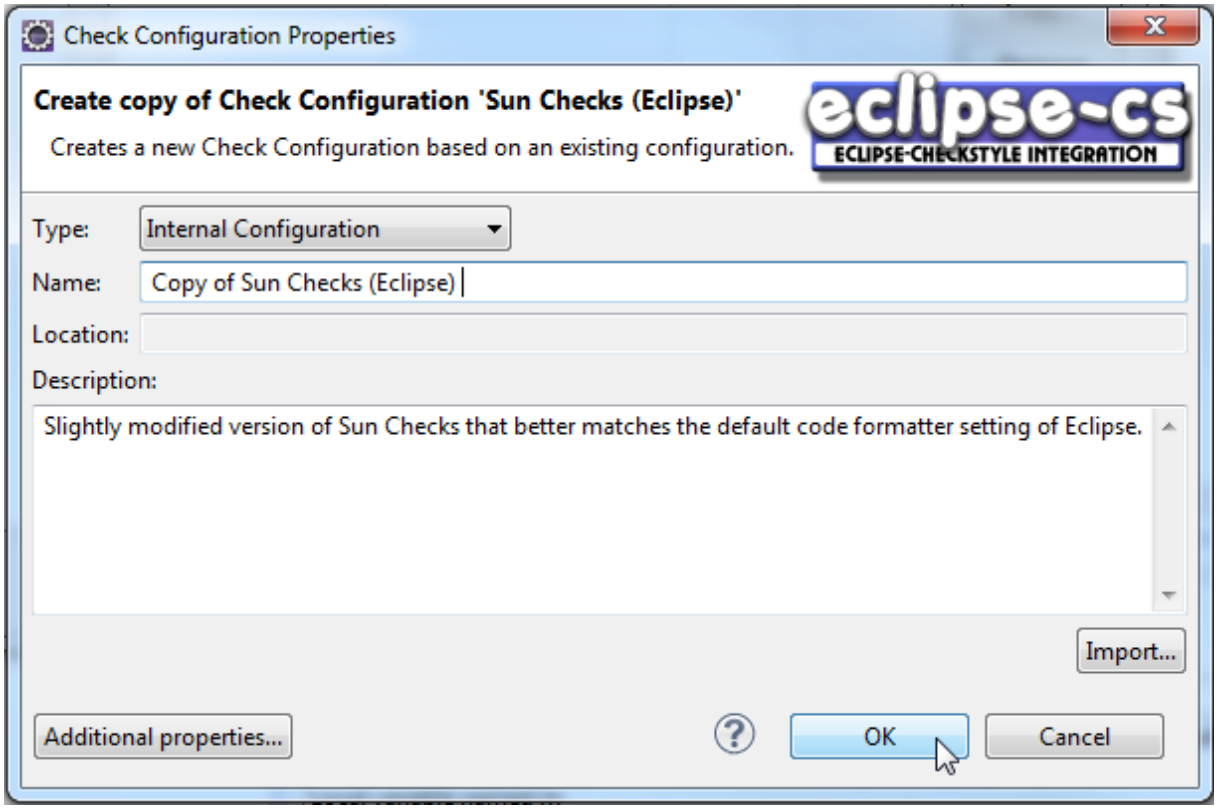
Wenn man sich auf Coding-Guidelines geeinigt hat, dann müssen diese überprüft werden. Diese Überprüfung ermöglicht Checkstyle (http://checkstyle.sourceforge.net/). Dabei steht eine sehr große Auswahl von Regeln zur Verfügung, so dass eine projektindividuelle Konfiguration unbedingt vorgenommen werden muss. Die Installation erfolgt, wie in Kapitel 9 beschrieben, mit der Adresse http://eclipse-cs.sf.net/update/ .



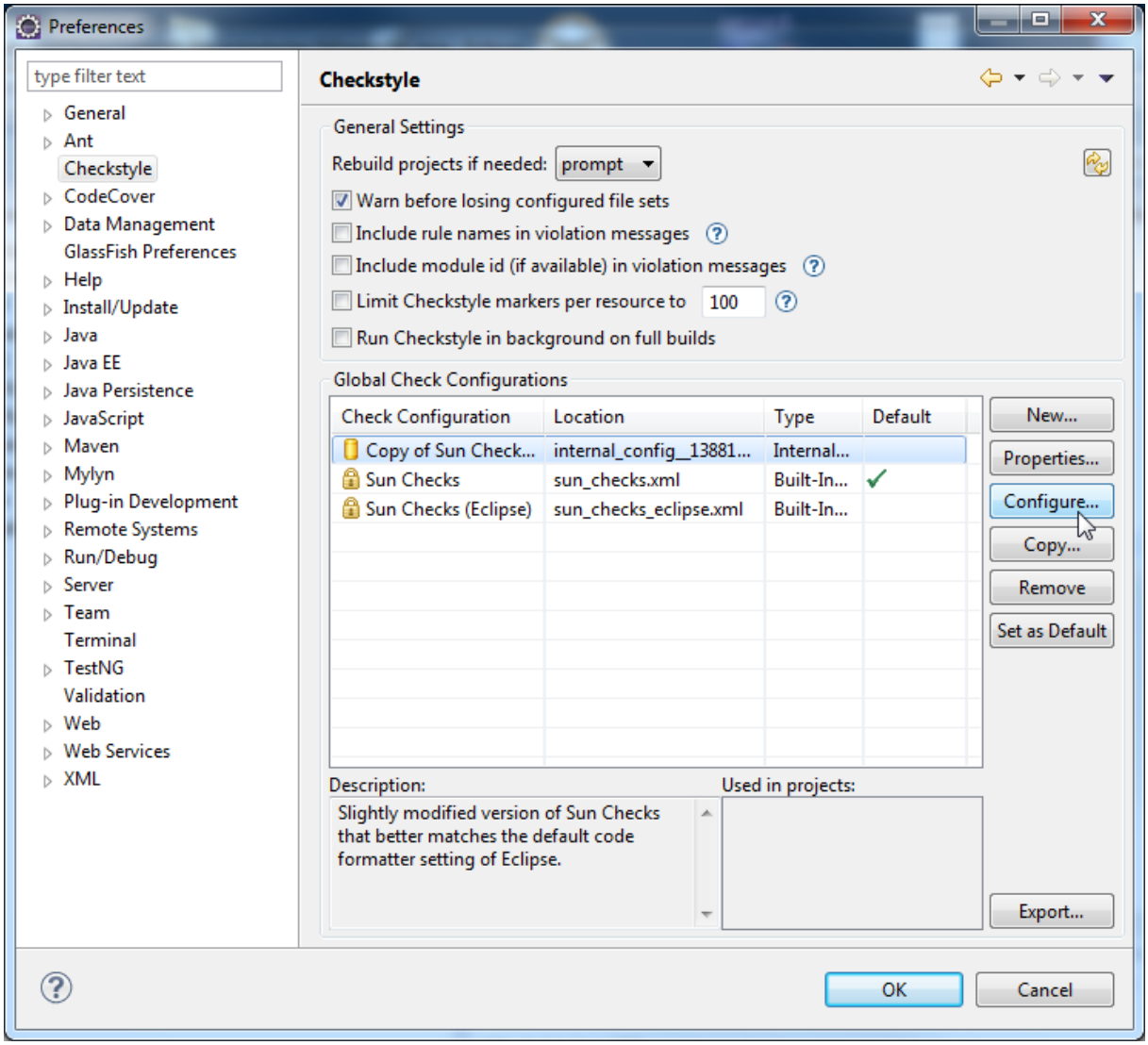
Die Auswahl der Regelmenge wird unter „Window > Preferences“ vorgenommen. Als Ausgangspunkt ist es sinnvoll, „Sun Checks (Eclipse)“ zu nehmen und auf der rechten Seite auf „Copy...“ zu drücken.



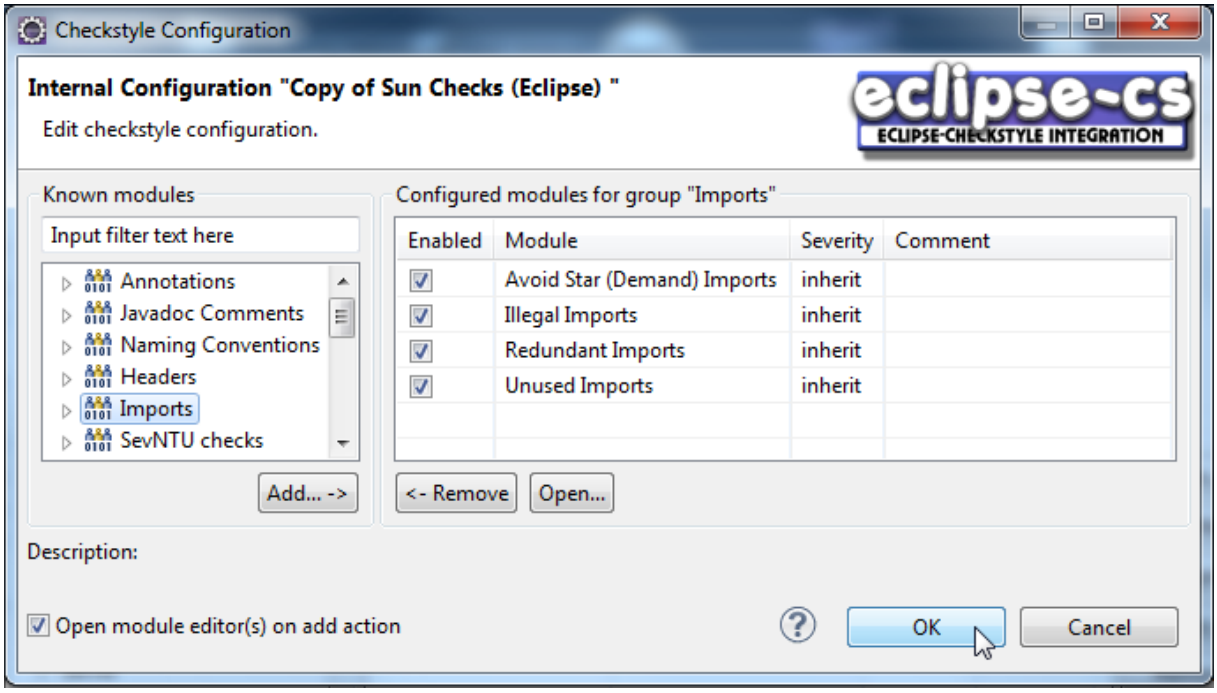
Die Kopie erhält einen eigenen Namen und kann eine eigene Beschreibung erhalten.



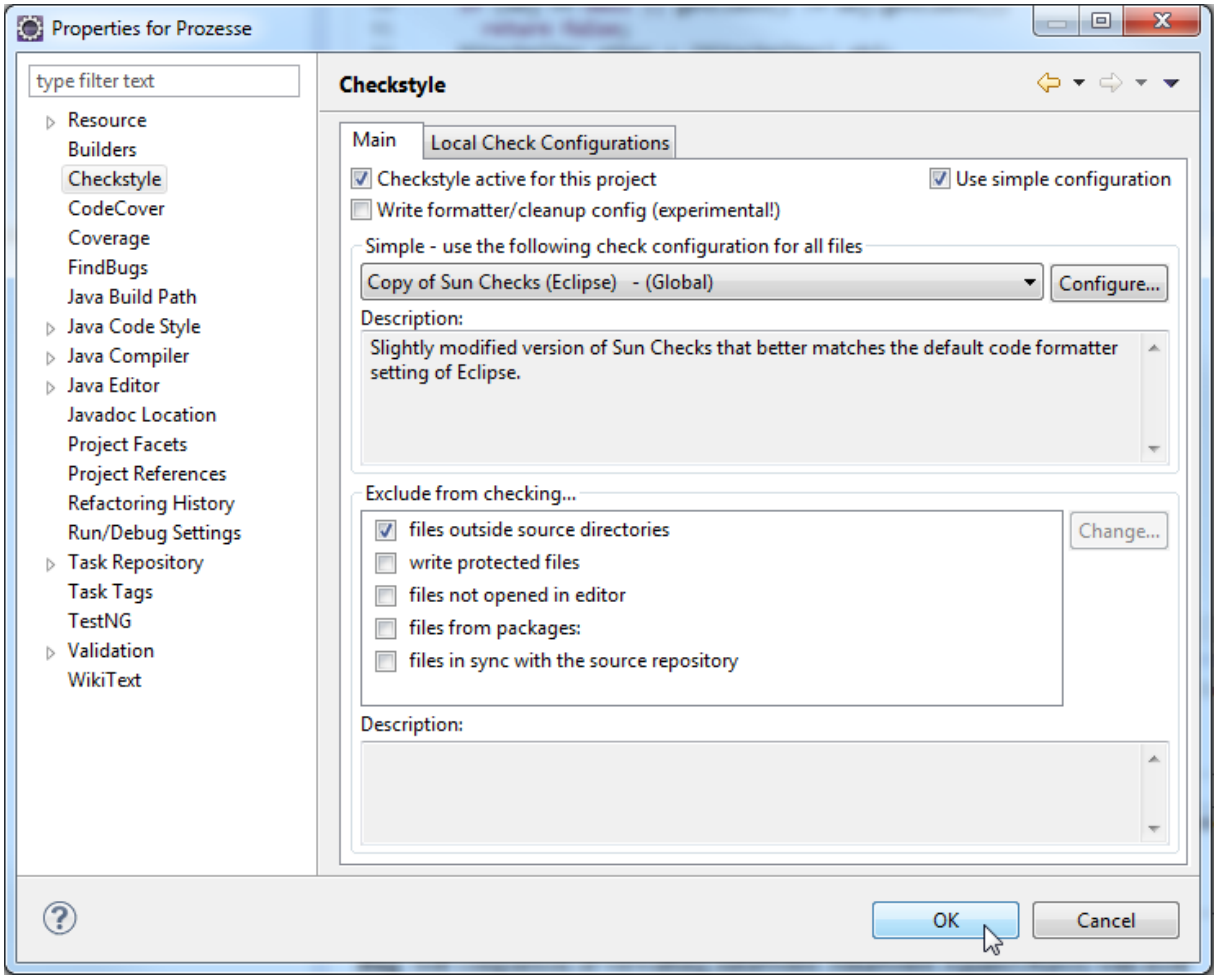
Nach einem „OK“ steht die Konfiguration zur Verfügung und kann unter „Configure...“ individuell eingestellt werden.



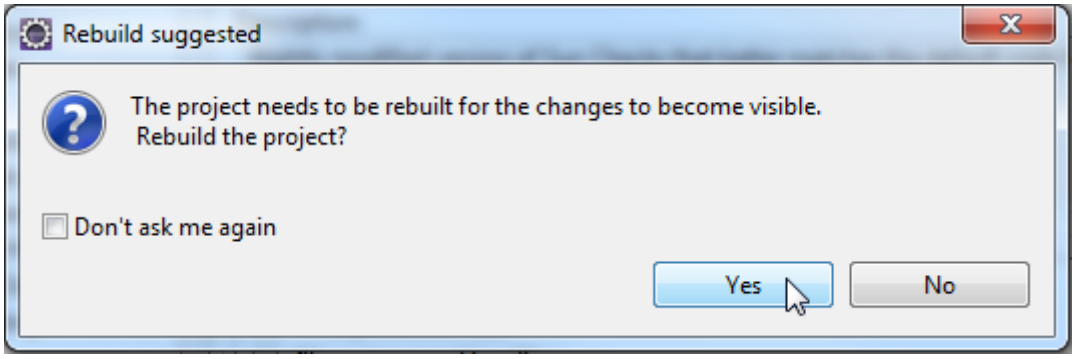
Ein Ausschnitt aus dem vorhandenen Regelkatalog sieht wie folgt aus.



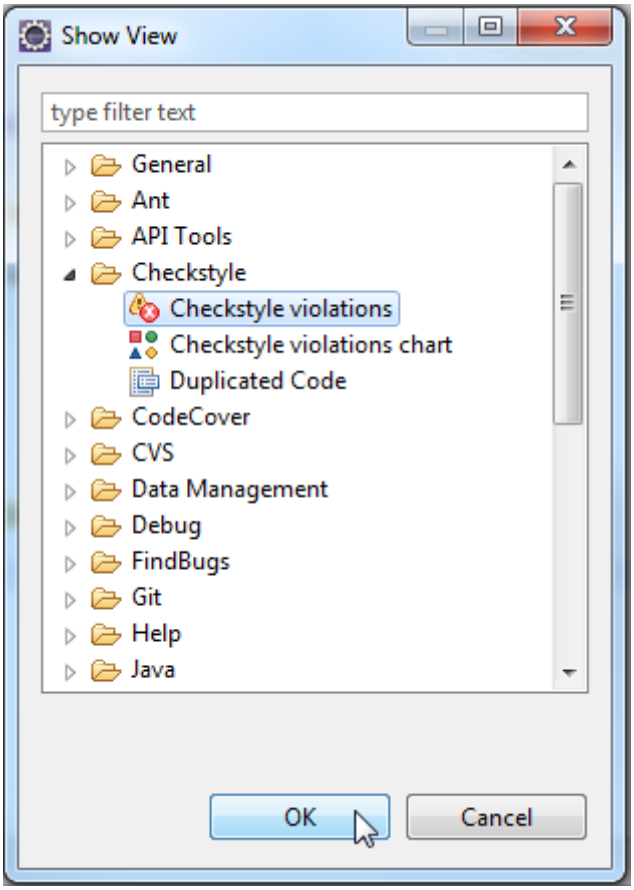
Checkstyle muss über die Properties des Projekts aktiviert werden. Weiterhin kann eingestellt werden, ob Checkstyle automatisch ausgeführt werden soll. Die neu erstellte Konfiguration muss dabei ausgewählt werden.

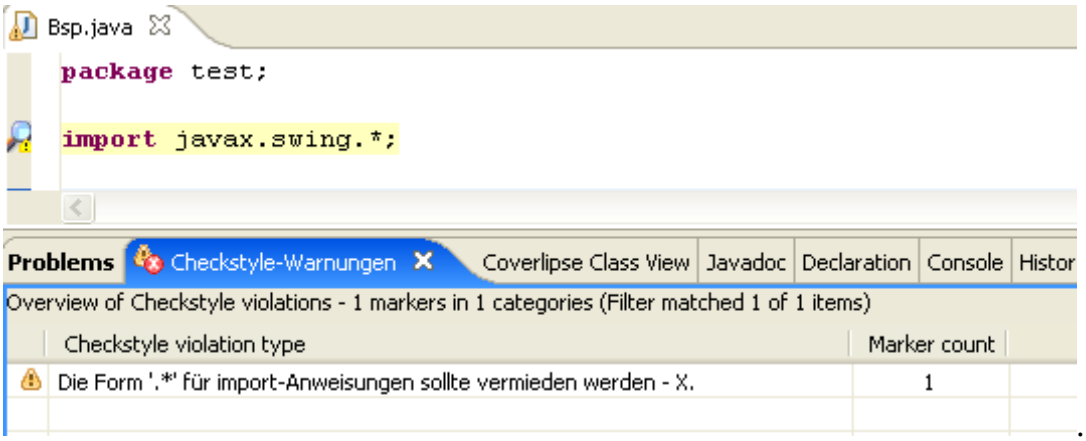


Am Anfang ist eine Neuübersetzung des zu untersuchenden Programms notwendig.



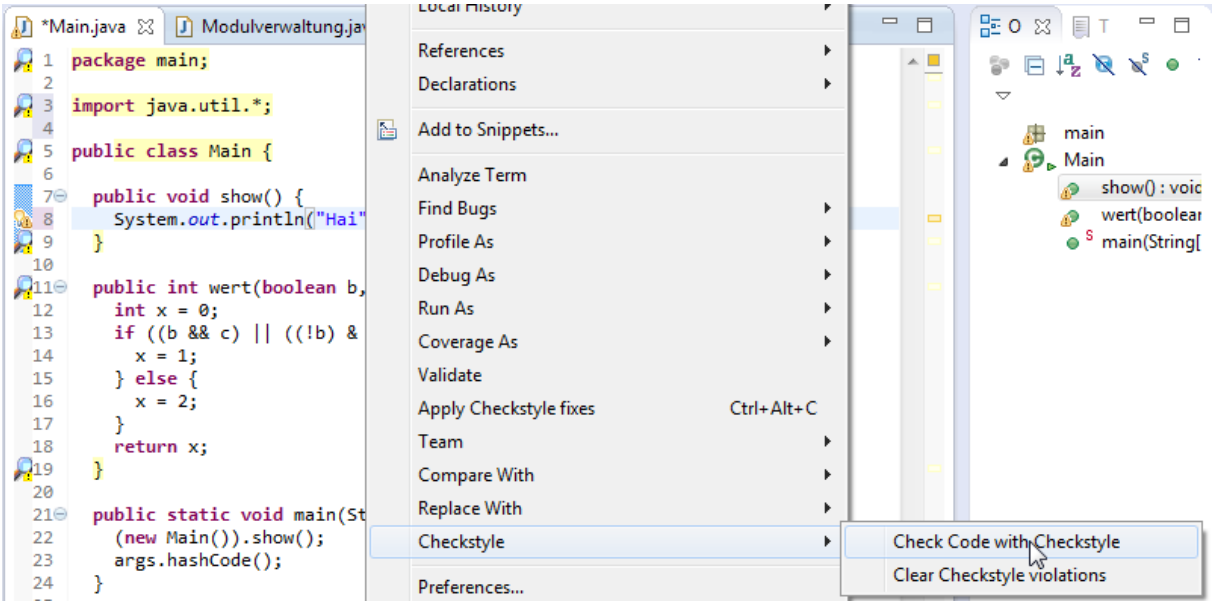
Probleme werden dann mit einem gesonderten Symbol angezeigt, weiterhin gibt es unter „Window > Show View > Other“ die Möglichkeit, einen View für Checkstyle-Warnungen auszuwählen.





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

Man kann CheckStyle auch für eine konkrete, gerade in Bearbeitung befindliche Java-Datei nutzen. Dazu wird ein Rechtsklick an einer freien Stelle des Editors gemacht und „CheckStyle > „ ausgewählt.


*Main.java
Modulverwaltung.ja
1 package main;
3 import java.util.*;
5 public class Main {
7 public void show() {
8 System.out.println("Hai"
9 }
11 public int wert(boolean b,
12 int x = 0;
13 if ((b && c) || ((!b) &
14 x = 1;
15 } else {
16 x = 2;
17 }
18 return x;
19 }
21 public static void main(St
22 (new Main()).show();
23 args.hashCode();
24 }
References
Declarations
Add to Snippets...
Analyze Term
Find Bugs
Profile As
Debug As
Run As
Coverage As
Validate
Apply Checkstyle fixes Ctrl+Alt+C
Team
Compare With
Replace With
Checkstyle
Preferences...
Check Code with Checkstyle
Clear Checkstyle violations
main
Main
show() : voic
wert(boolear
main(String[

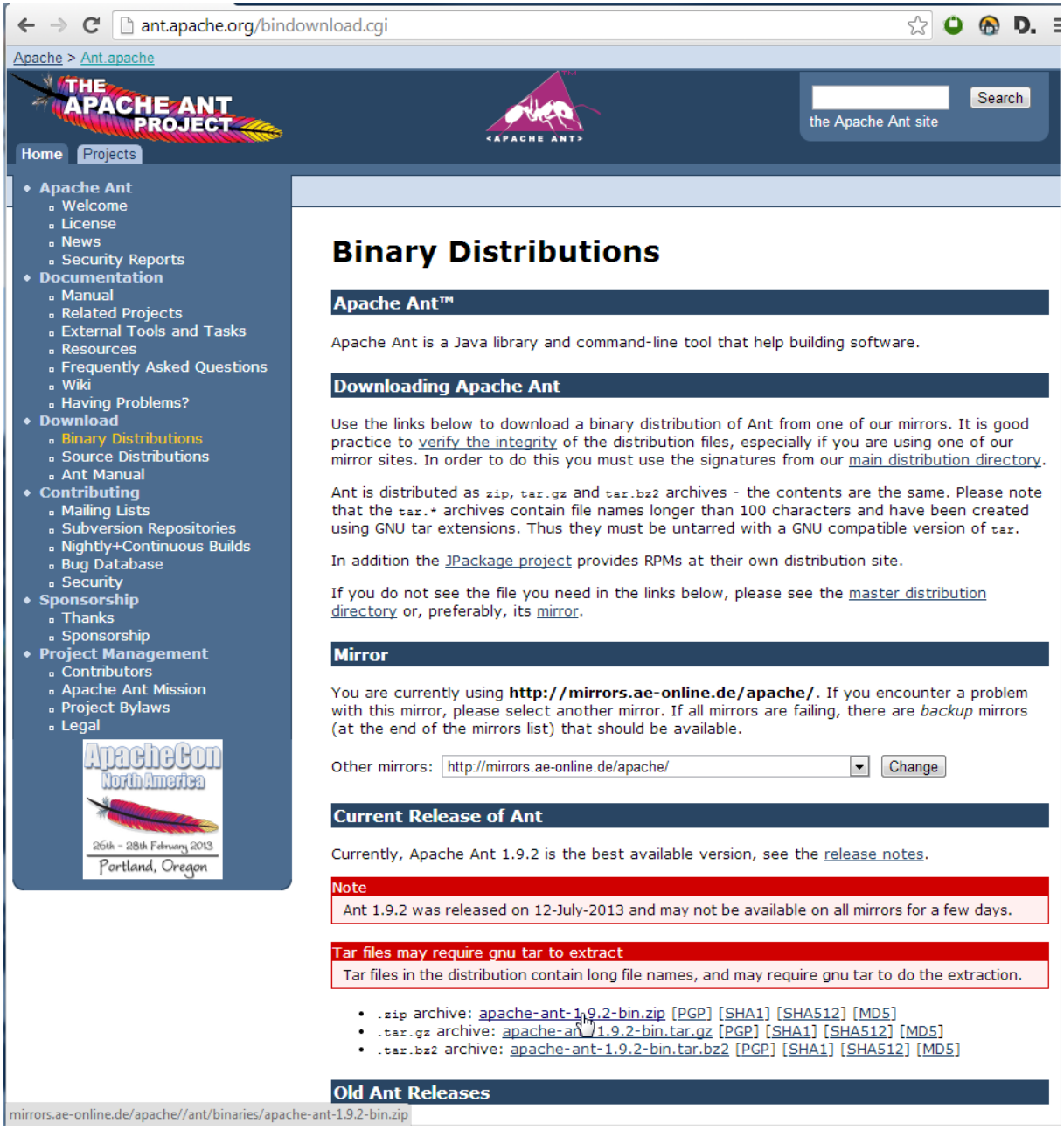
Anhang

Im Anhang werden Informationen zu weiteren Werkzeugen hinterlegt, die nicht unmittelbar in Eclipse eingebunden sind, aber häufig im Zusammenhang mit Eclipse-Projekten genutzt werden. Die letzte Klammer gibt immer die betrachtete Versionsnummer an. Im Text werden dann teilweise ältere Versionen genannt, insofern sich der jeweilige Nutzungsprozess nicht verändert hat.

Weiterhin sind in den Anhang einige ältere Information zu Eclipse-Plugins enthalten, die aus Gründen der Versionssicherung aufgeführt werden.

A Ant-Installation (1.9.2)

Vor dem Anfang sei u. a. auf die Ant-Dokumentation hingewiesen, die gut geschrieben und viel mehr nützliche Informationen als dieser kleine Einführungstext enthält. Geladen wird Ant (http://ant.apache.org/) als Zip-Verzeichnis.



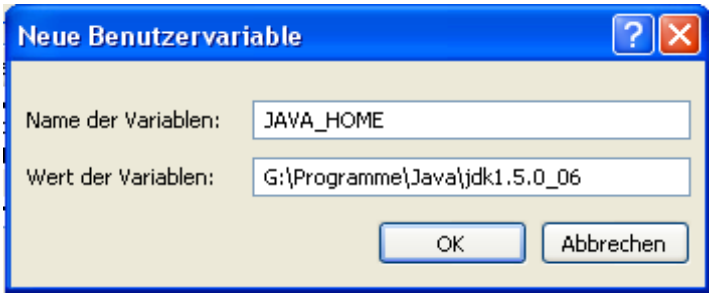
Das Verzeichnis mit der Dopkumentation ist H:\Programme\ant\docs\manual\index.html, wenn Ant unter H:\Programme ausgepackt wurde. Ant selber ist Teil von Eclipse und muss nur installiert werden, wenn es unabhängig von Eclipse zur Verfügung stehen soll.

Zur Installation wird die Zip-Datei 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>
```

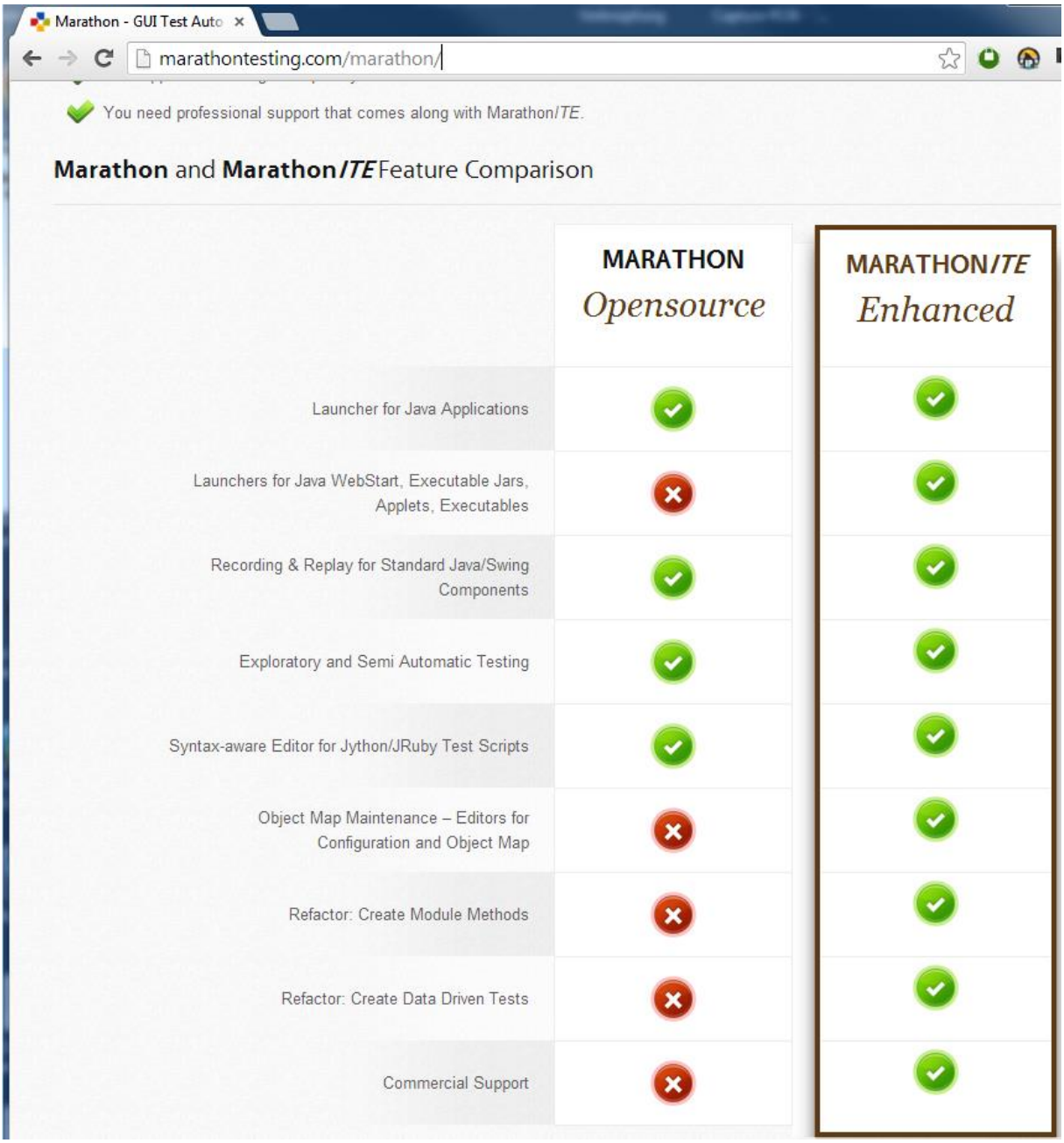
Ant arbeitet typischerweise eine Datei build.xml ab, in der die einzelnen Befehle stehen.

B Marathontesting (3.3.8.2)

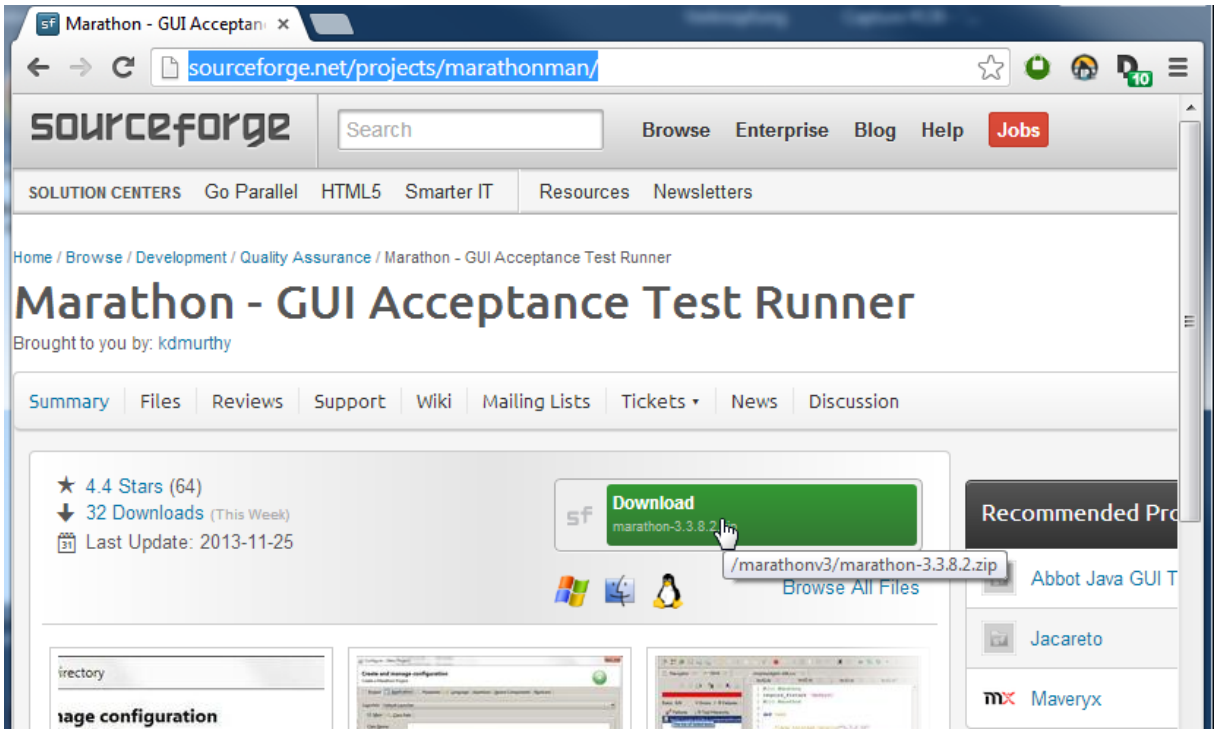
Marathontesting (http://www.marathontesting.com) ist ein sogenanntes Capture-and-Replay-Werkzeug zum Testen von SW ausgehend von Ihrer Oberfläche. Aktionen des Nutzers können aufgezeichnet und wieder abgespielt werden. Die Aufzeichnungen werden um Zusicherungen ergänzt, die bei erneuten Abläufen geprüft werden.

B.1 Installation

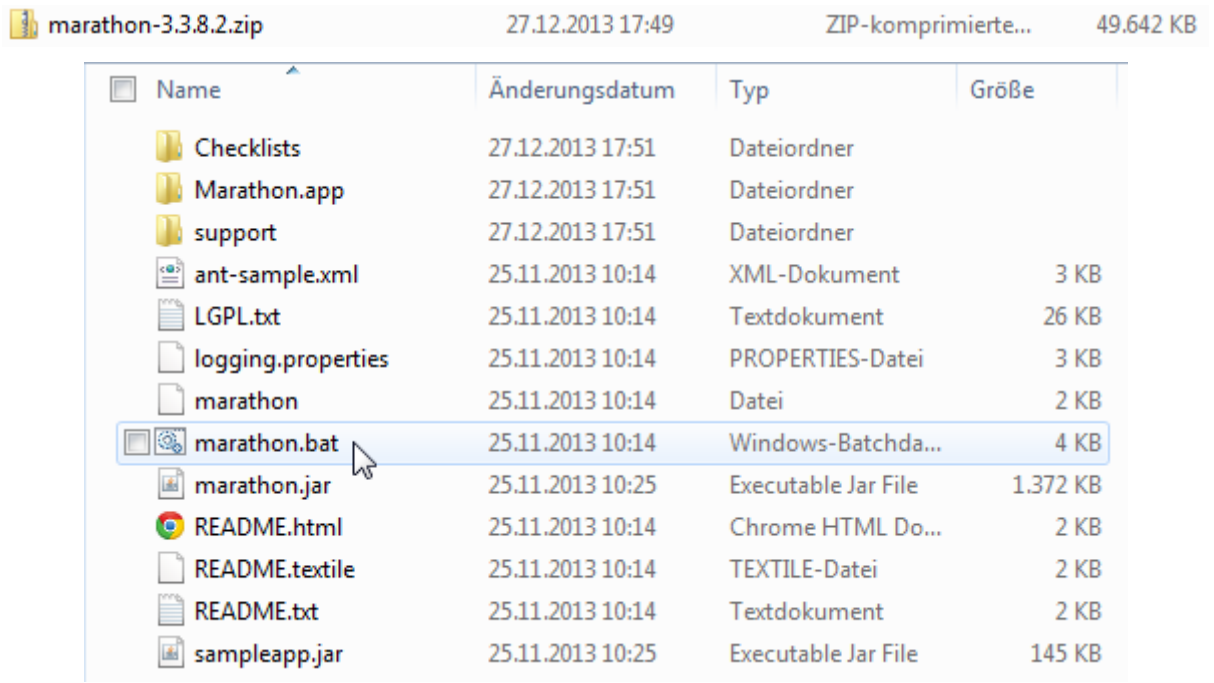
Man beachte, dass hier die OpenSource-Variante genutzt wird, die weniger Funktionen zur Verfügung stellt.



Der eigentliche Download findet unter http://sourceforge.net/projects/marathonman/ statt. Hier wird der direkt angebotene Download genutzt, alternativ kann man auch den darunter liegenden Link „Browse All Files“ nutzen, falls man andere Versionen benötigt.



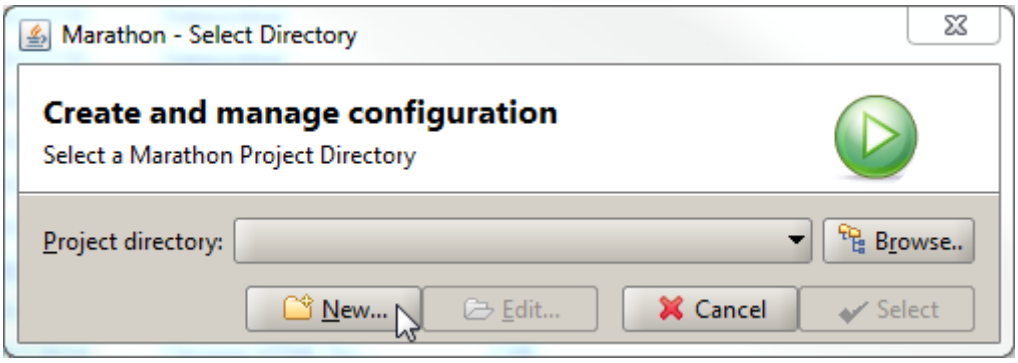
Nach dem Runterladen kann das Programm in einem beliebigen Verzeichnis ausgepackt werden.



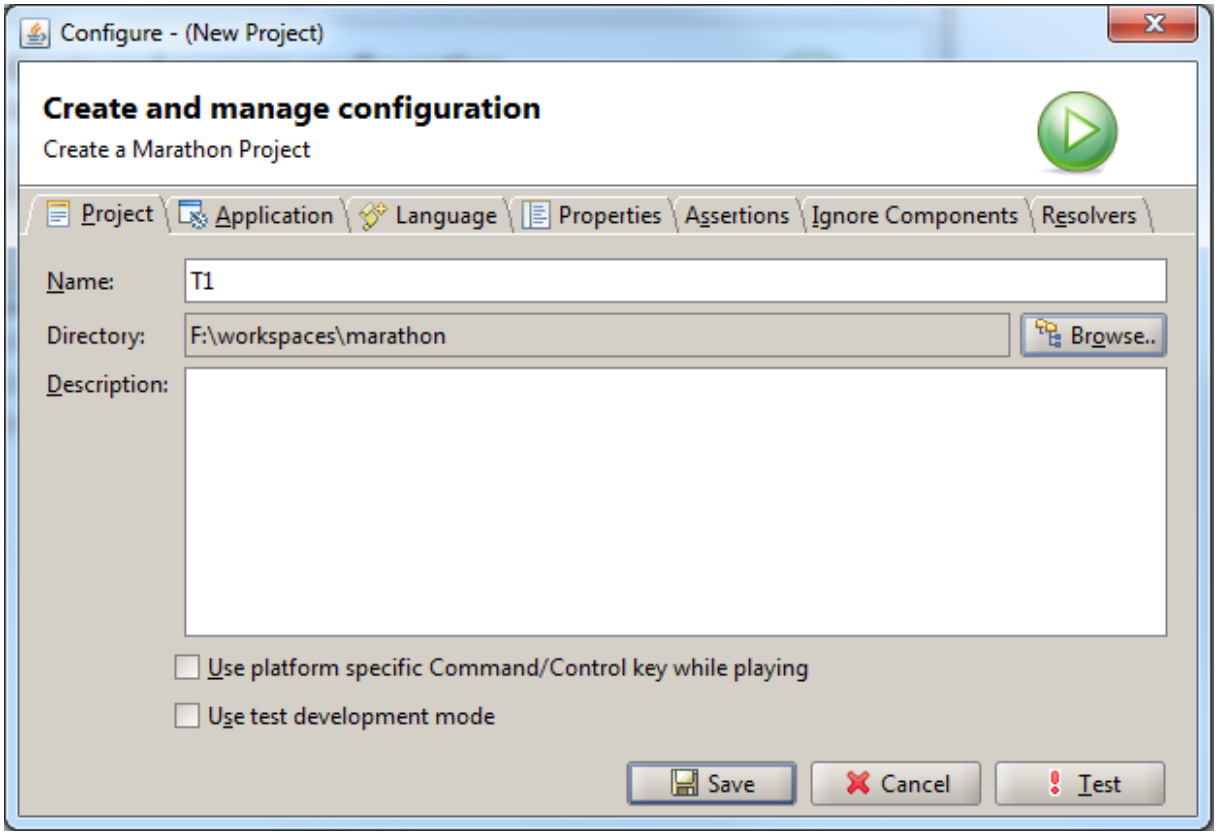
Abhängig vom Betriebssystem wird die passende Datei zum Programmstart ausgewählt, im obigen Beispiel ist dies marathon.bat. Eventuell müssen Einstellungen der eigenen Sicherheitsprogramme beachtet werden.

B.2 Anlegen eines Projekts, einer Konfiguration

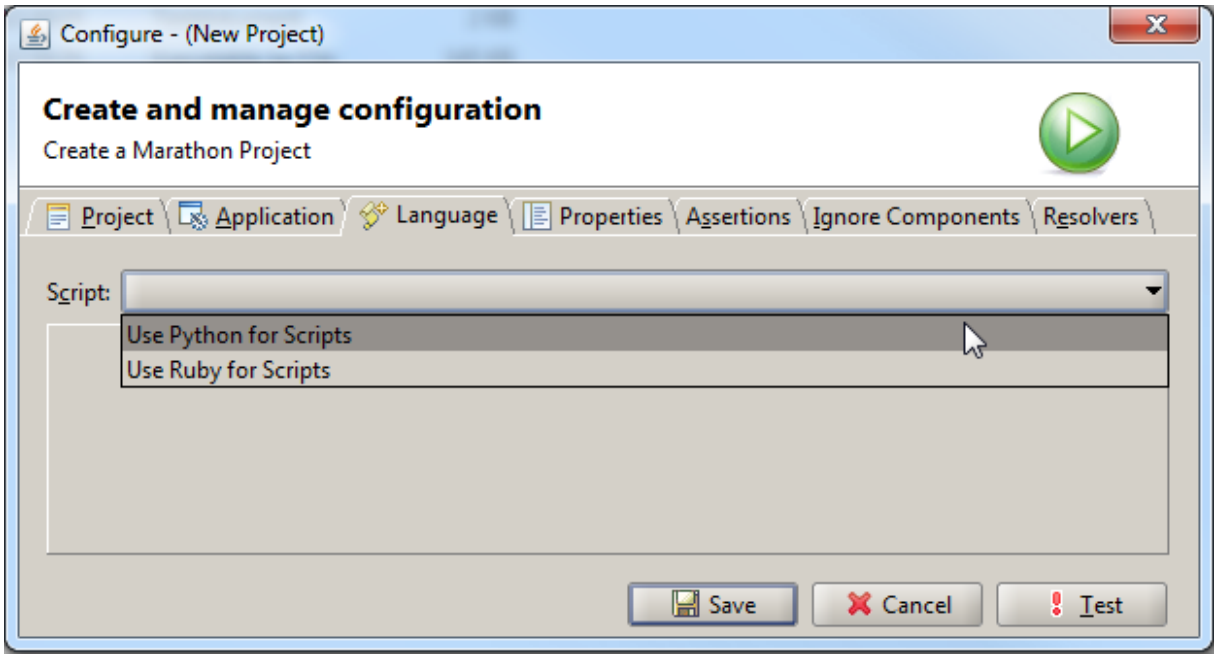
Nach dem Startbildschirm können neue Projekte angelegt und existierende Projekte ausgewählt werden. Hier wird ein neues Projekt über „New...“ angelegt.



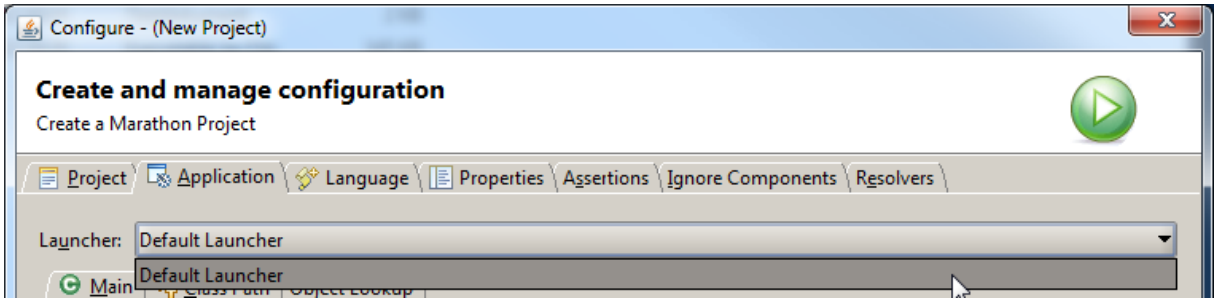
Es wird ein Projekt mit Namen T1 angelegt, das Projektverzeichnis unter „Directory“ ist frei wählbar und muss nicht das Java-Projektverzeichnis sein.



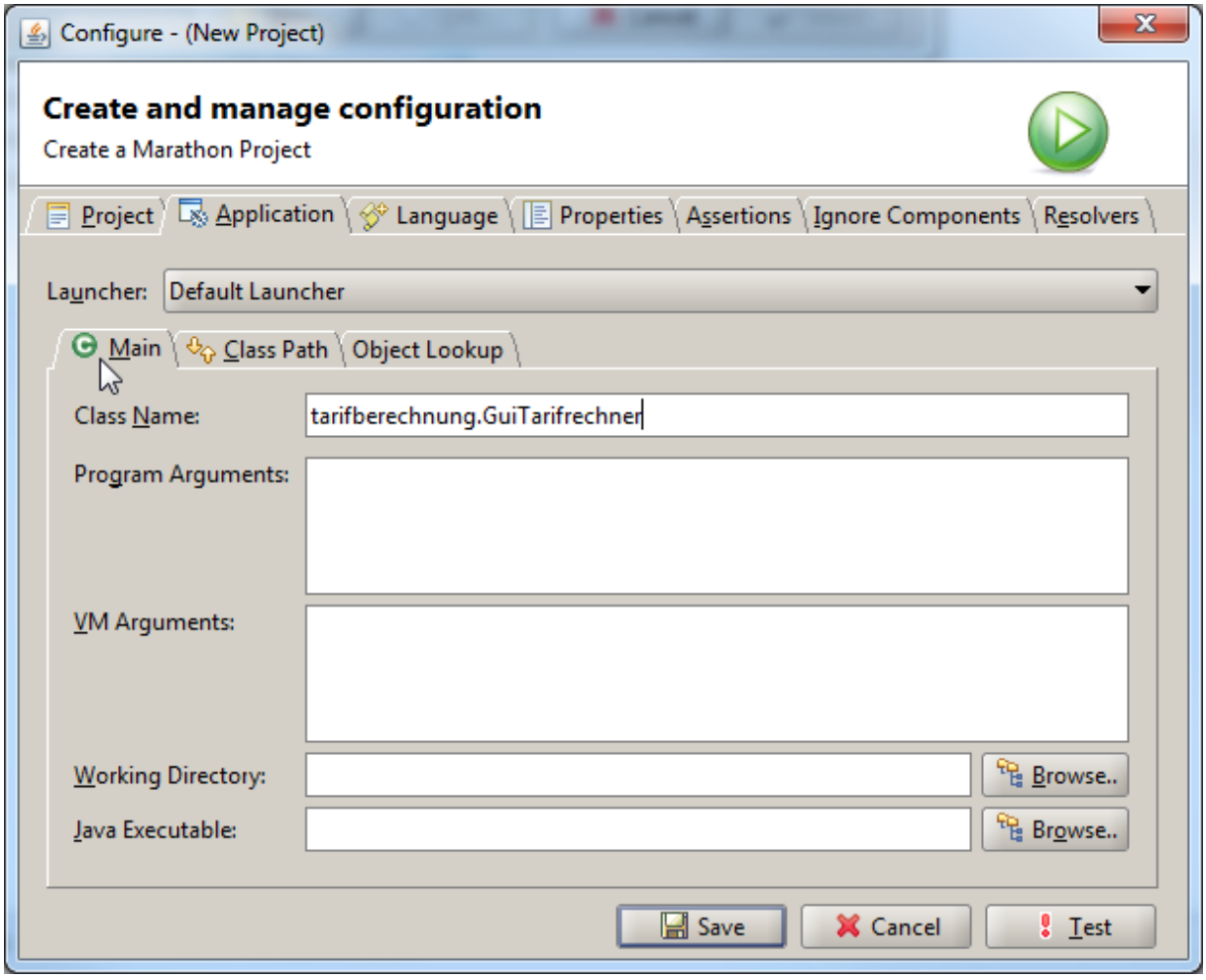
Unter Language wird die Script-Sprache für die Tests festgelegt. Hier wird auf keine Details eingegangen und Jython gewählt.



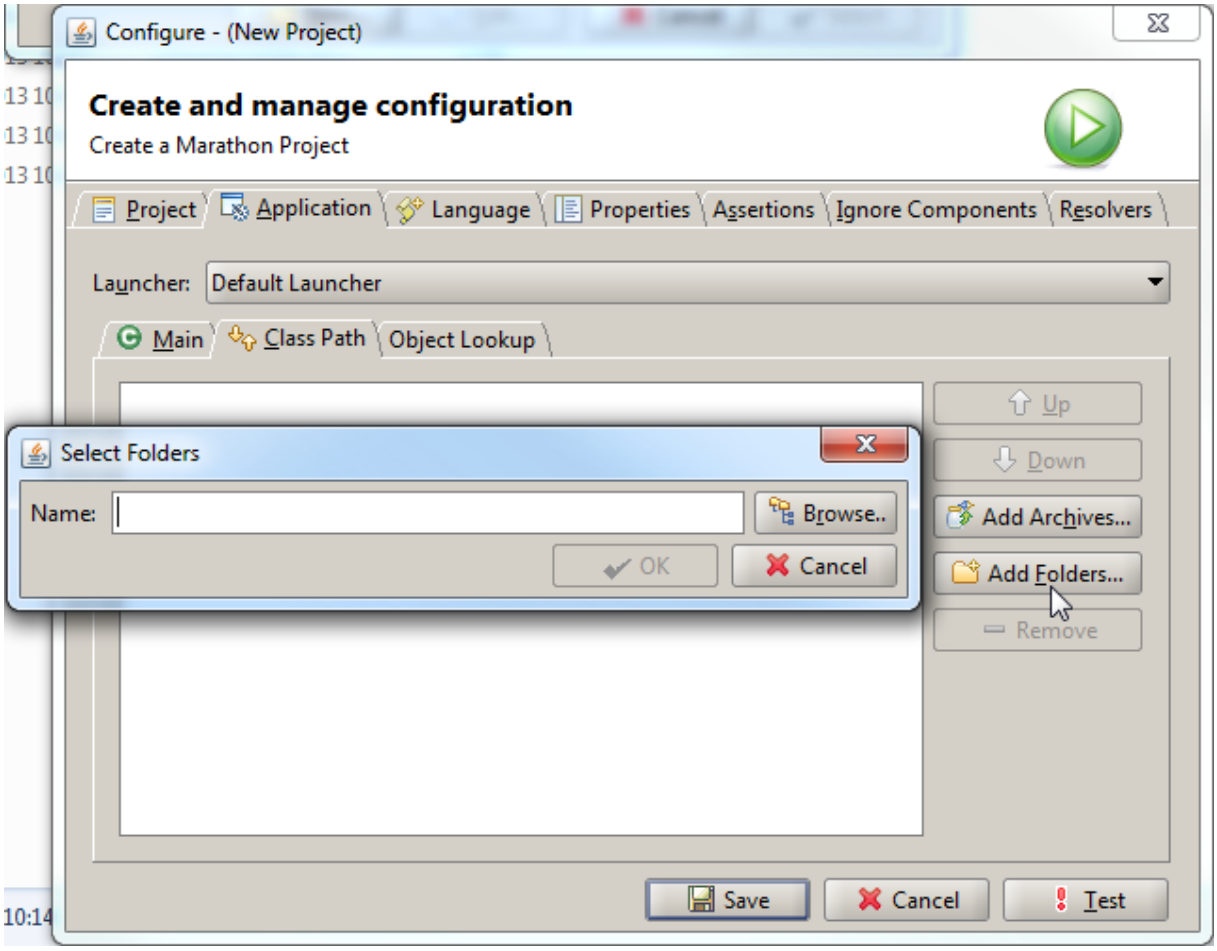
Unter Application muss zunächst ein Launcher, hier ist nur der Default Launcher möglich, selektiert werden, damit danach weitere Einstellungen möglich sind.



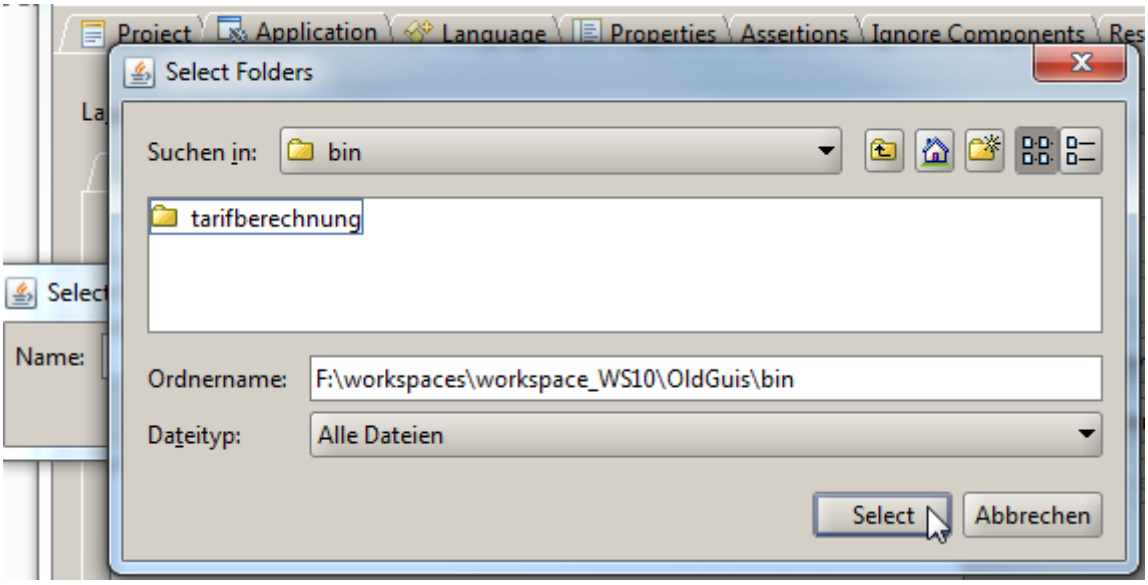
Beim Reiter „Main“ wird unter „Class Name:“ der Name der zu startende Klasse inklusive des Paketnamens angegeben.



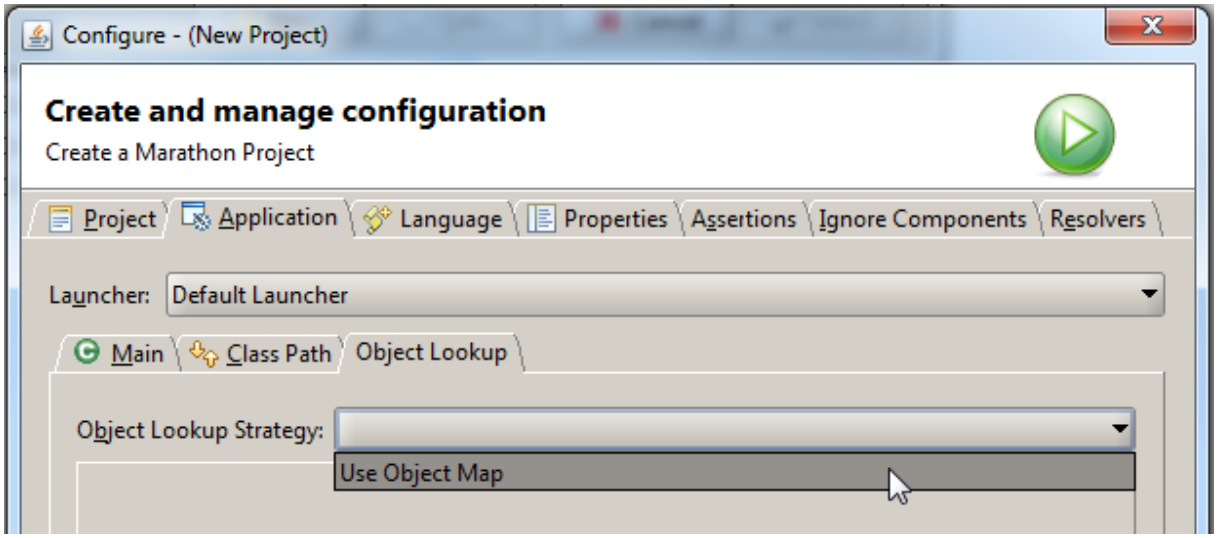
Unter dem Reiter „Class Path“ werden alle für das Projekt benötigten Bibliotheken (jar-Files) angegeben. Befindet sich die auszuführende Klasse nicht in einem dieser jar-Files, wird der Pfad zum Projekt angegeben. In diesem Beispiel ist es das bin-Verzeichnisses eines Eclipse-Projekts, dass über „Add Folders“ ergänzt wird.



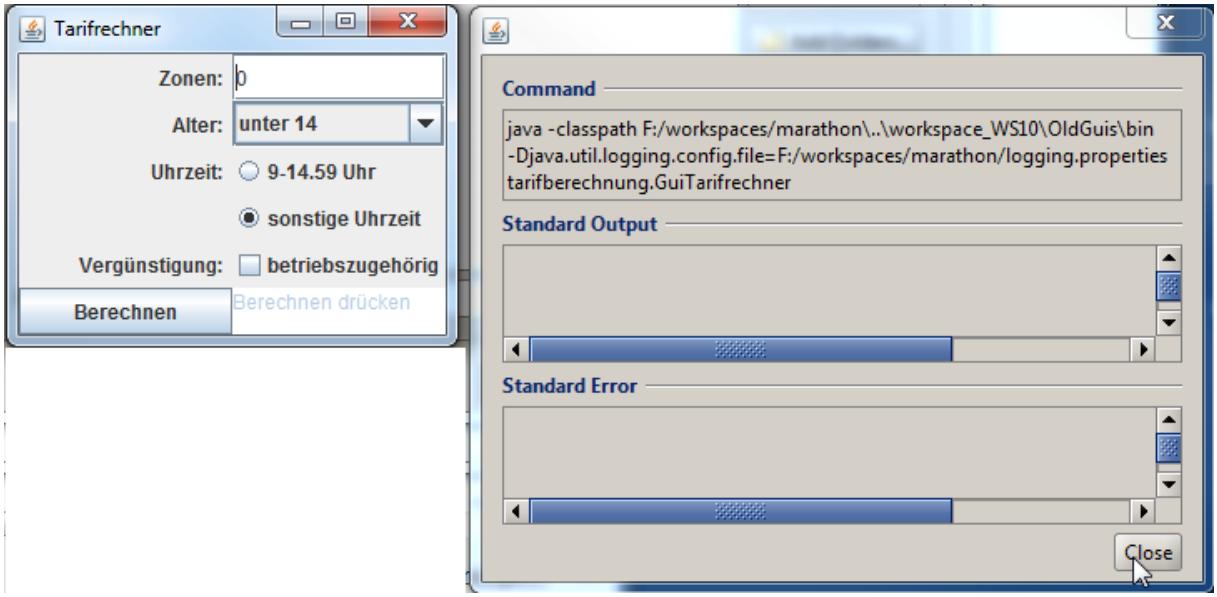
Über Browse wird dann das passende Verzeichnis gefunden.



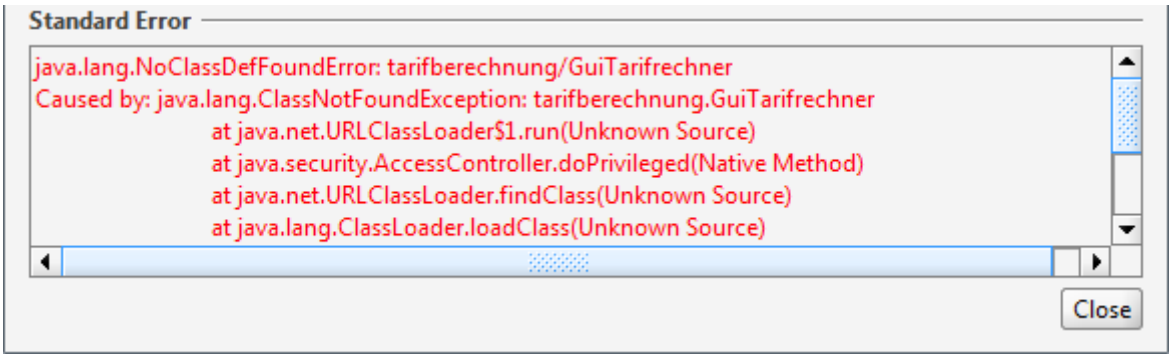
Nun wird im Reiter „Object Loop“ eine „Object Loop Strategy“, hier „Use Object Map“ ausgewählt.



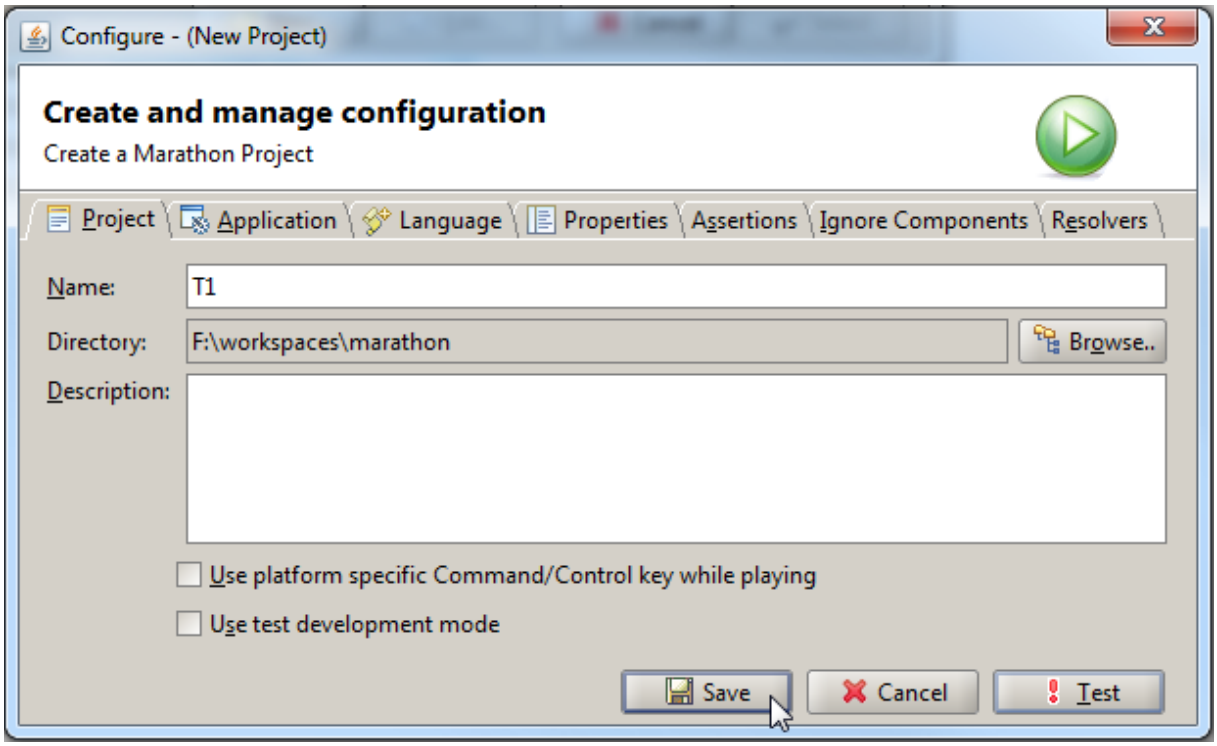
Danach kann über „Test“ festgestellt werden, ob das Programm erfolgreich gestartet wird. Bei Erfolg sieht man das Programmfenster und die von Marathon gesammelten Daten zum Start.



Sollte der Start fehlschlagen, erhält man mehr oder minder hilfreiche Hinweise. Im folgenden Beispiel wurde vergessen, das Verzeichnis bin mit anzugeben.

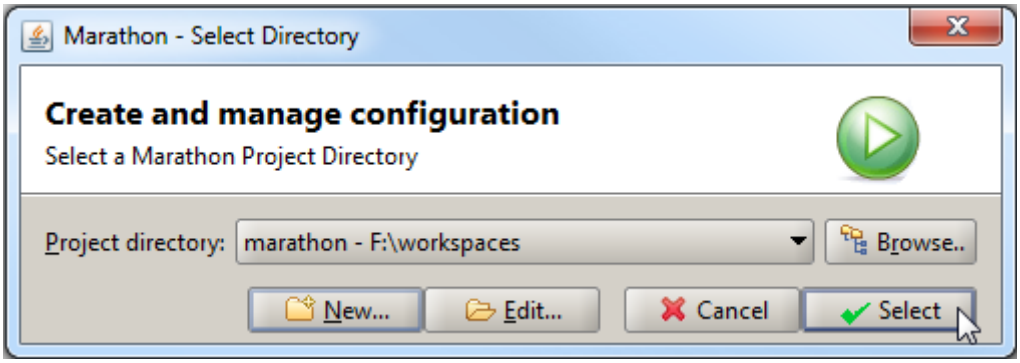


Nach erfolgreichem Test kann die erstellte Konfiguration mit „Save“ abgespeichert werden. Die Konfiguration wird zu den bisher vorhanden ergänzt.

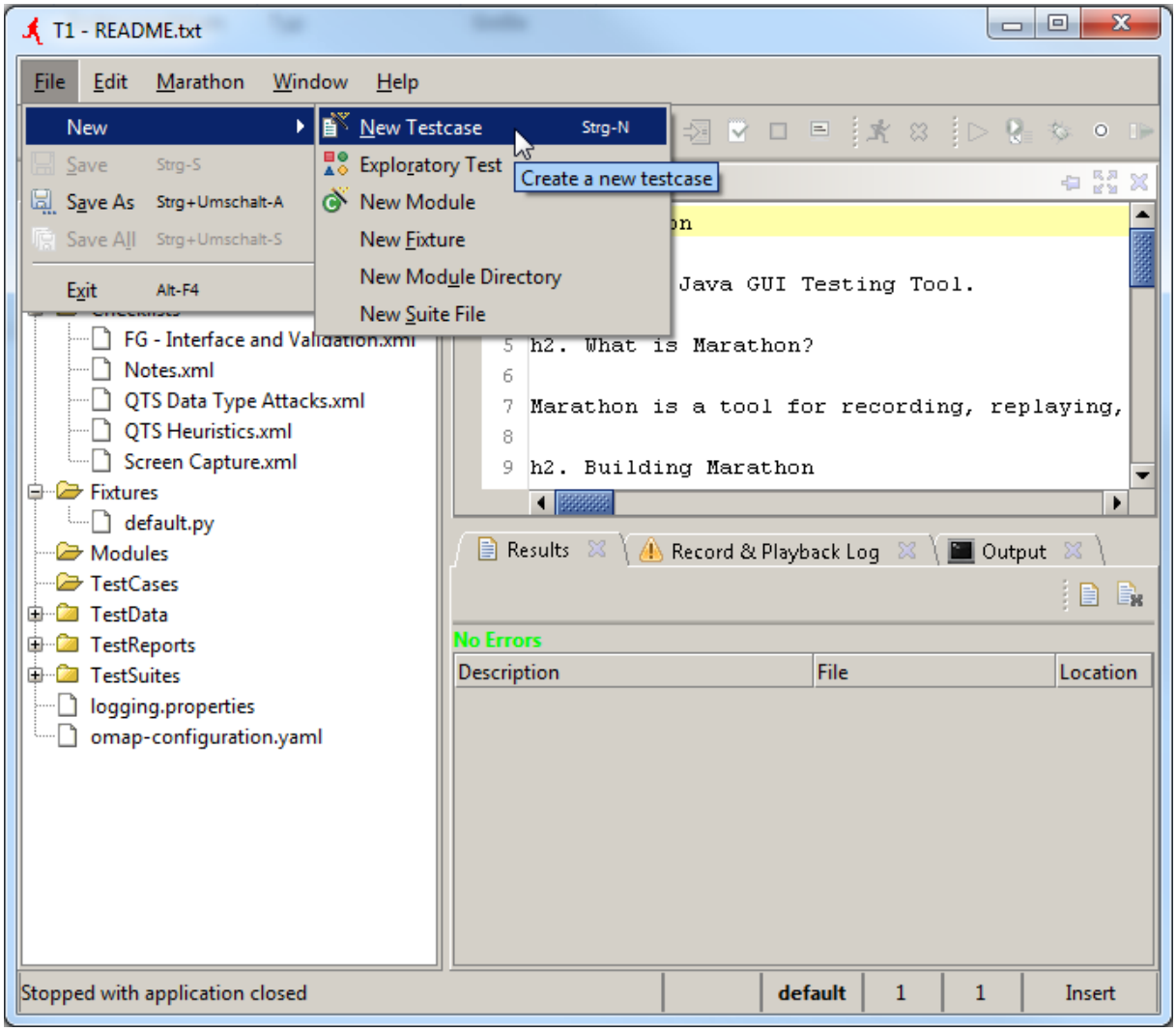


B.3 Aufzeichnung eines Skriptes

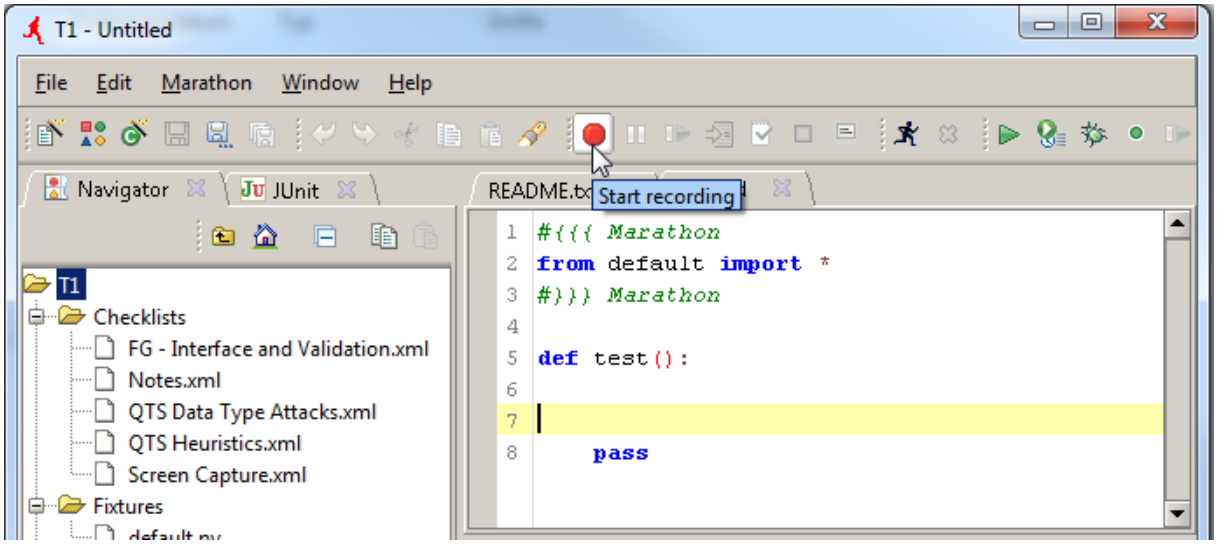
Über „Select“ kann die eigentliche Nutzung beginnen.



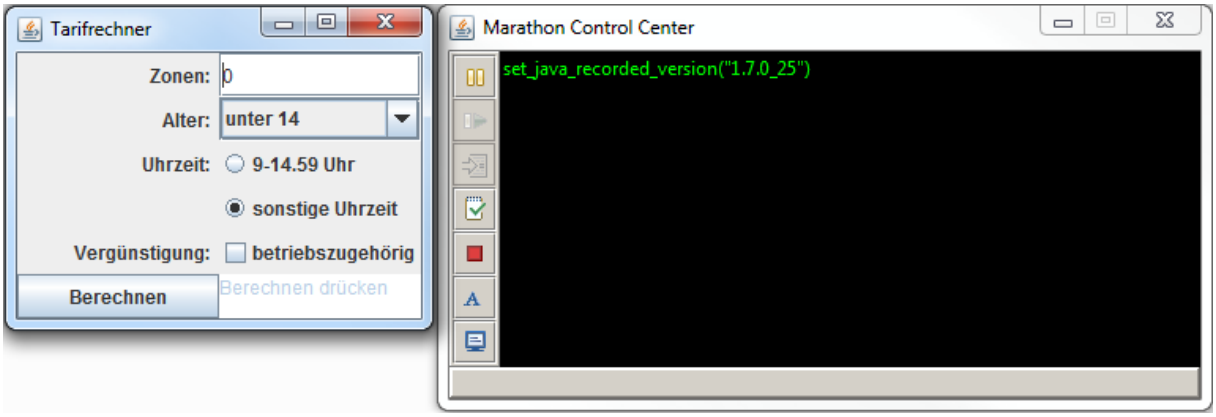
Zum Anlegen eines neuen Testes, genauer zunächst einer Aufnahme wird „File > New > New Testcase“ gewählt.



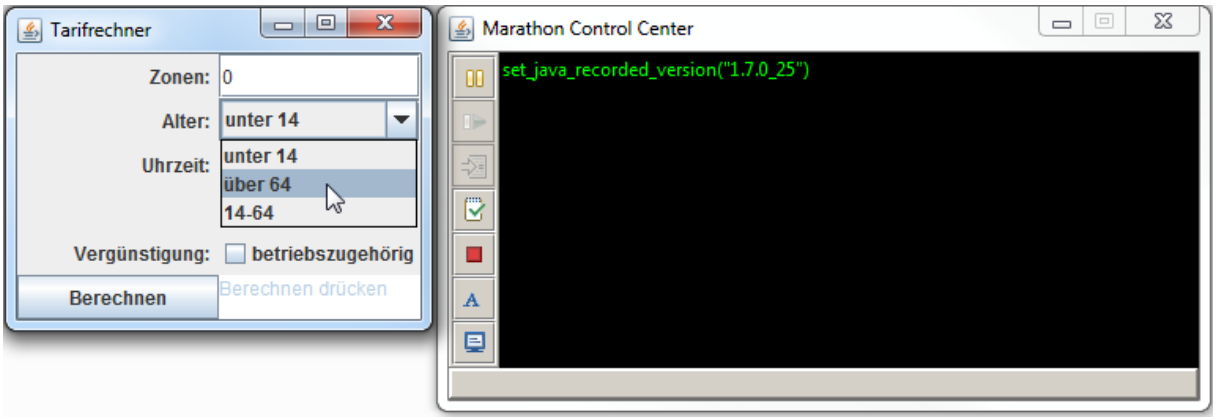
Über den roten Knopf kann die Aufnahme gestartet werden. Im unteren Reiter „Untitled“ wird das bisherige Skript angezeigt.



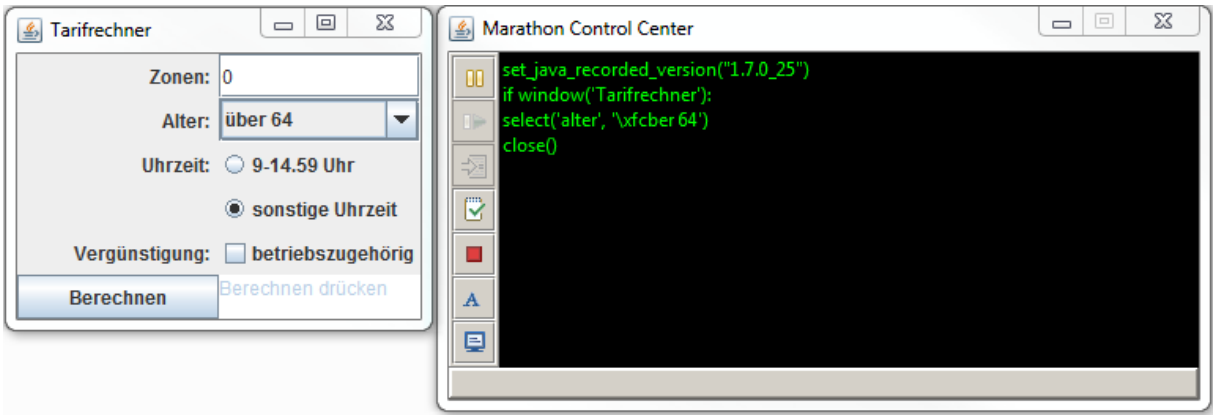
Nach Drücken des roten Knopfes startet die Applikation auf der linken Seite, auf der rechten Seite sieht man das „Marathon Control Center“, in dem die aufgezeichneten Aktionen gezeigt werden.



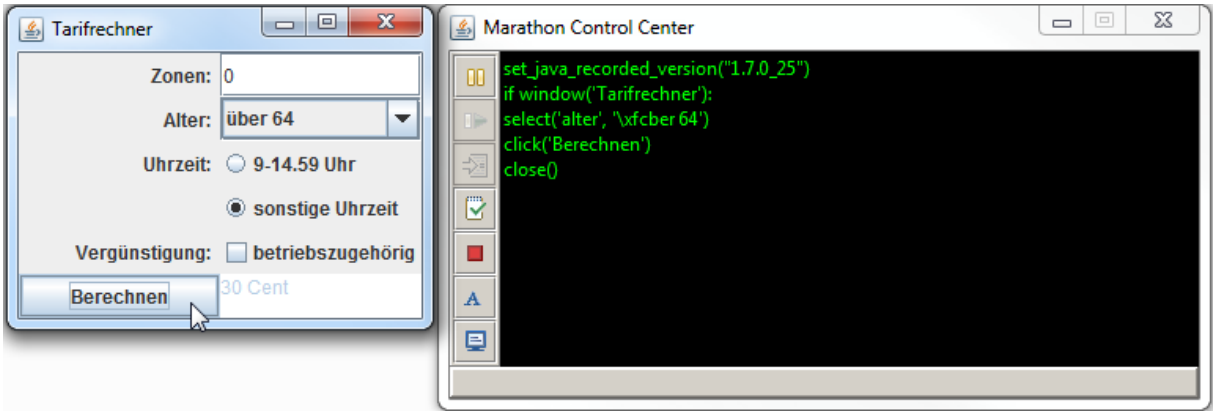
Das Auswahlfeld wird heruntergeklappt.



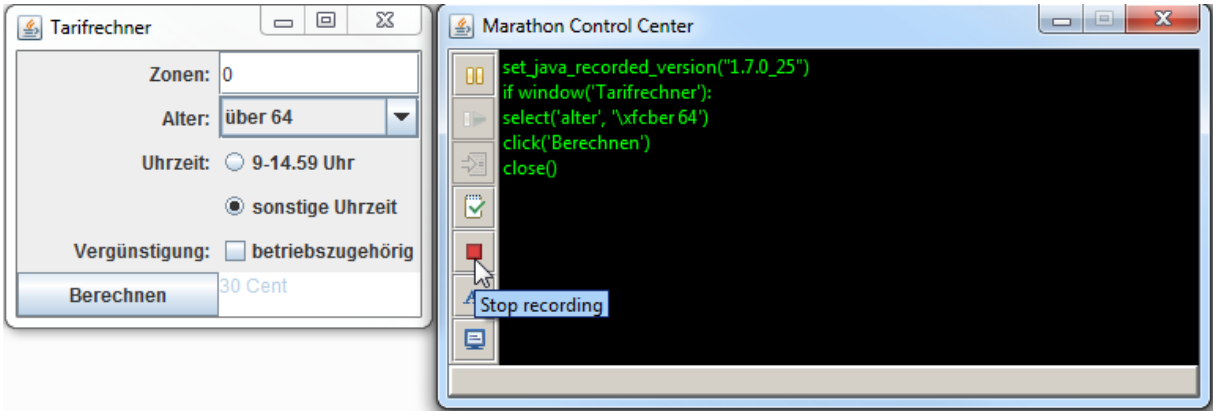
Auf der rechten Seite wird der passende Code angezeigt.



Es wird der Berechnen-Knopf geklickt.

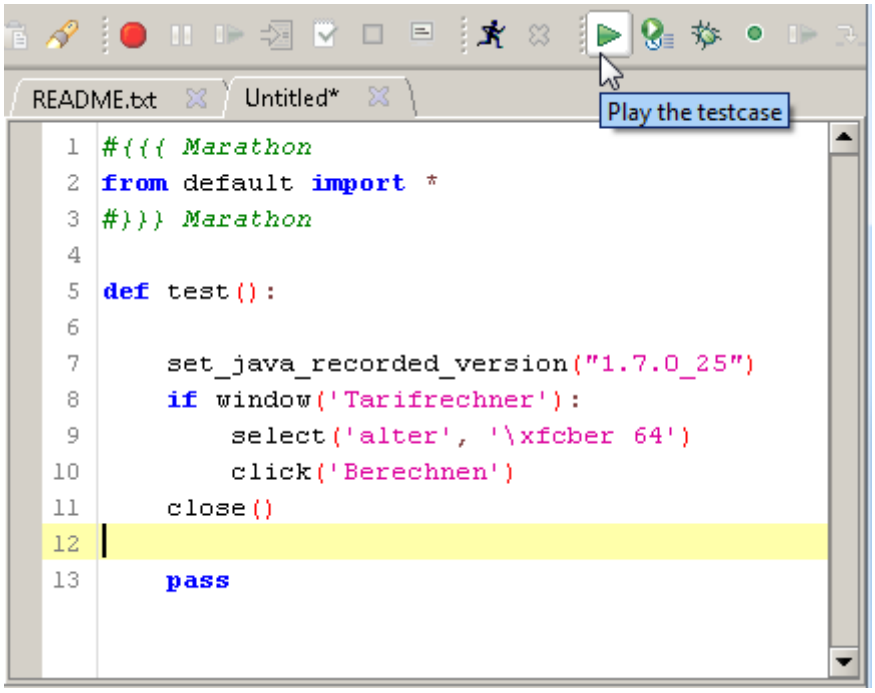


Die Aufzeichnung soll über das Control Center mit dem roten Quadrat und nicht über das Schließen der Applikation beendet werden, was bei späteren Ausführungen zu Problemen führen kann.

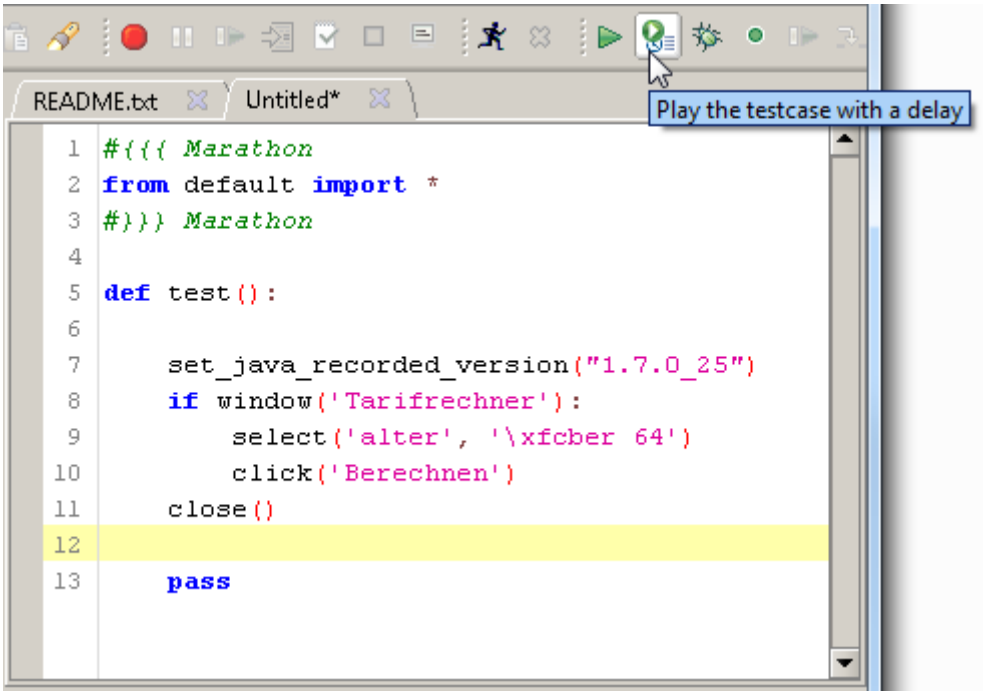


B.4 Abspielen eines Skriptes

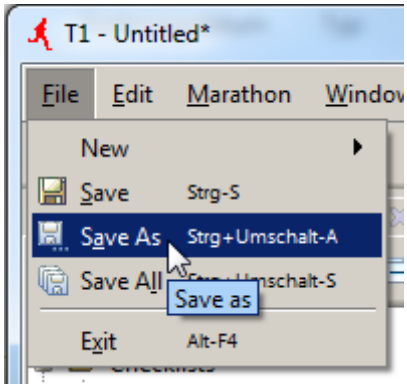
Man sieht das aufgezeichnete Skript, dass man über dem Play-Button jederzeit abspielen kann.



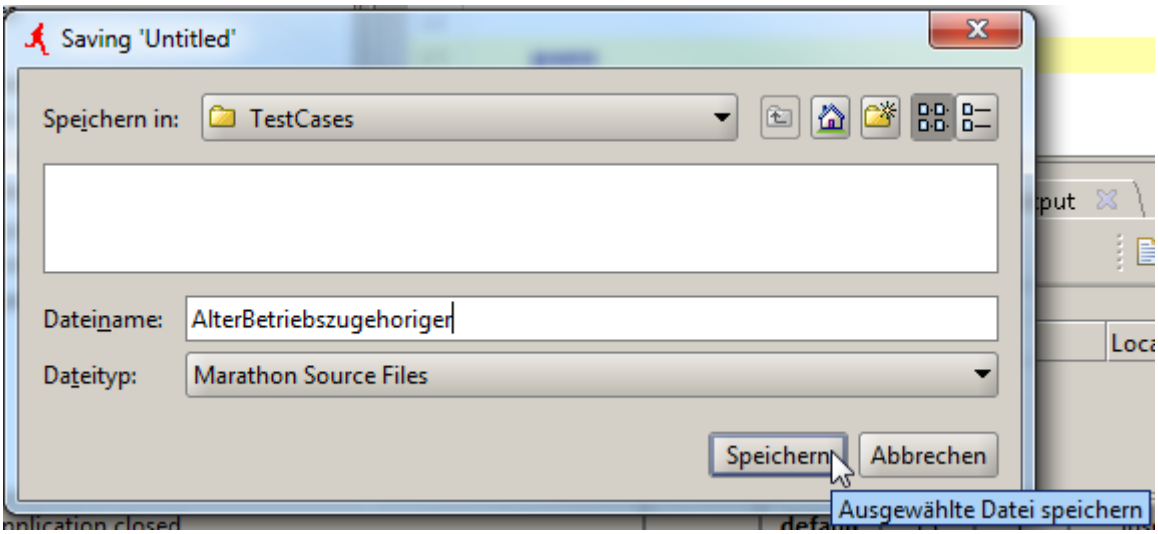
Rechts neben dem Play-Button befindet sich ein Knopf mit dem das Skript langsamer abläuft und bei dem man die Schritte im Scriptcode genau verfolgen kann.



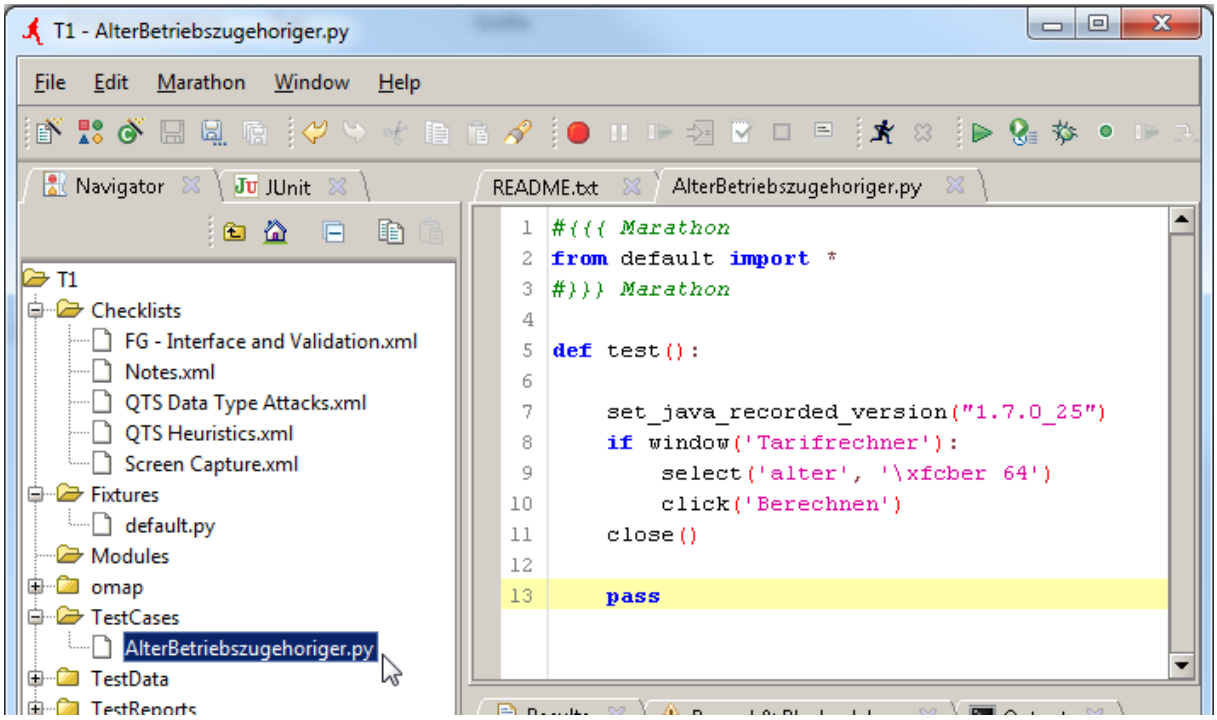
Generell müssen Skripte und „File > Save As“ abgespeichert werden, damit sie später nutzbar sind.



Der Name des Testfalls kann frei gewählt werden, sollte aber zum Ablauf passen.



Die vorhandenen Testfälle werden auch unter „TestCases“ im Navigator angezeigt.



B.5 Skripte zu Testfällen erweitern

Die Testfälle werden entweder in der Skriptsprache Jython oder in der Skriptsprache JRuby beschrieben. Zusicherungen müssen ebenfalls in diesen Sprachen ergänzt werden. Mit etwas Erfahrung kann man die Zusicherungen direkt in das Programmschreiben. Die typische Form ist dabei:

```
assert_p('<Name des GUI-Elements>', '<Property>', '<erwarteter Wert>')
```

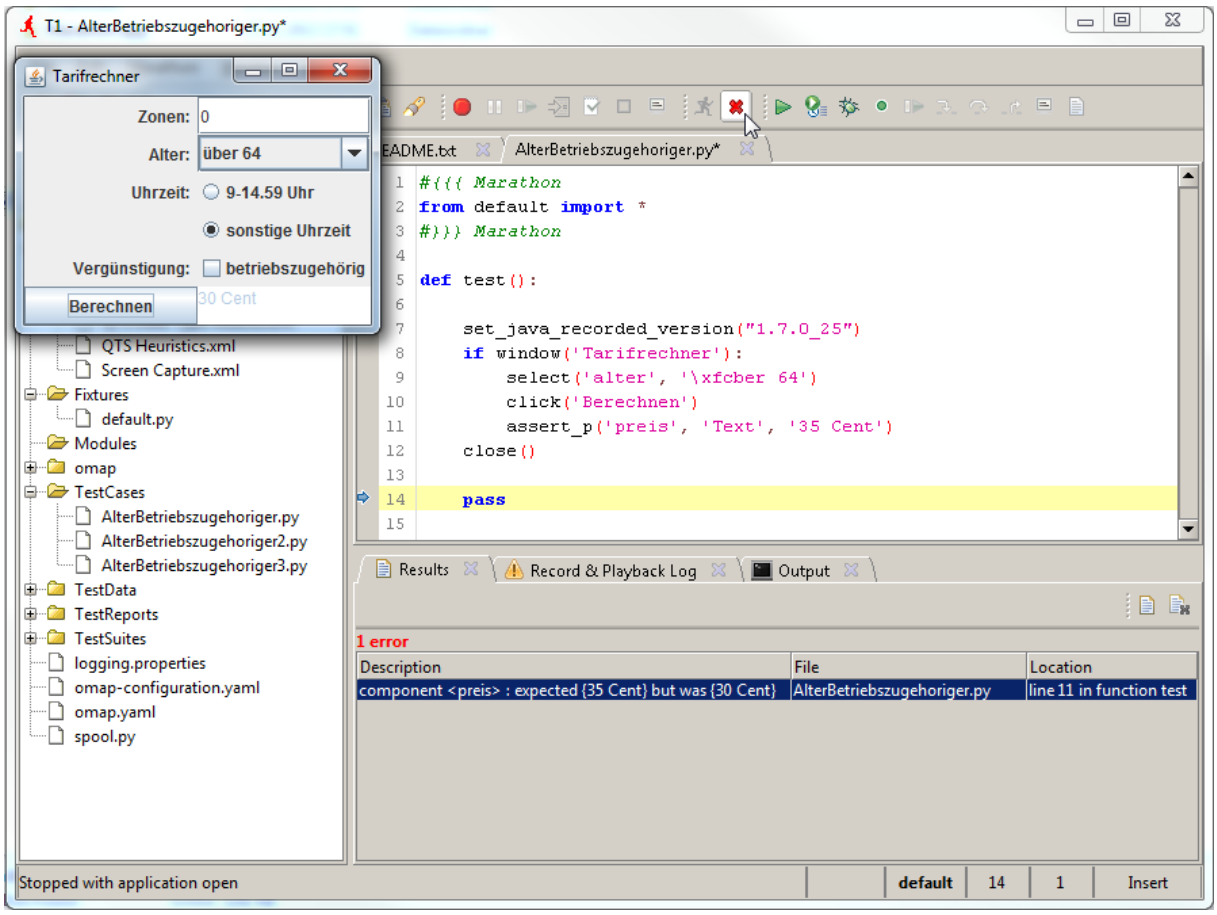
Bei Property (Eigenschaft) handelt es sich um eine Eigenschaft des GUI-Elements, die mit get und set verändert werden kann, eine typische Eigenschaft ist die Beschriftung, die über Text gelesen werden kann. Das bisher aufgezeichnete Skript kann durch die vorletzte Zeile ergänzt werden.

```
#{{{ Marathon
from default import *
#}}} Marathon

def test():

  set_java_recorded_version("1.7.0_25")
  if window('Tarifrechner'):
    select('alter', '\xfcber 64')
    click('Berechnen')
    assert_p('preis', 'Text', '30 Cent')
  close()

  pass
```

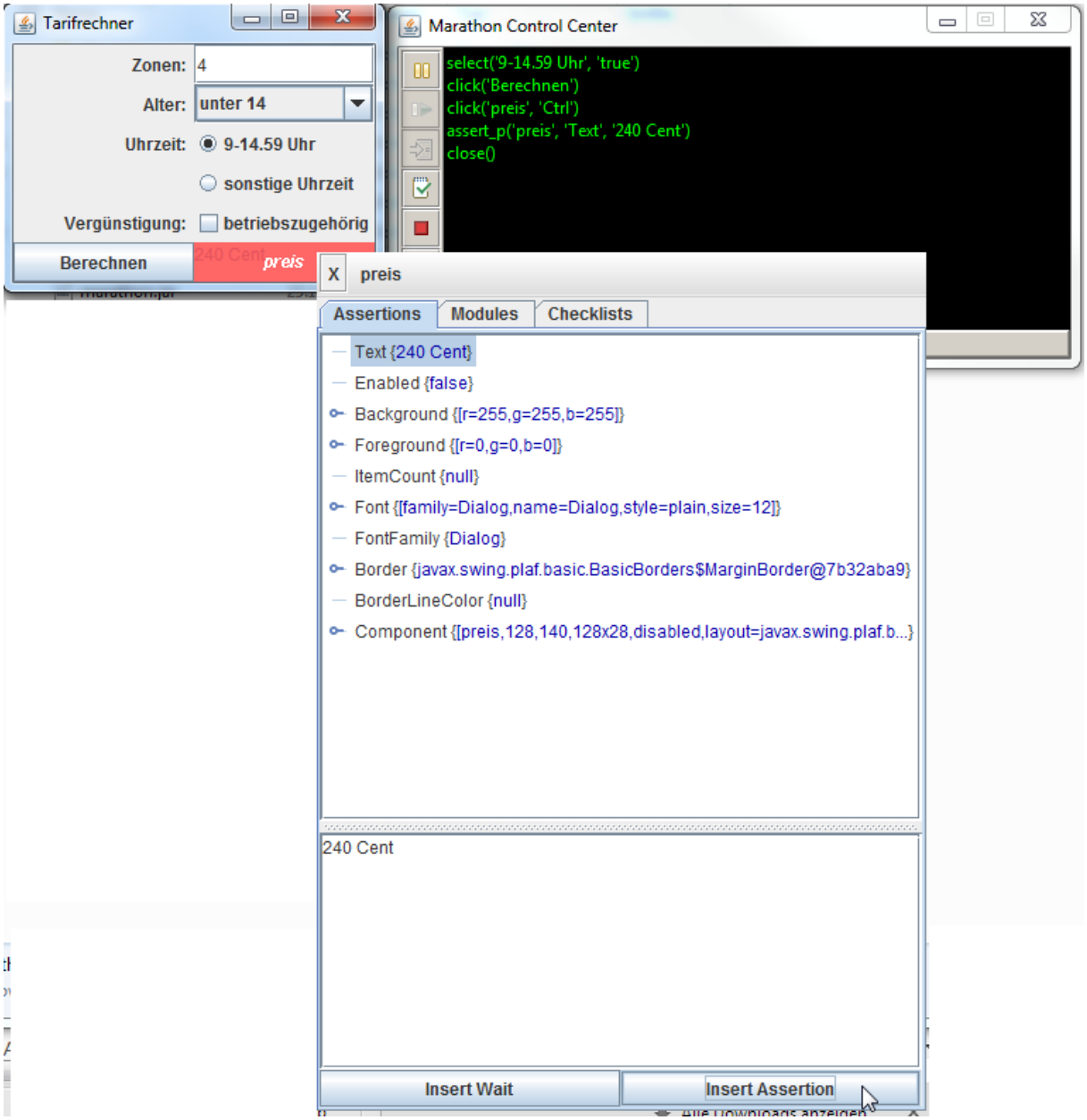
Der Test läuft über „Play“ erfolgreich durch. Ändert man die letzte Zeile in

```
assert_p('preis', 'Text', '35 Cent')
```

hält der Test mit der folgenden Fehlermeldung an. Wichtig ist, dass die geöffnete Applikation niemals über ihre Beendigungsmöglichkeit geschlossen wird. Es muss immer das kleine X von Marathon (siehe Mauszeiger) genutzt werden.

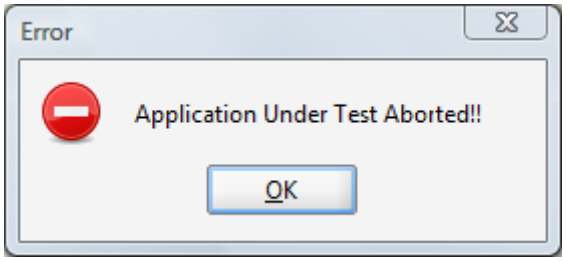


Die Zusicherungen können auch schon direkt während der Aufzeichnung ergänzt werden. Dazu lässt man das Programm zunächst laufen, will man dann eine Prüfung ergänzen, wird Strg+rechte Maustaste zusammen auf dem zu untersuchenden Element gedrückt. Alternativ ist auch Strg+F8 nutzbar. Man kann dann die passende Eigenschaft auswählen, ggfls. den Wert ändern und die Zusicherung über „Insert Assertion“ hinzufügen. Danach kann man weitere Zusicherungen ergänzen oder über „Close“ die Aufzeichnung fortführen. Man beachte, dass bei der Anwahl der Name des Element rot angegeben wird. Sollte kein Name gesetzt sein, wird ein interner Name genutzt.



Neben assert_p steht einem die gesamte Mächtigkeit von Jython zur Verfügung, um weitere Prüfungen zu definieren und so z. B. Verknüpfungen zwischen GUI-Elementen aufzubauen.

Leider zeigt die Benutzung, dass ab und zu keine Testfälle mehr neu erzeugt werden können, auch wenn die zu testende Applikation, genauer der Ablauf, ordnungsgemäß über das Control Center beendet wurde. Erhält man die folgende Fehlermeldung sollte man alle Daten sichern und die Applikation neu starten.



Generell hat Marathon eine sehr gute Dokumentation, die weiteren Einblick in wichtige Möglichkeiten von Marathon bietet. Dazu gehören u. a.

- Festlegung von Test-Fixtures, die beschreiben was vor und nach Tests immer ausführt werden soll
- Module, Programmstücke, die in andere Skripte eingebaut werden können und so eine Strukturierung der Testfälle ermöglichen
- Nutzung eines Debuggers, um ab einem Breakpoint die Ausführung Schritt für Schritt erfolgen zu lassen und ggfls. Objektwerte zu verändern

C Kurzhinweise für weitere Werkzeuge (alt)

Die folgenden Werkzeuge könnten auf dem HS-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.

C.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.

C.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.

C.3 Jester

Jester ist ein JUnit-Testtester, mit dem die Genauigkeit von Junit 3-Tests analysiert werden kann.

C.4 Ganttproject

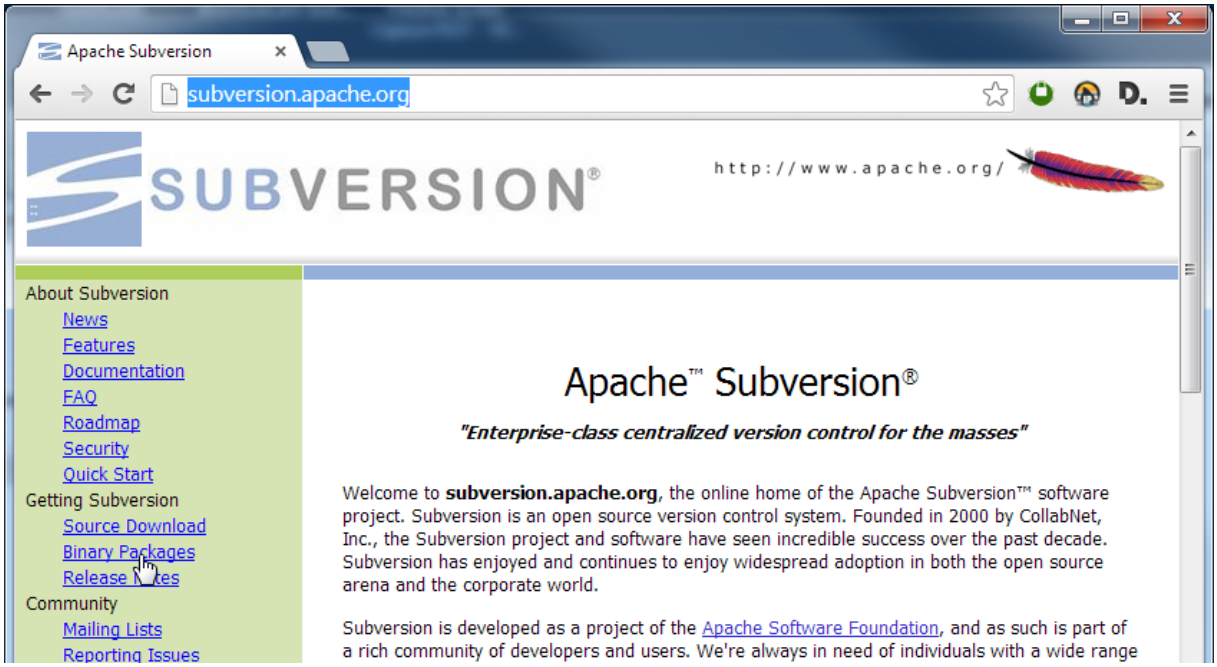
Ganttproject (Installation: ganttproject-2.0.3.exe) kann bei der Projektplanung hilfreich sein, da man Projekte auf Phasen aufteilen, Abhängigkeiten zwischen den Phasen spezifizieren und den Phasen Ressourcen zuordnen kann. Das Werkzeug ist bei Weitem nicht so mächtig wie andere Managementwerkzeuge,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

D Subversion (1.8.5)

Subversion ist ähnlich wie Git und 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.
Zur SVN-Nutzung gibt es ein freies, gut geschriebenes Buch unter http://svnbook.red-bean.com, dessen Lektüre zumindest der ersten drei Kapitel für ein Projekt unerlässlich ist.

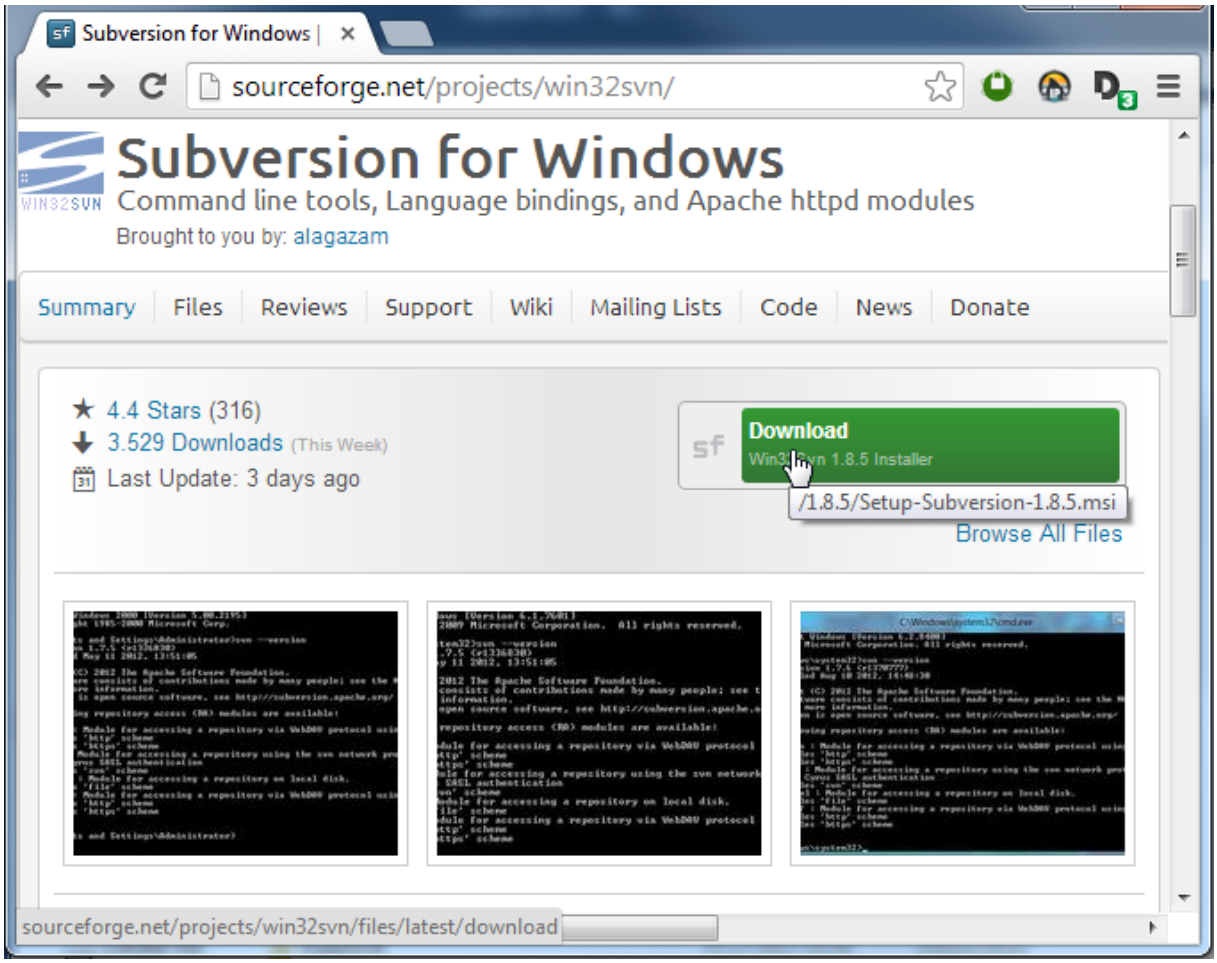
D.1 Subversion-Installation

Die aktuelle Quellseite dieses Werkzeugs ist http://subversion.apache.org/ (früher http://subversion.tigris.org), die ausführbare Dateien zum Download anbietet.

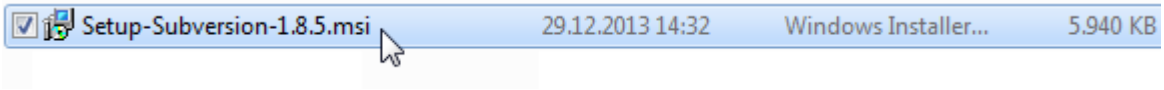


Man wählt die passende Version aus und wird zur Download-Seite weitergeleitet.

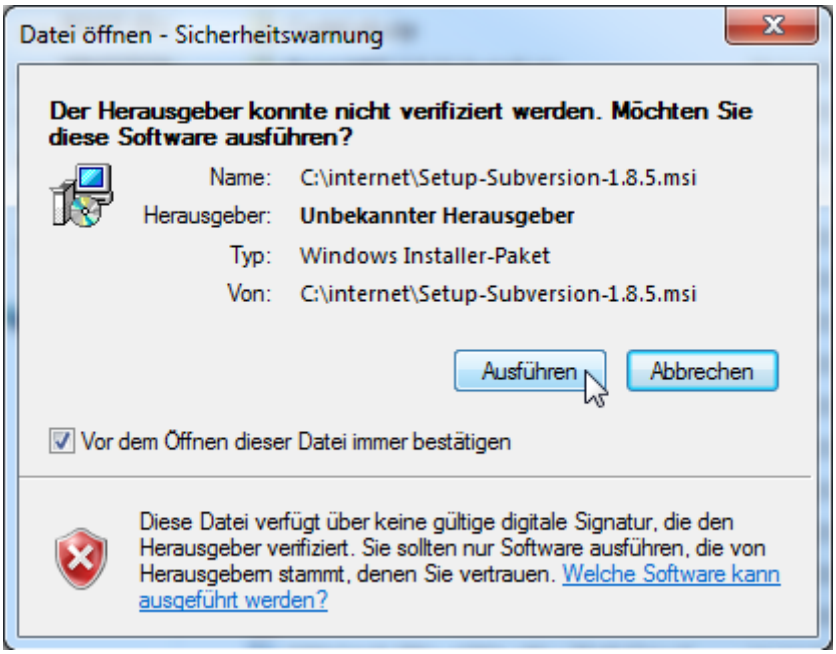
Man kann den Installer nutzen, der direkt unter Download angeboten wird.



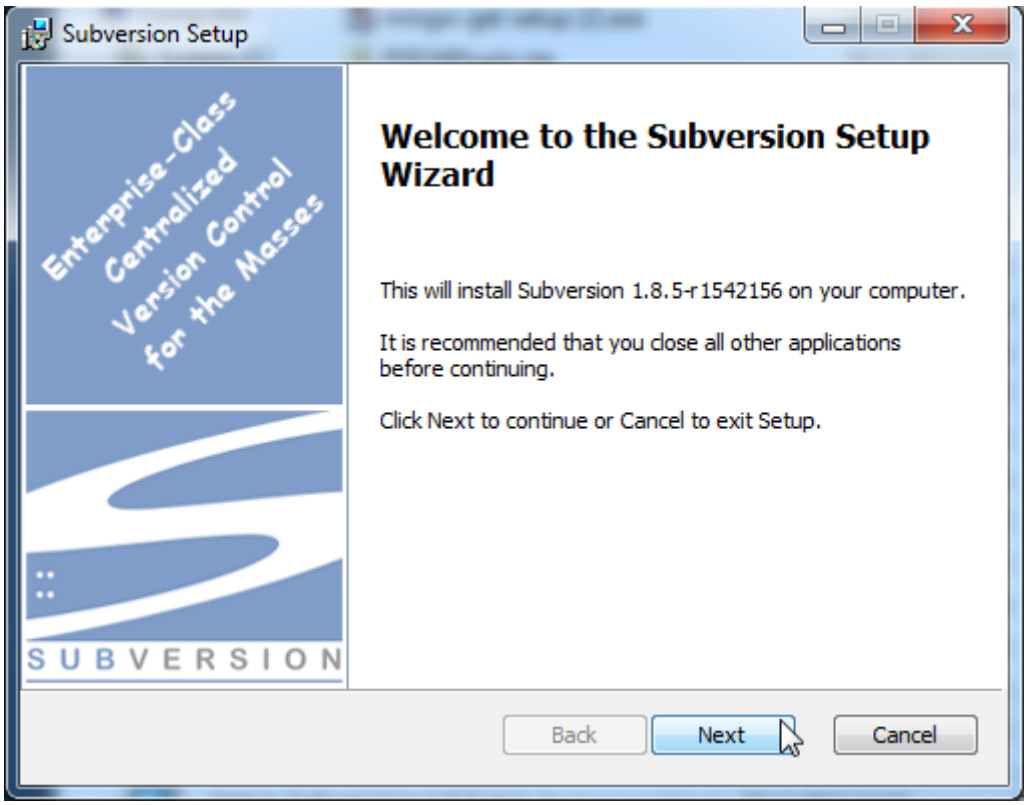
Mit einem Doppelklick auf der Datei wird das Programm installiert.

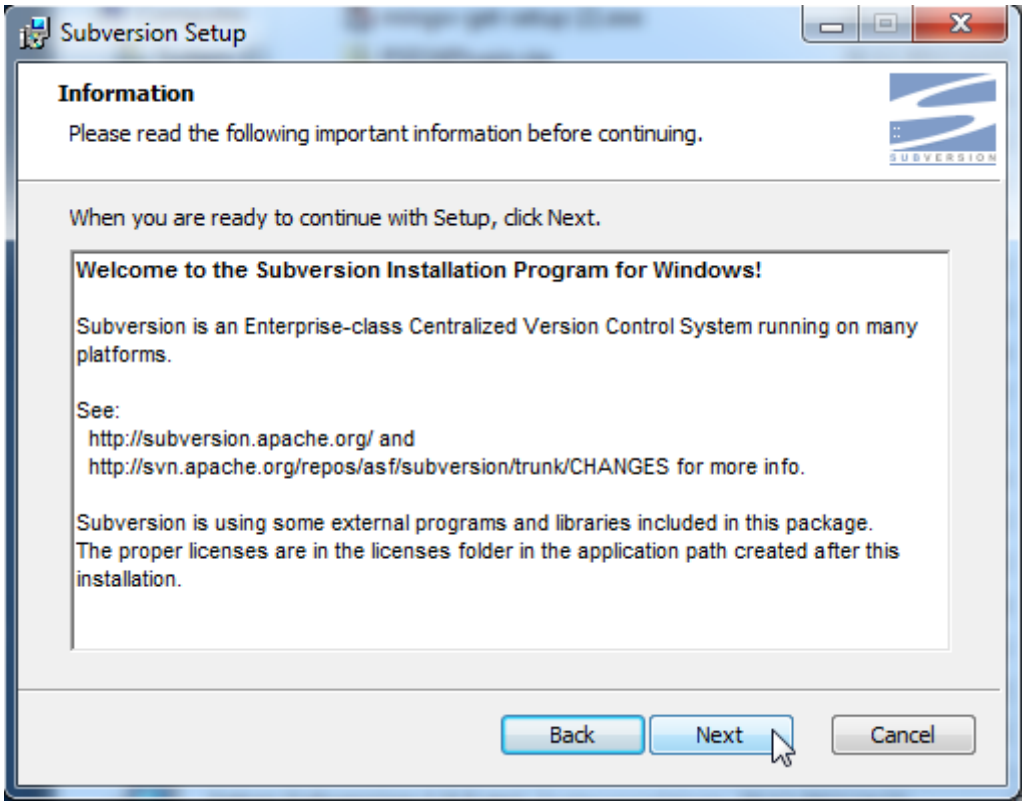


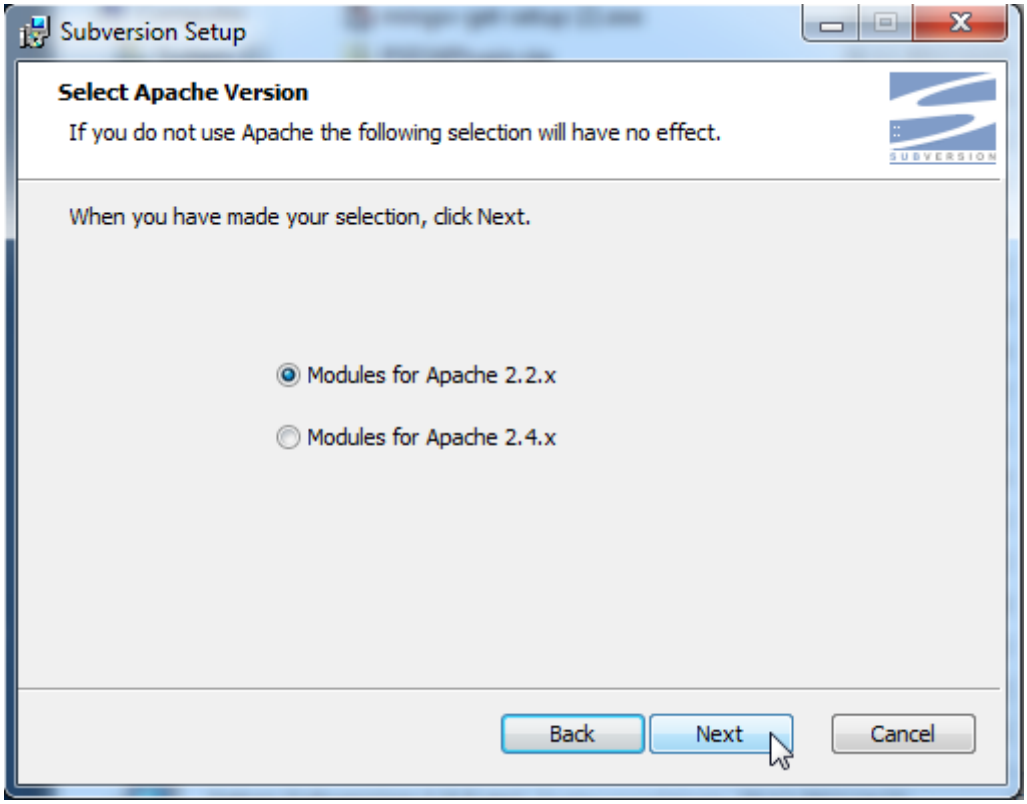
Auf Abfragen von Sicherheitsprogrammen wird hier nicht im Detail eingegangen.

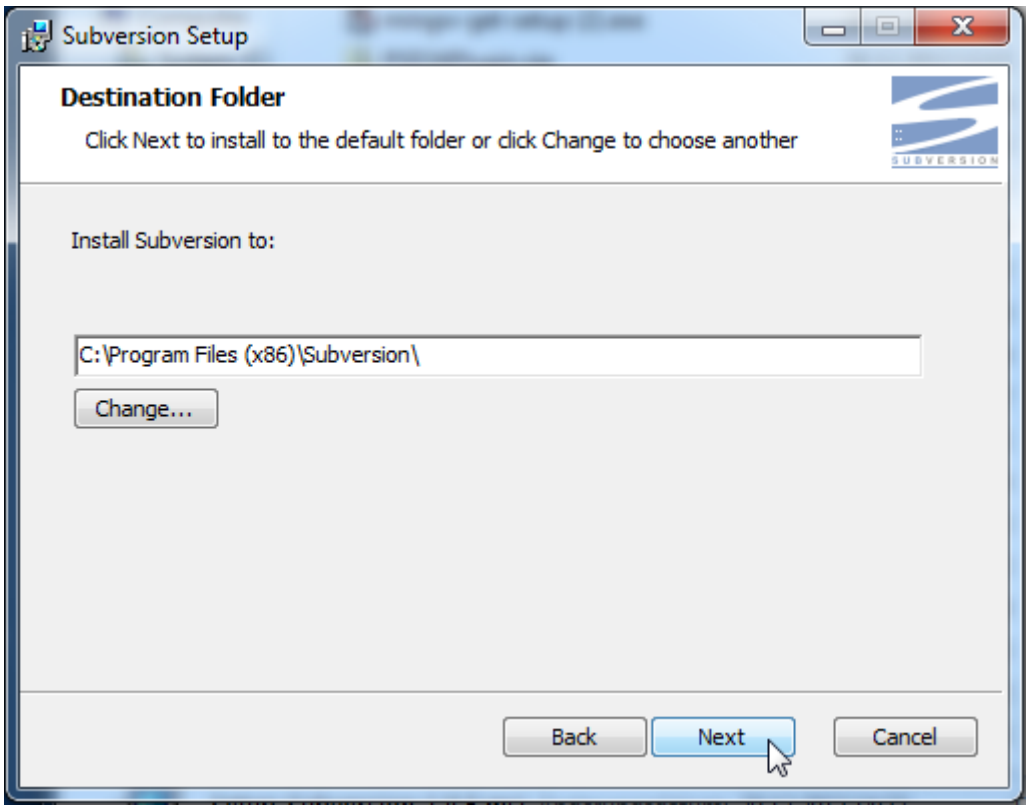


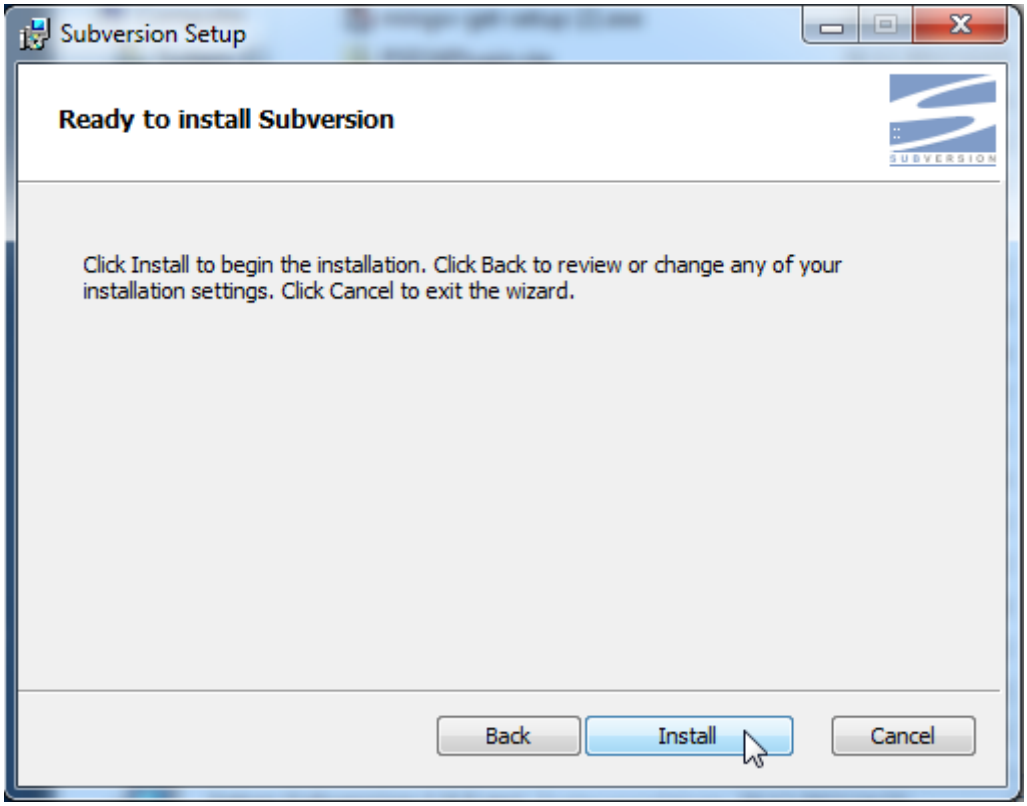
Die nachfolgenden Installationsbildschirme können mit „Next“ und „Install“ einfach durchlaufen werden.

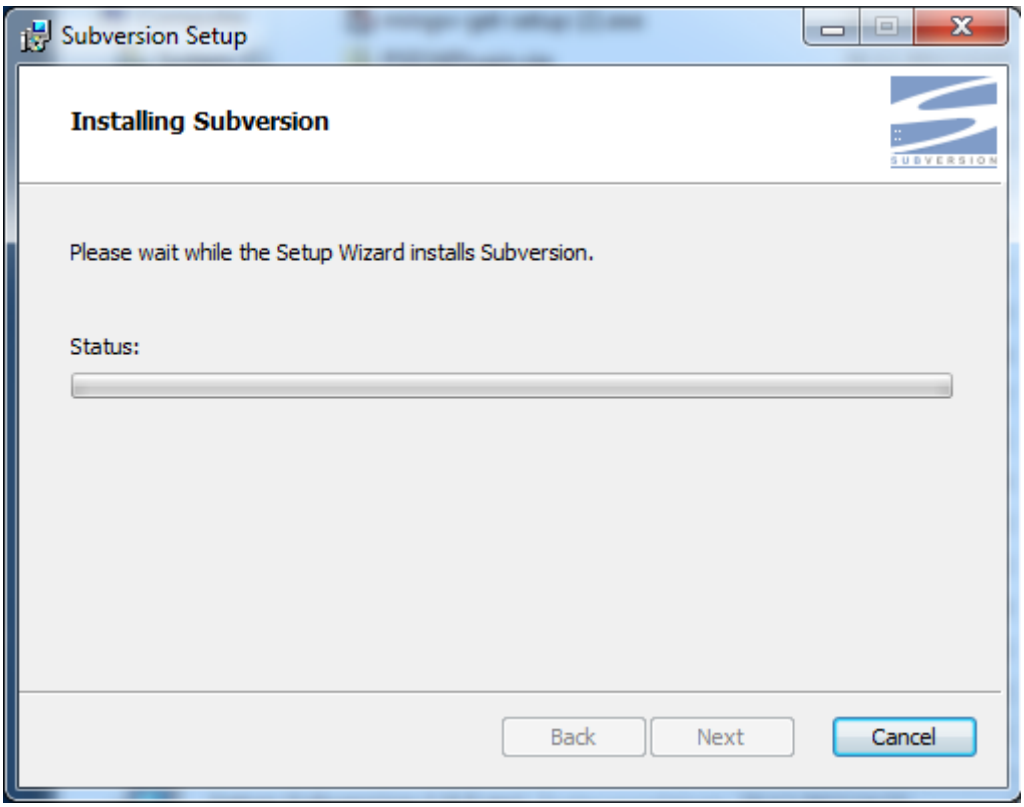


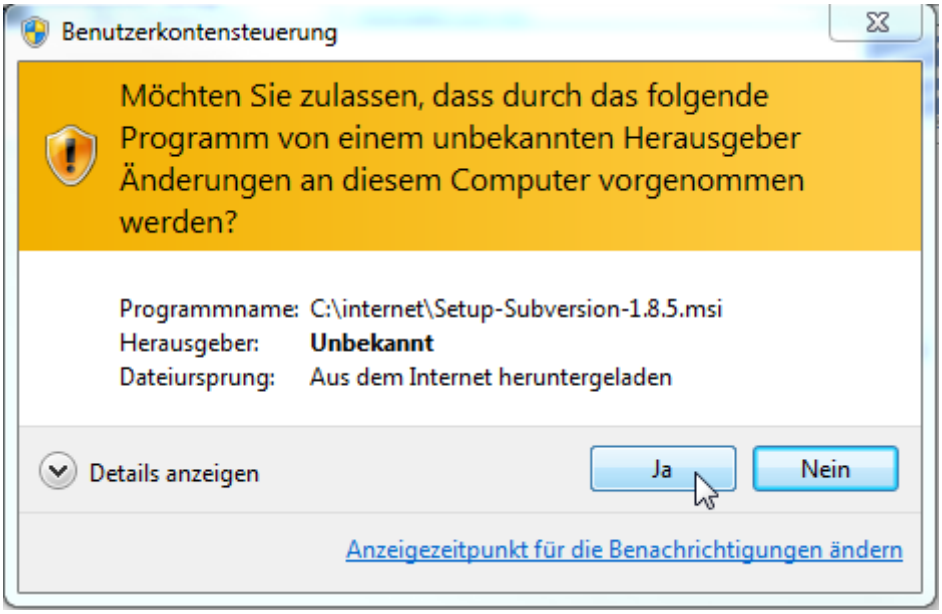


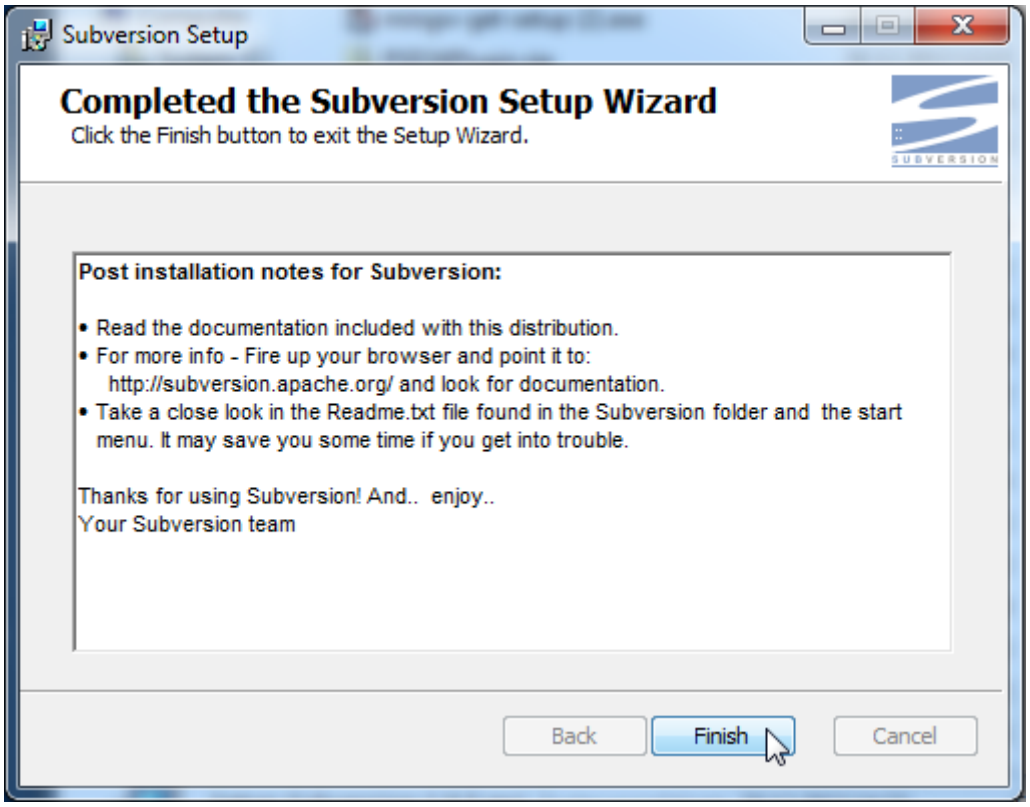

Subversion Setup
Select Apache Version
If you do not use Apache the following selection will have no effect.
When you have made your selection, click Next.
Modules for Apache 2.2.x
Modules for Apache 2.4.x
Back
Next
Cancel


Subversion Setup
Destination Folder
Click Next to install to the default folder or click Change to choose another
Install Subversion to:
C:\Program Files (x86)\Subversion\
Change...
Back
Next
Cancel

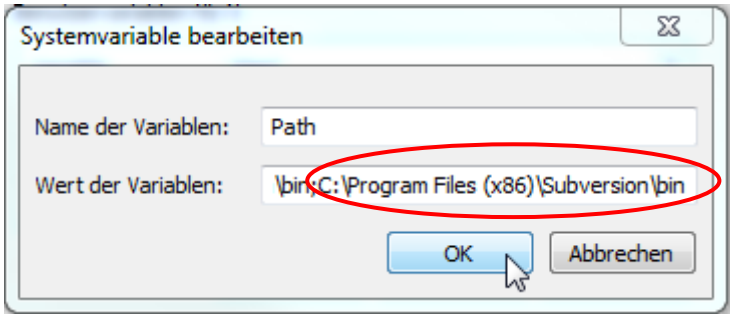

Subversion Setup
Ready to install Subversion
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


Subversion Setup
Installing Subversion
Please wait while the Setup Wizard installs Subversion.
Status:
Back
Next
Cancel

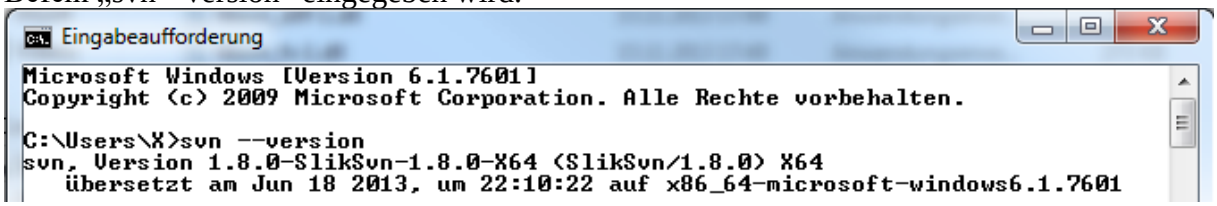




Bei der Installation wird das zugehörige bin-Verzeichnis C:\Program Files (x86)\Subversion\bin in die Pfadvariable eingetragen. Falls cygwin auf dem Rechner installiert ist, muss der Subversion-Pfad vor dem cygwin-Pfad stehen, da cygwin typischerweise bereits eine veraltete svn-Version enthält. Das Vorgehen zur Bearbeitung der Path-Variable wurde bereits ab Seite **Fehler! Textmarke nicht definiert.** beschrieben.



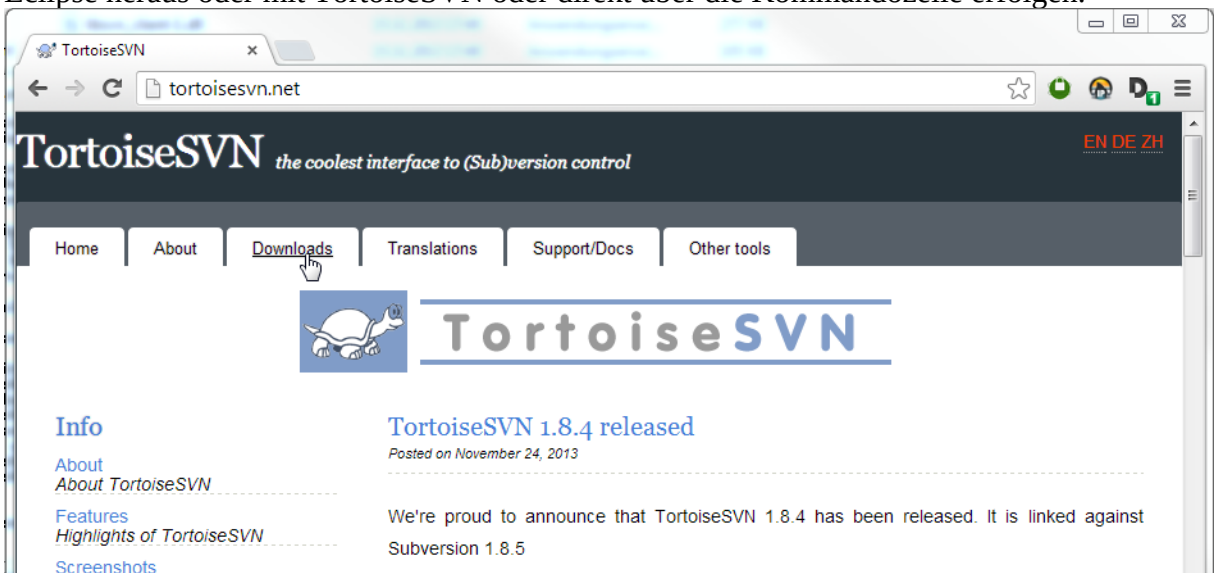
Die Installation kann in einer DOS-Box, bzw. einer cmd-Shell überprüft werden, indem der Befehl „svn --version“ eingegeben wird.



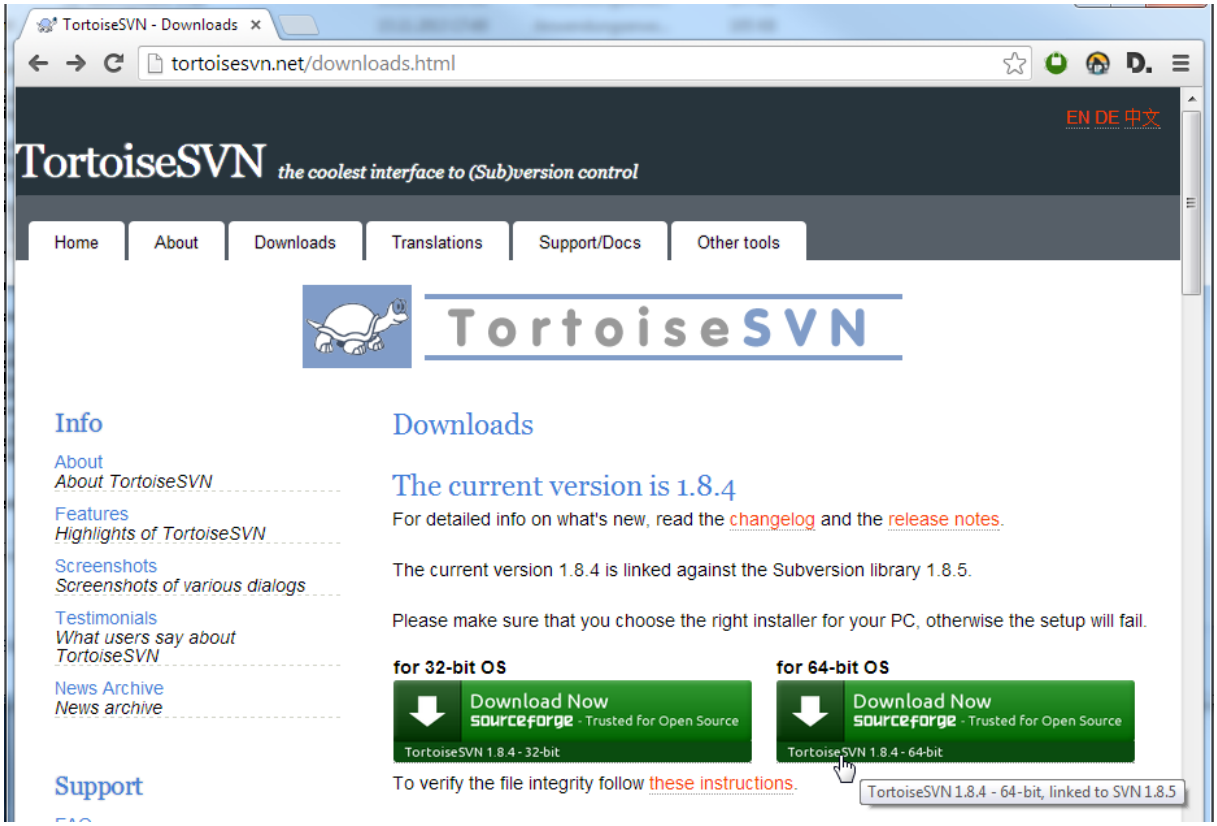
Danach kann Subversion wie in der Veranstaltung beschrieben genutzt werden.

D.2 Nutzung von TortoiseSVN

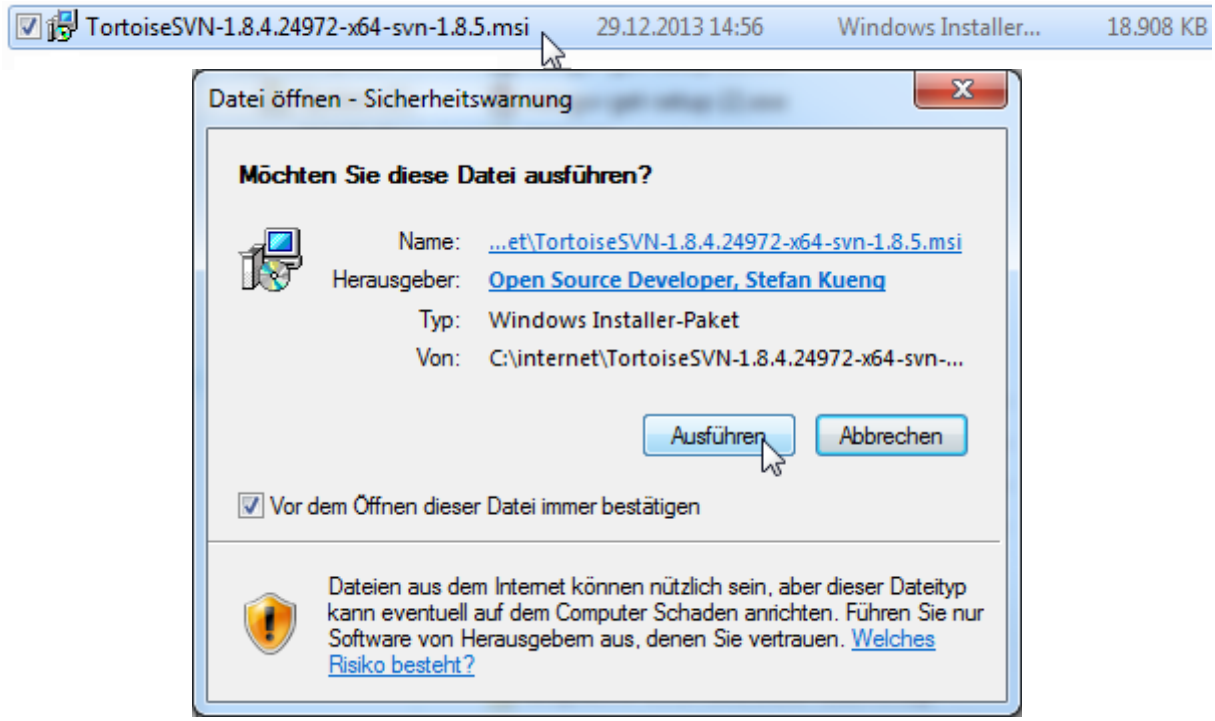
Wer nicht gerne auf der Kommandozeile arbeitet, kann die Subversion-Erweiterung TortoiseSVN von der Web-Seite http://tortoisesvn.net/ nutzen. Dieses Programm integriert die Fähigkeiten von Subversion direkt in den Browser und kann sehr hilfreich beim Umgang mit Dateien sein, die nicht durch Eclipse verwaltet 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.



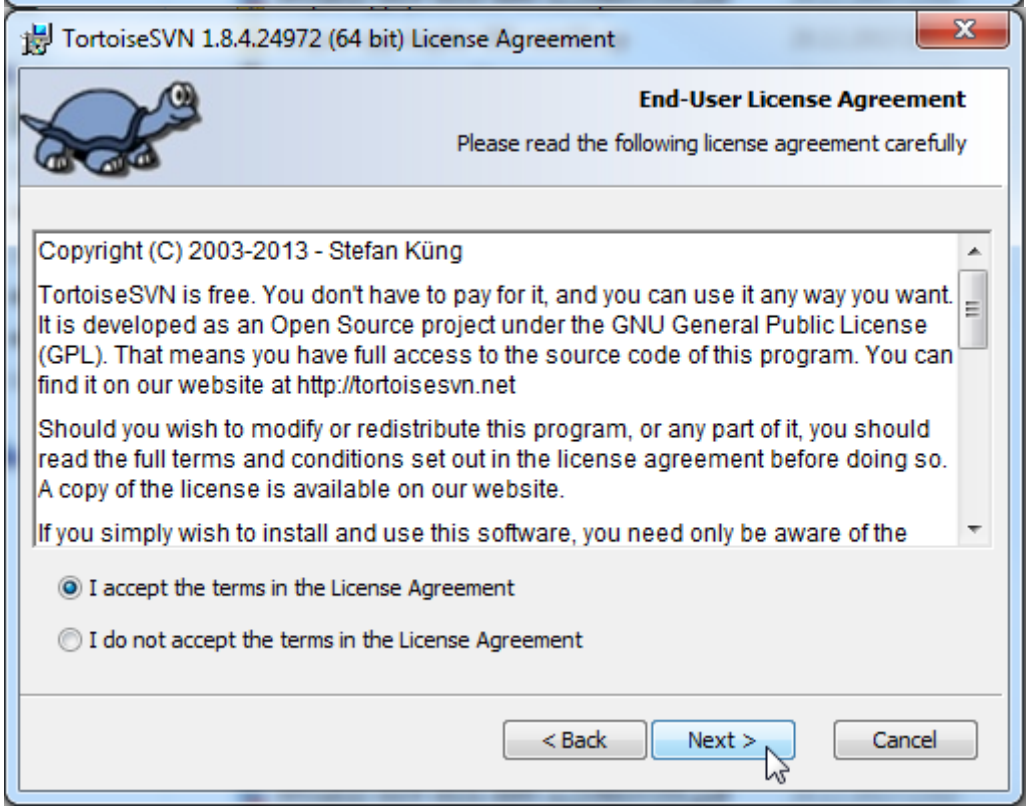
Man wählt den passenden Download aus, der dann startet.

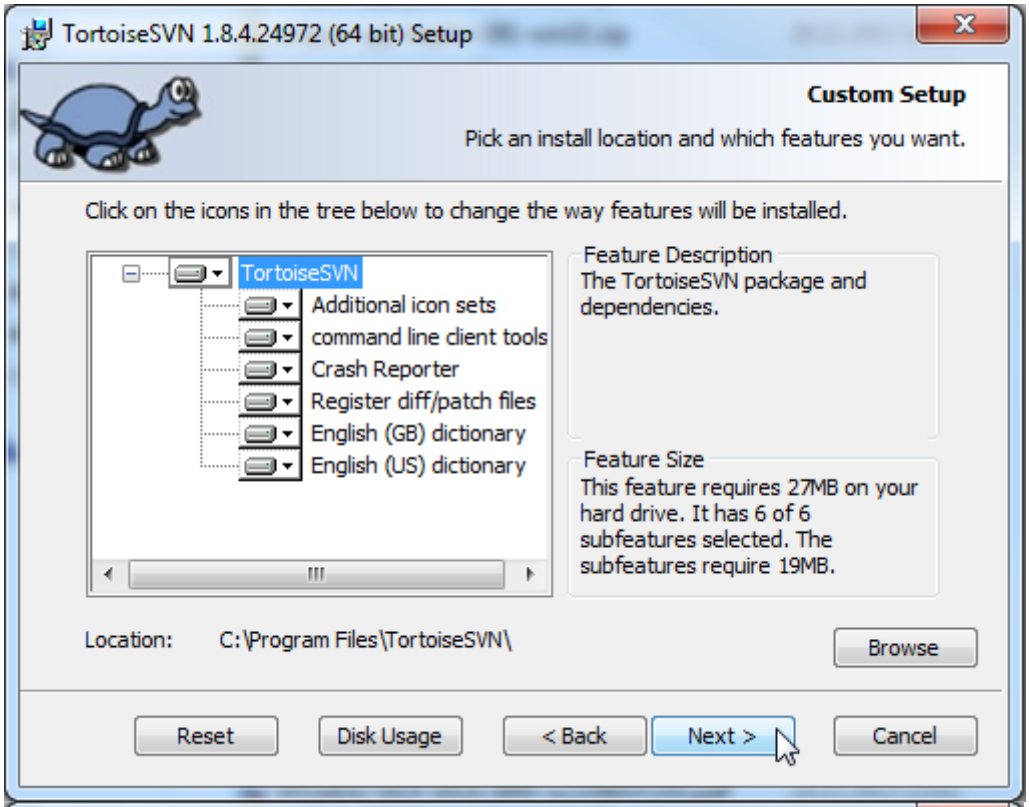


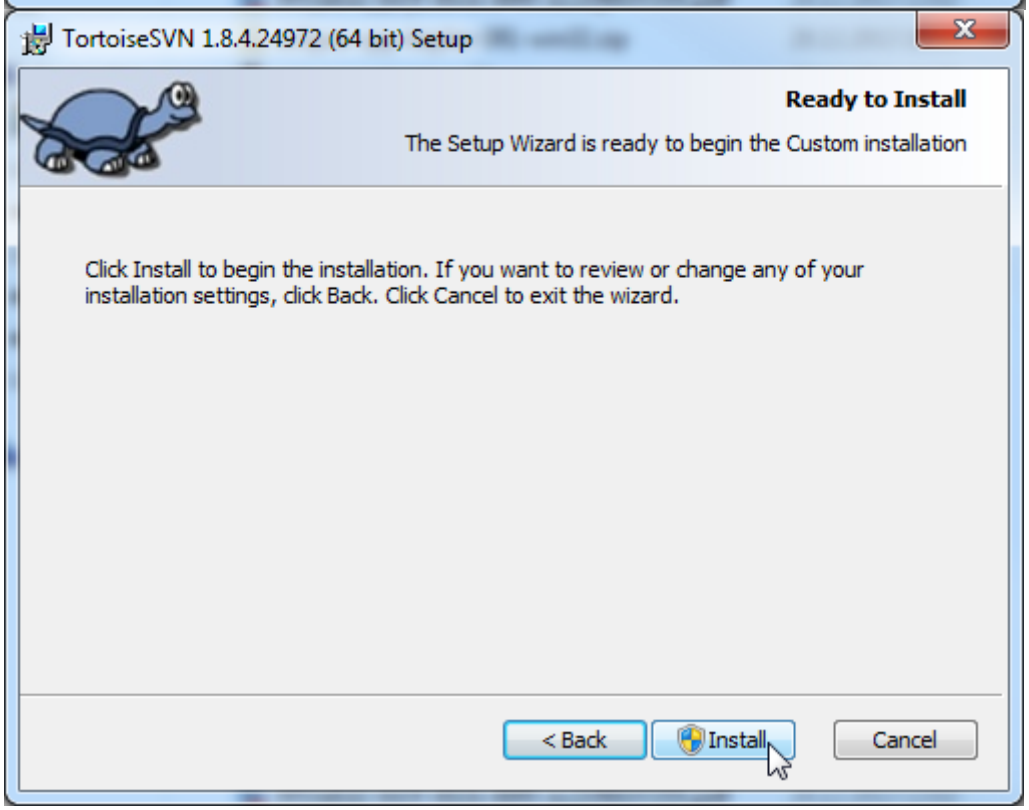
Die Installation beginnt mit einem Doppelkick auf der Datei, die nachfolgenden Schritte sind selbsterklärend.

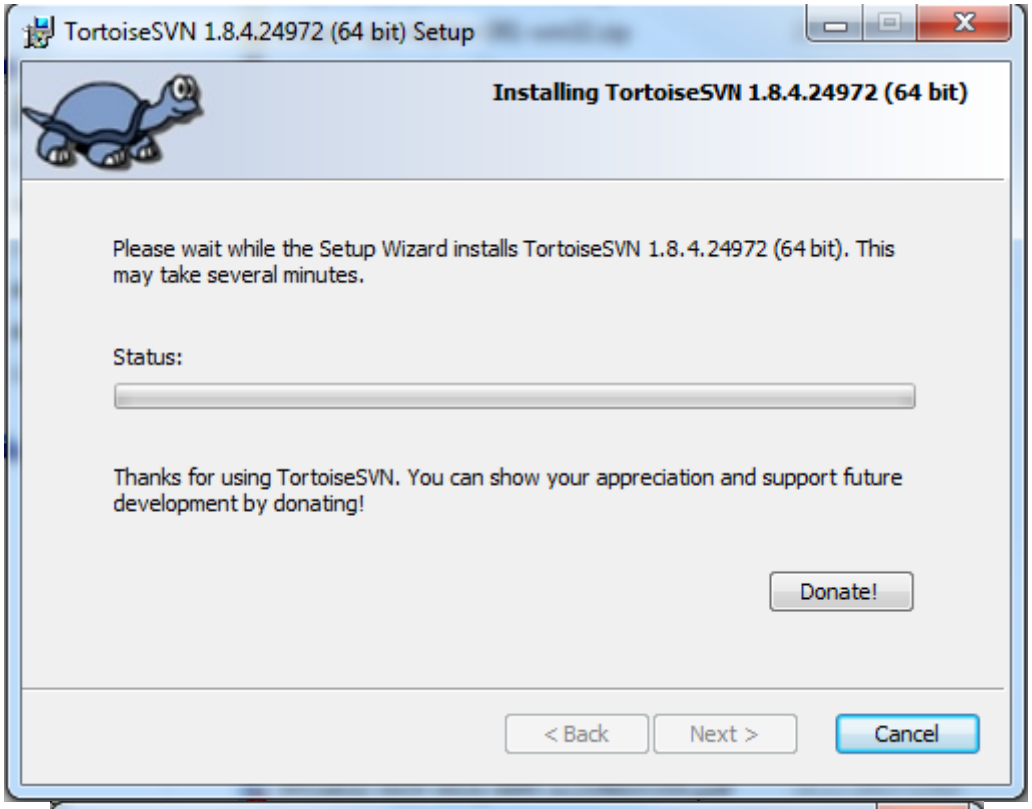


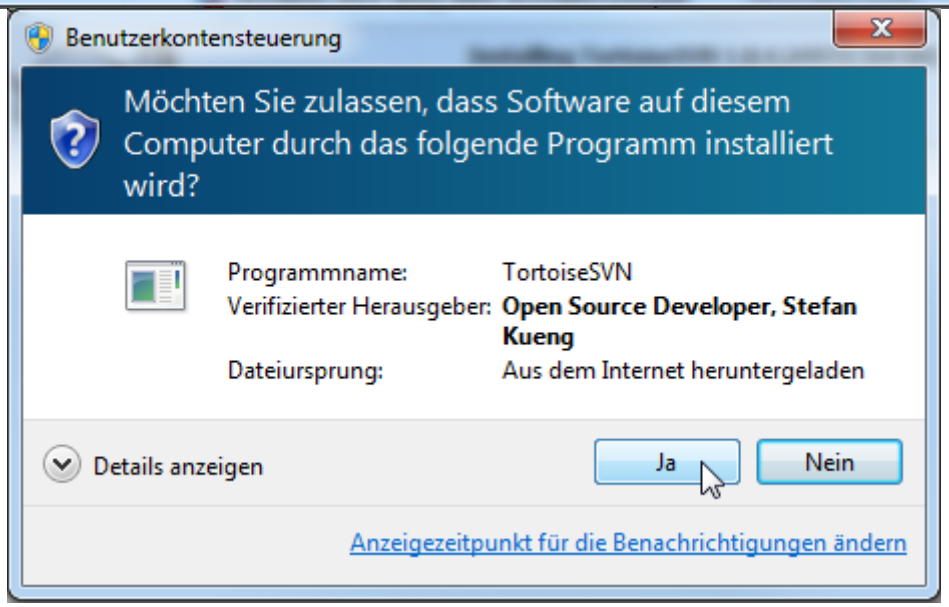

TortoiseSVN 1.8.4.24972 (64 bit) Setup
TortoiseSVN
Welcome to the TortoiseSVN 1.8.4.24972 (64 bit) Setup Wizard
The Setup Wizard will install TortoiseSVN 1.8.4.24972 (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
< Back
Next >
Cancel

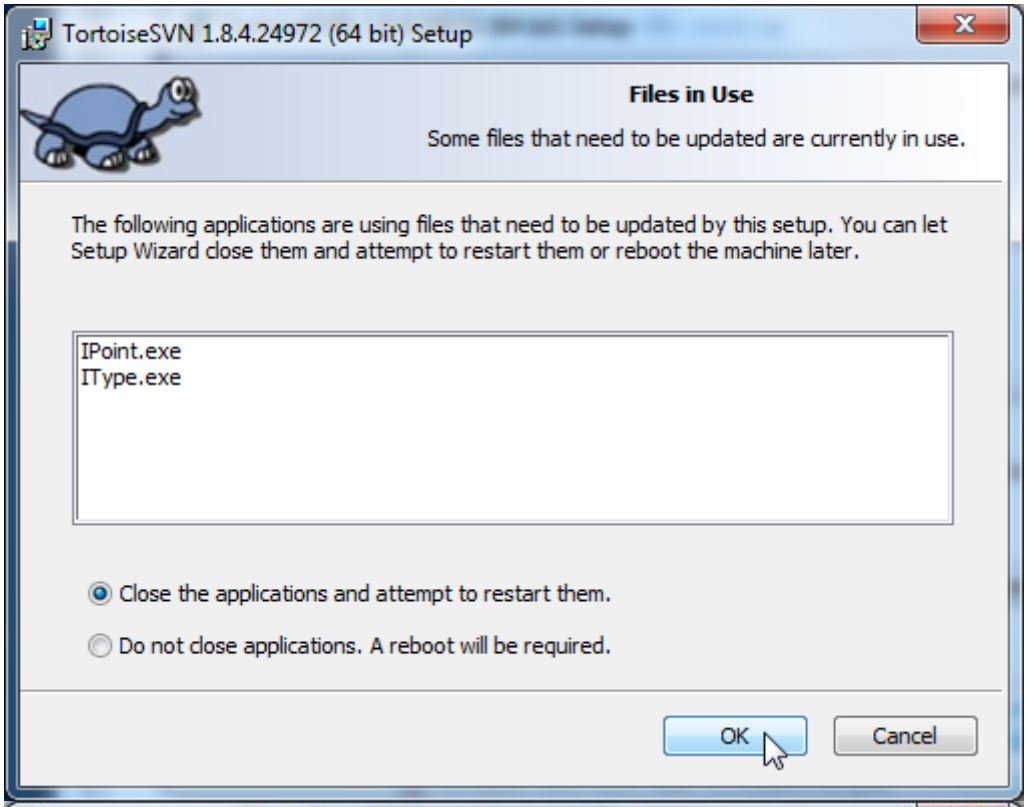

TortoiseSVN 1.8.4.24972 (64 bit) License Agreement
End-User License Agreement
Please read the following license agreement carefully
Copyright (C) 2003-2013 - Stefan Küng
TortoiseSVN is free. You don't have to pay for it, and you can use it any way you want. It is developed as an Open Source project under the GNU General Public License (GPL). That means you have full access to the source code of this program. You can find it on our website at http://tortoisesvn.net
Should you wish to modify or redistribute this program, or any part of it, you should read the full terms and conditions set out in the license agreement before doing so. A copy of the license is available on our website.
If you simply wish to install and use this software, you need only be aware of the
I accept the terms in the License Agreement
I do not accept the terms in the License Agreement
< Back
Next >
Cancel

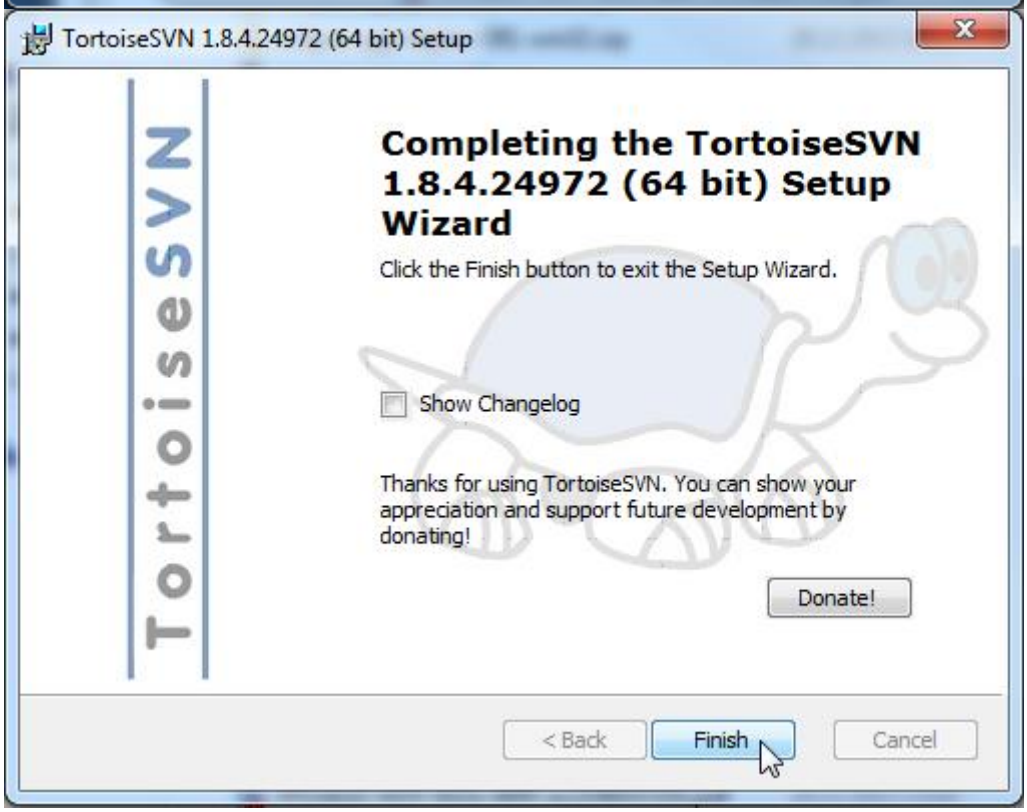

TortoiseSVN 1.8.4.24972 (64 bit) Setup
Custom Setup
Pick an install location and which features you want.
Click on the icons in the tree below to change the way features will be installed.
TortoiseSVN
Additional icon sets
command line client tools
Crash Reporter
Register diff/patch files
English (GB) dictionary
English (US) dictionary
Feature Description
The TortoiseSVN package and dependencies.
Feature Size
This feature requires 27MB on your hard drive. It has 6 of 6 subfeatures selected. The subfeatures require 19MB.
Location:
C:\Program Files\TortoiseSVN\
Browse
Reset
Disk Usage
< Back
Next >
Cancel


TortoiseSVN 1.8.4.24972 (64 bit) Setup
Ready to Install
The Setup Wizard is ready to begin the Custom installation
Click Install to begin the installation. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.
< Back
Install
Cancel

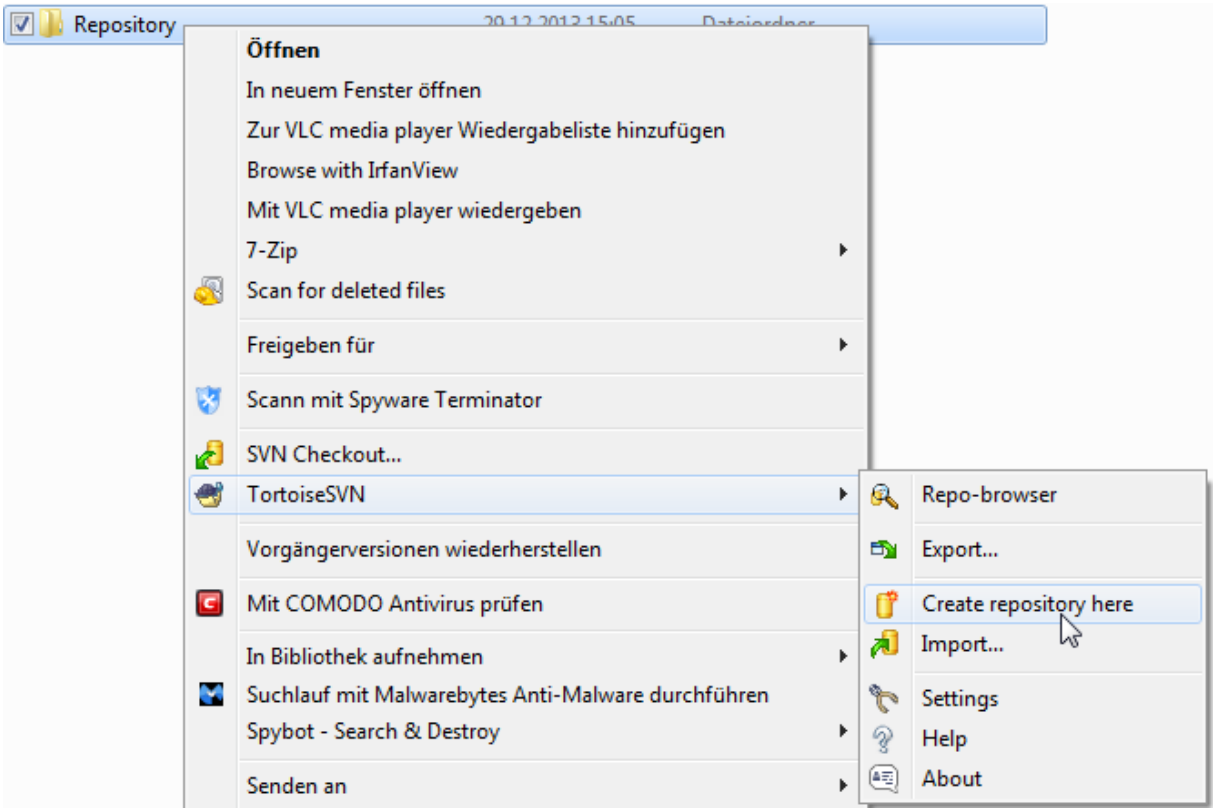

TortoiseSVN 1.8.4.24972 (64 bit) Setup
Installing TortoiseSVN 1.8.4.24972 (64 bit)
Please wait while the Setup Wizard installs TortoiseSVN 1.8.4.24972 (64 bit). This may take several minutes.
Status:
Thanks for using TortoiseSVN. You can show your appreciation and support future development by donating!
Donate!
< Back
Next >
Cancel


Benutzerkontensteuerung
Möchten Sie zulassen, dass Software auf diesem Computer durch das folgende Programm installiert wird?
Programmname: TortoiseSVN
Verifizierter Herausgeber: Open Source Developer, Stefan Kueng
Dateiursprung: Aus dem Internet heruntergeladen
Details anzeigen
Ja
Nein
Anzeigezeitpunkt für die Benachrichtigungen ändern

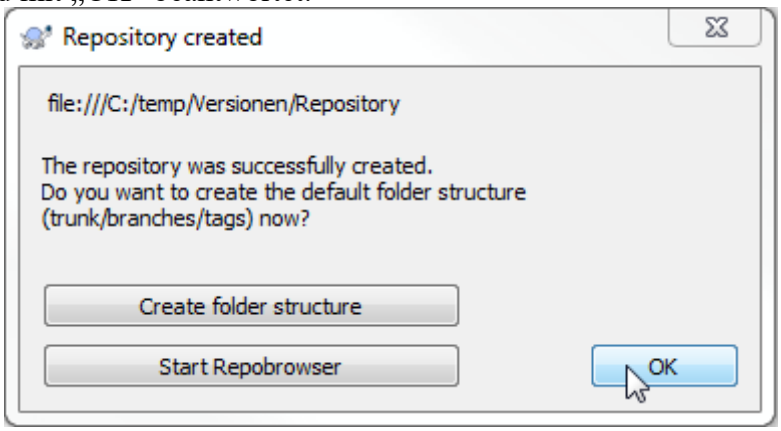




Unter anderem kann man mit Tortoise auch Repositories anlegen. Dazu wird ein leeres Verzeichnis erstellt und dann ein Rechtklick auf diesem Verzeichnis durchgeführt. Unter „TortoiseSVN > Create Repository here...“

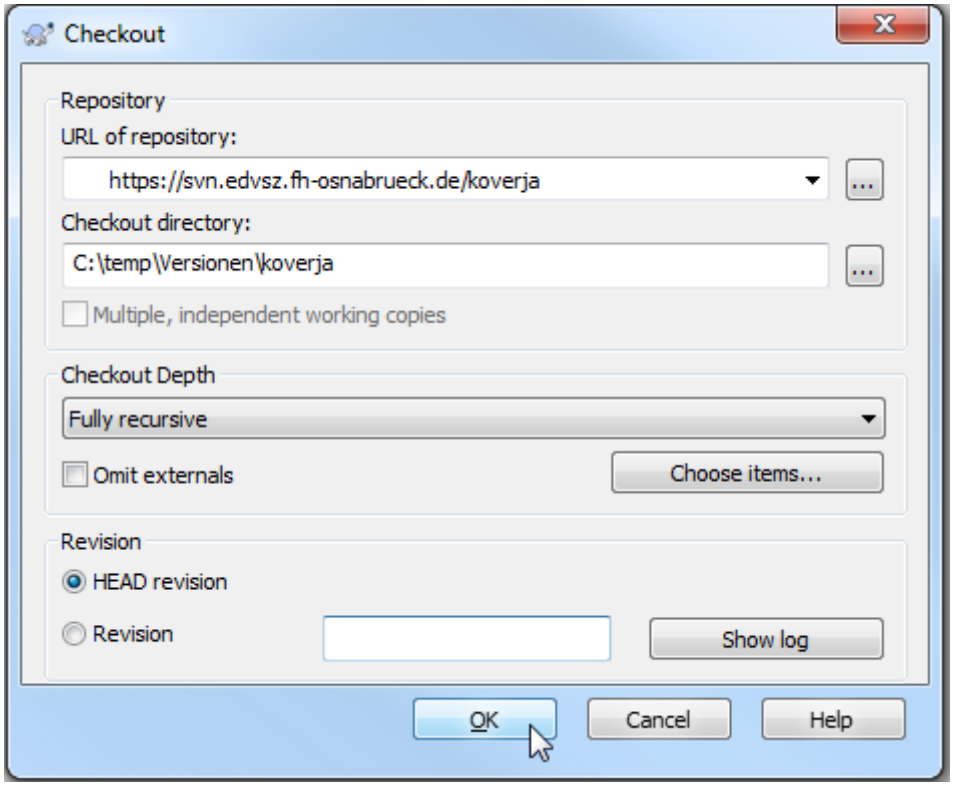


Die Abfrage wird mit „OK“ beantwortet.

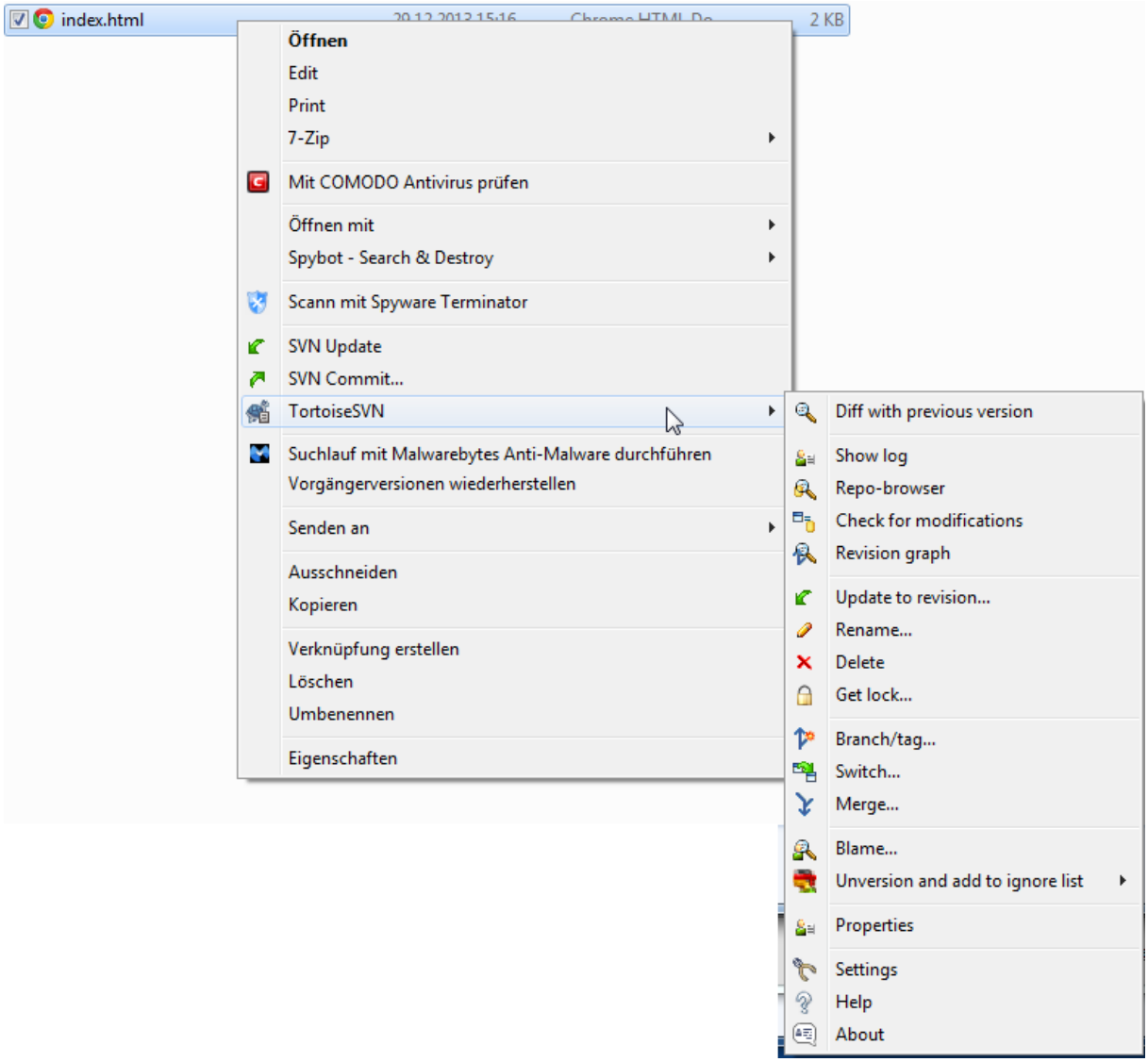


Die besondere Aufgabe des Ordners wird auch durch das neue Icon ersichtlich.

Weiterhin ist interessant, dass man wie in einem vorherigen Bild gezeigt auch Dateien auschecken kann. Bei einem Rechtsklick in einem Ordner kann der Punkt „SVN Checkout ...“ gewählt werden. Man muss dabei oben das Repository auswählen.

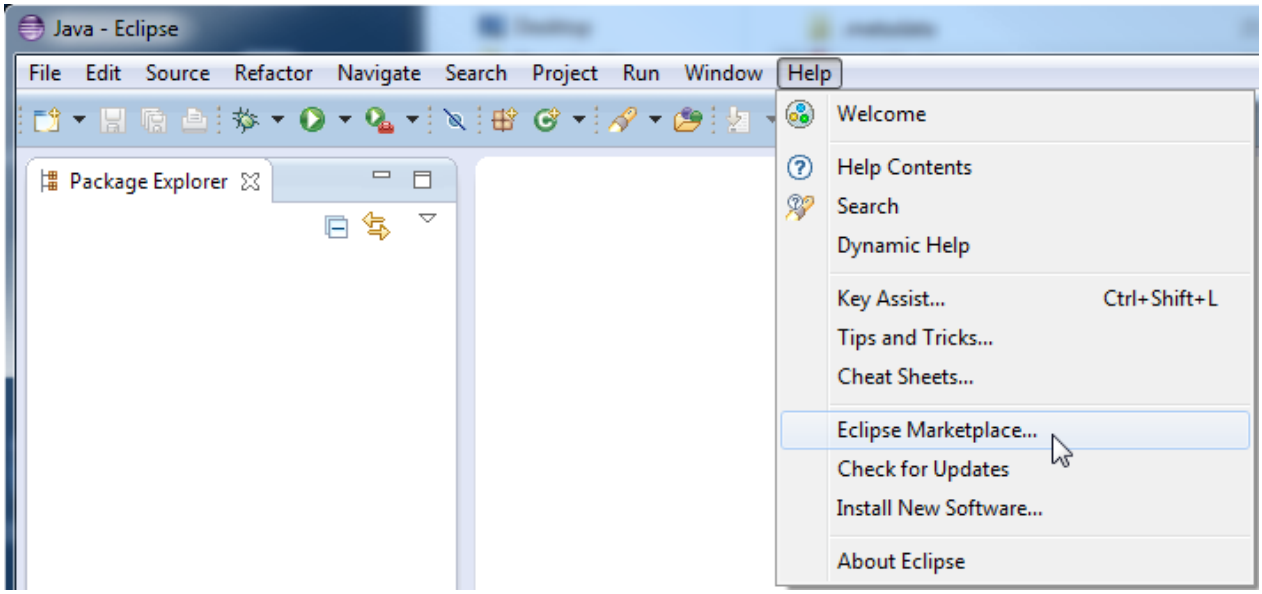


Bei Dateien, die unter Versionsverwaltung stehen, wird Commit und Update über einen Rechtsklick angeboten. Weitere Funktionalität befindet sich unter „TortoiseSVN“.

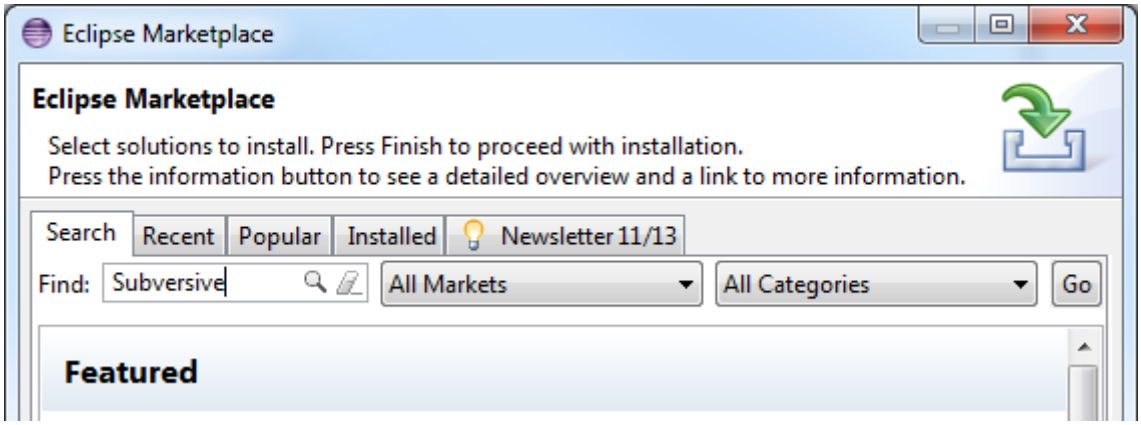


D.3 Subversive-Installation

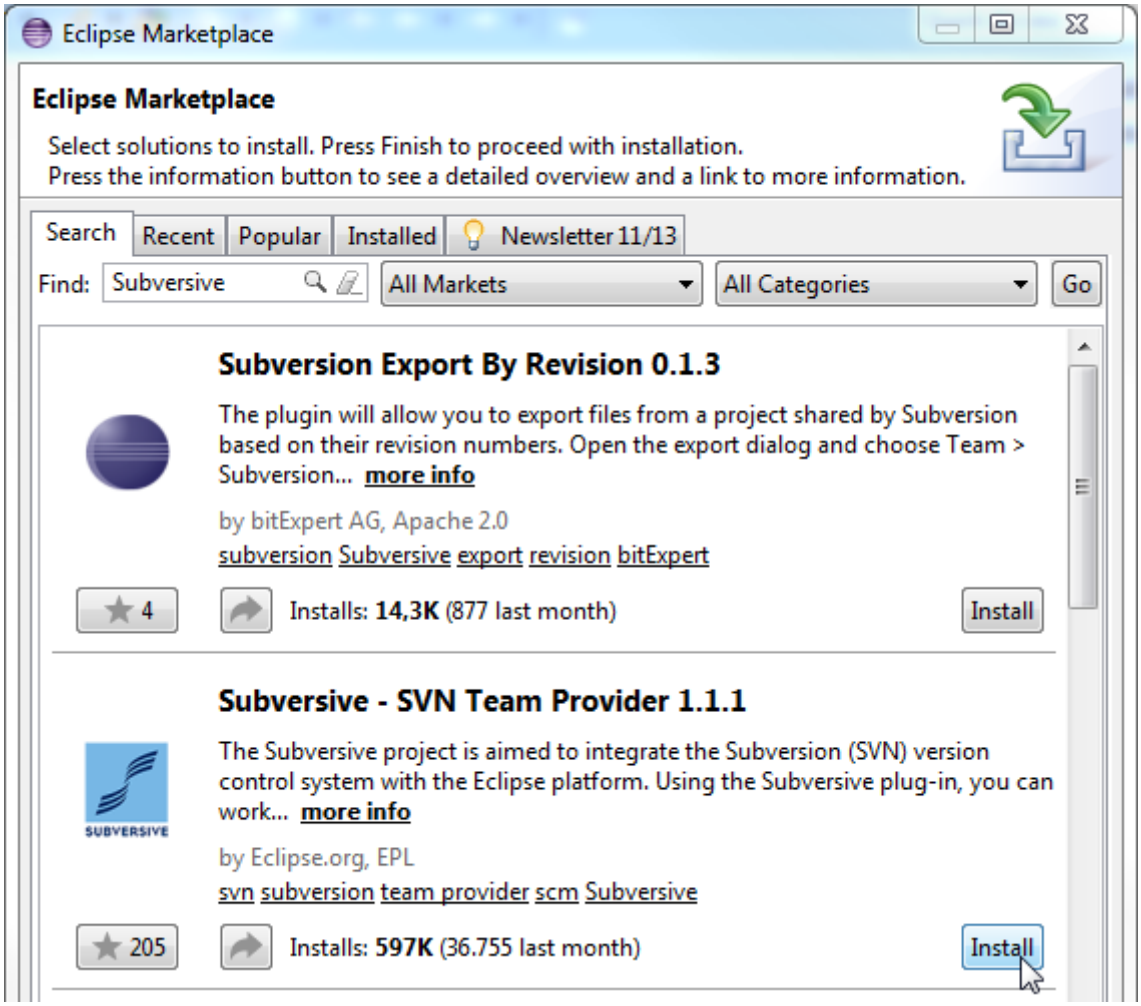
Subversive erlaubt die Nutzung von Subversion von Eclipse aus und bietet alle zentralen Funktionalitäten zur Arbeit mit SVN-Repositories. Die Installation erfolgt am einfachsten über den Eclipse Marketplace. Der unter „Help“ zu erreichen ist.



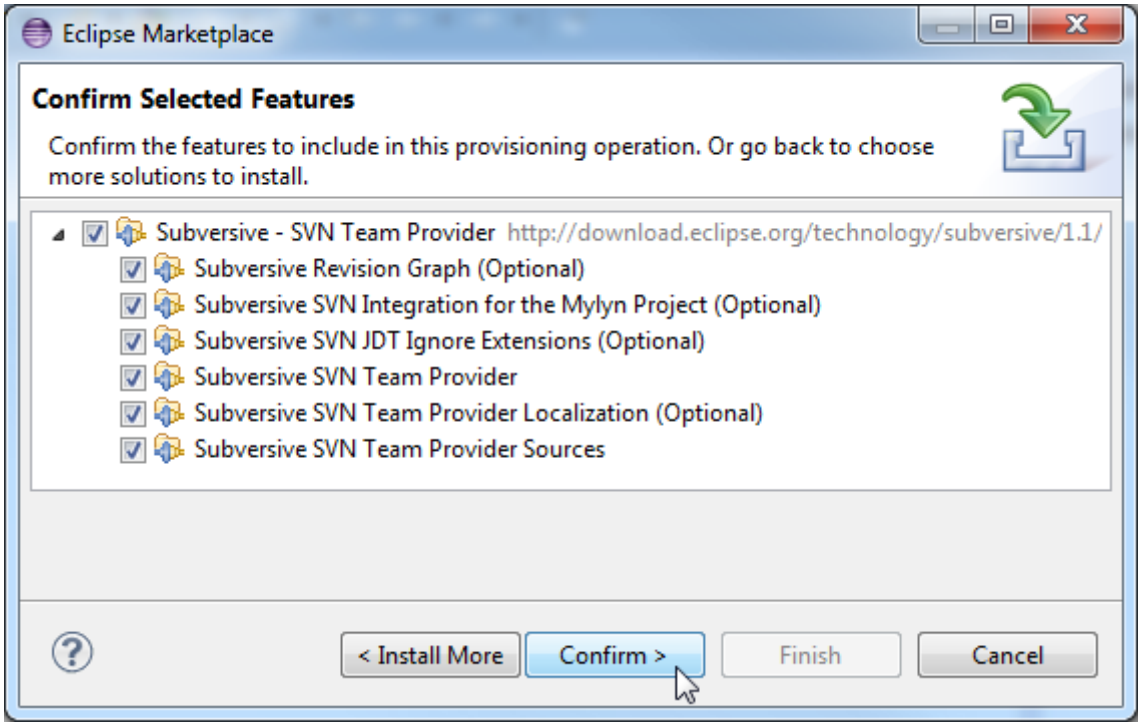
Im Suchfenster wird Subversive eingegeben.



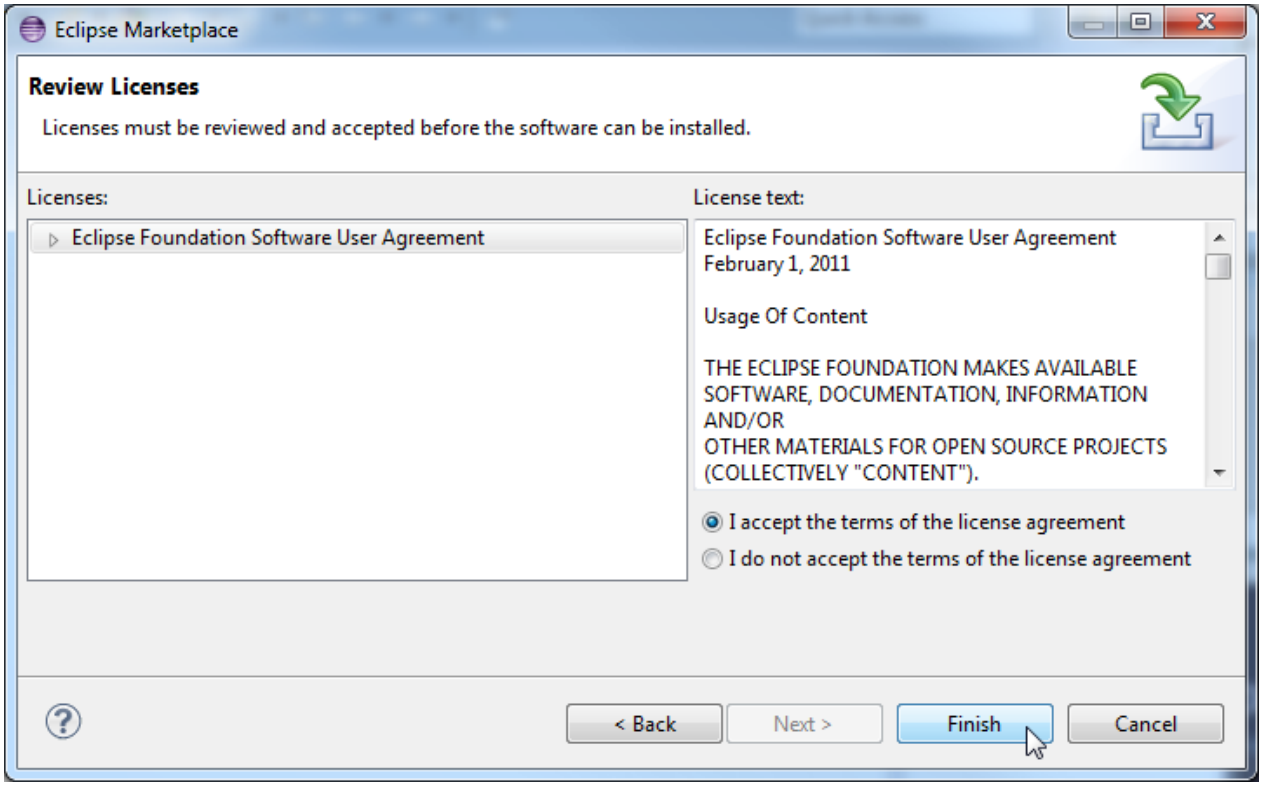
Man kann direkt auf „Install“ klicken.



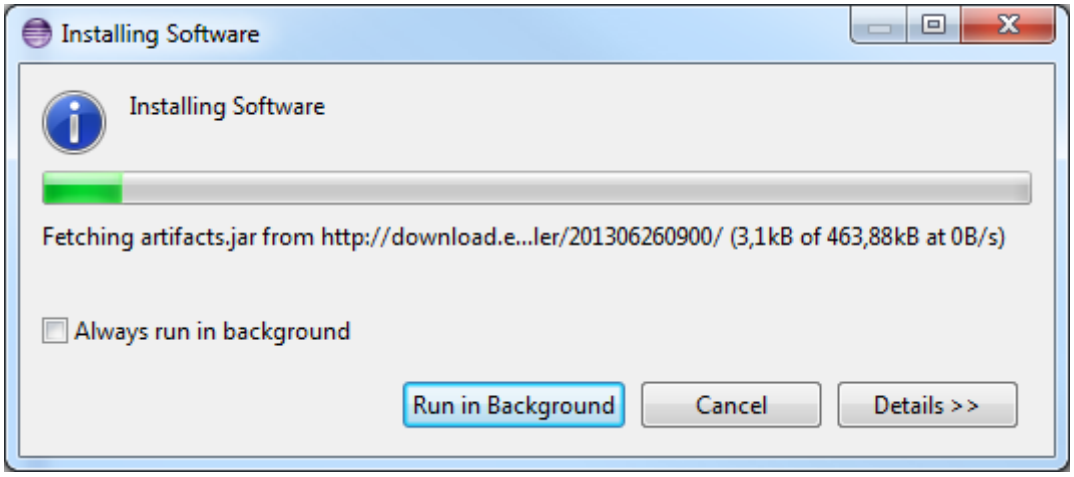
Man klickt direkt auf „Confirm“.



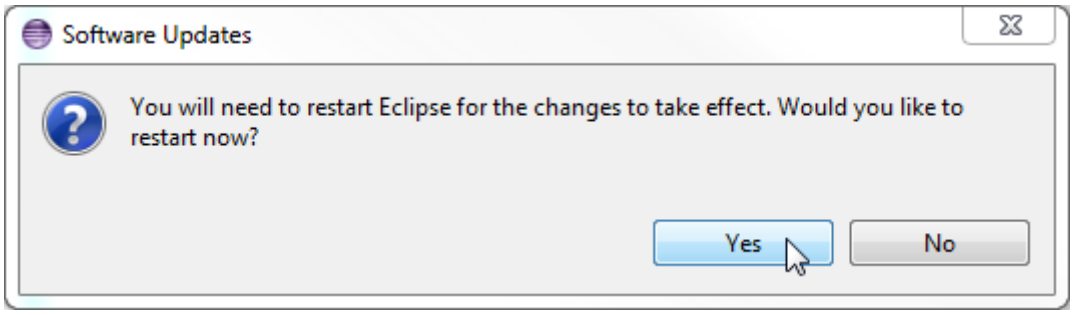
Die Lizenzbedingungen müssen akzeptiert und dann „Finish“ geklickt werden.



Die Installation dauert etwas.



Dann muss ein Neustart erfolgen.



alt: Direktinstallation

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.

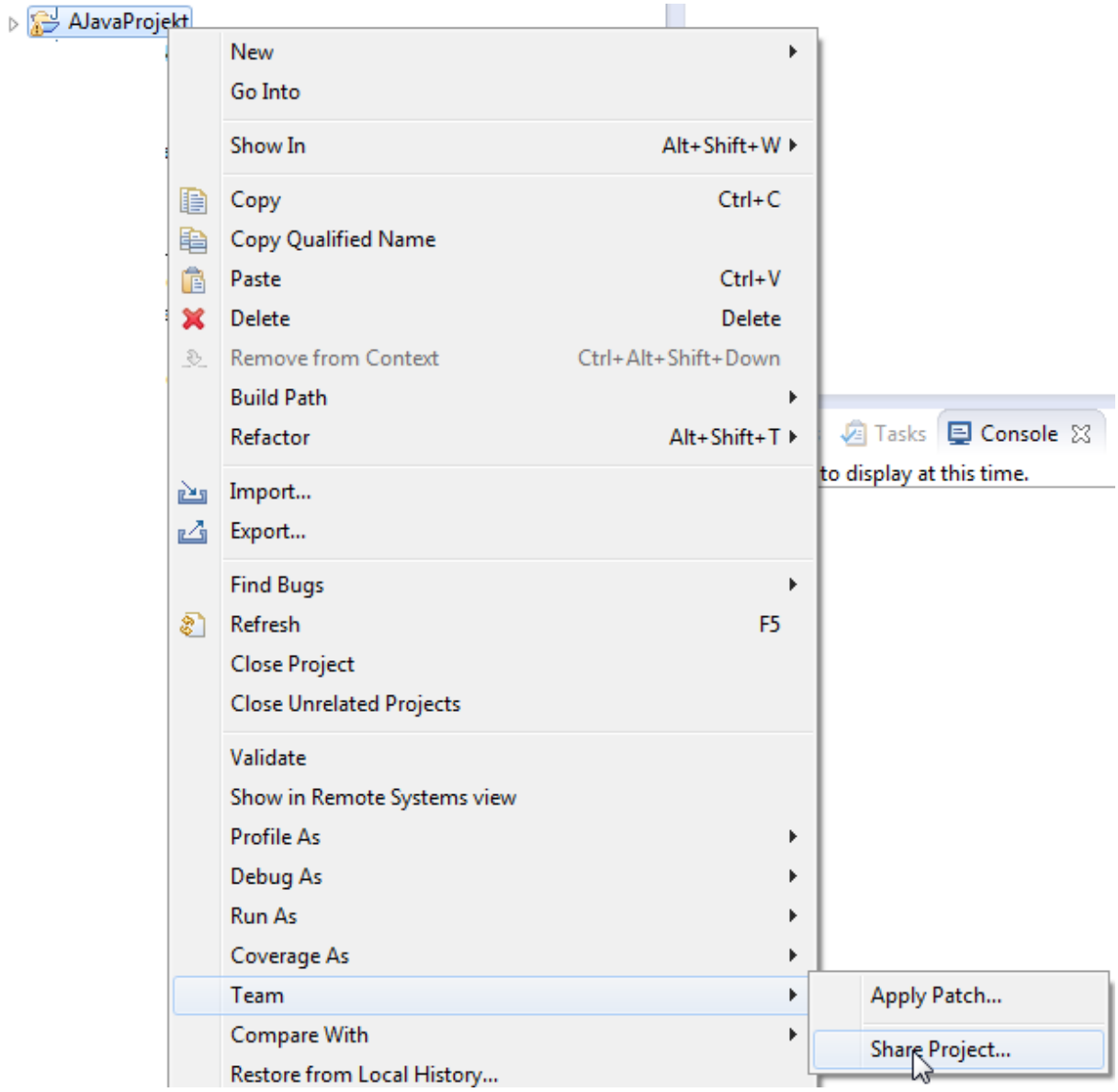
```
Subversive-incubation-0.7.0.v20080218.zip
```

```
Subversive-connectors-2.0.0.v20080214.zip
```

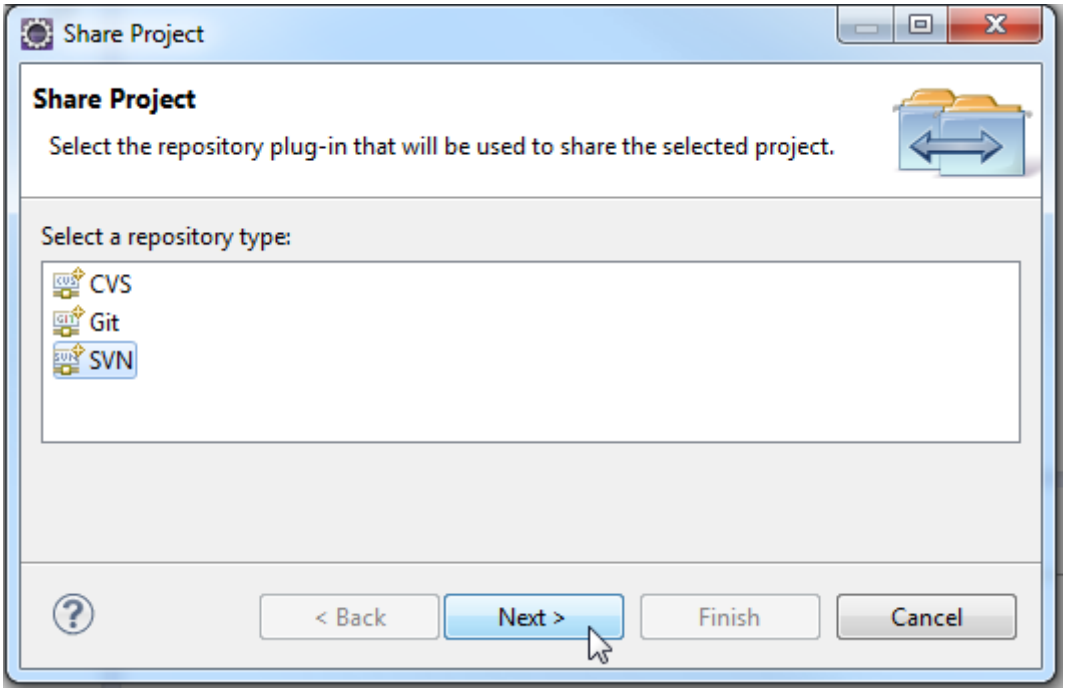
heißen können. Subversive ist im zur Verfügung gestellten Eclipse-Paket integriert.

D.4 Einrichtung des Projekts

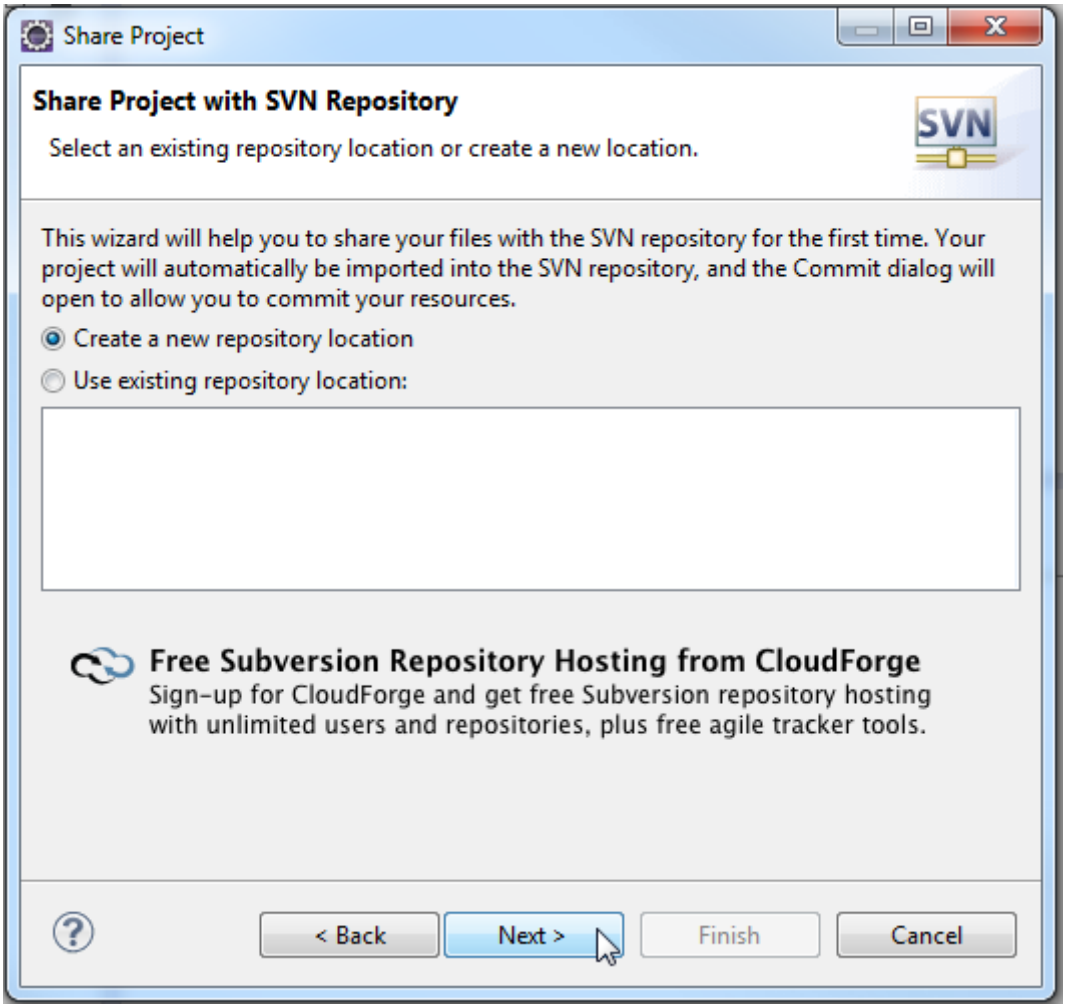
Möchte man ein neues Projekt zum ersten Mal in ein Repository einspielen, wird ein Rechtsklick auf dem Projekt gemacht und „Team > Share Project…“ ausgewählt.



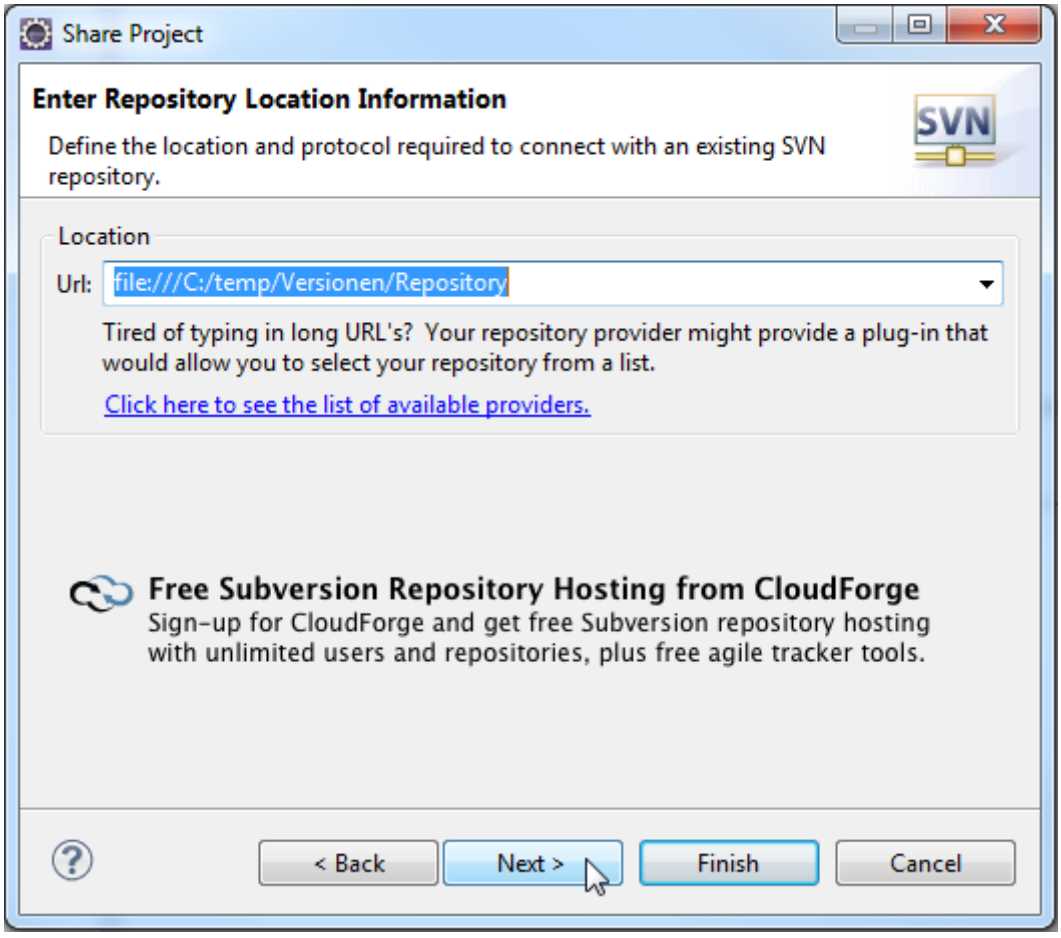
Es wird „SVN2 gewählt und „Next>“ geklickt.



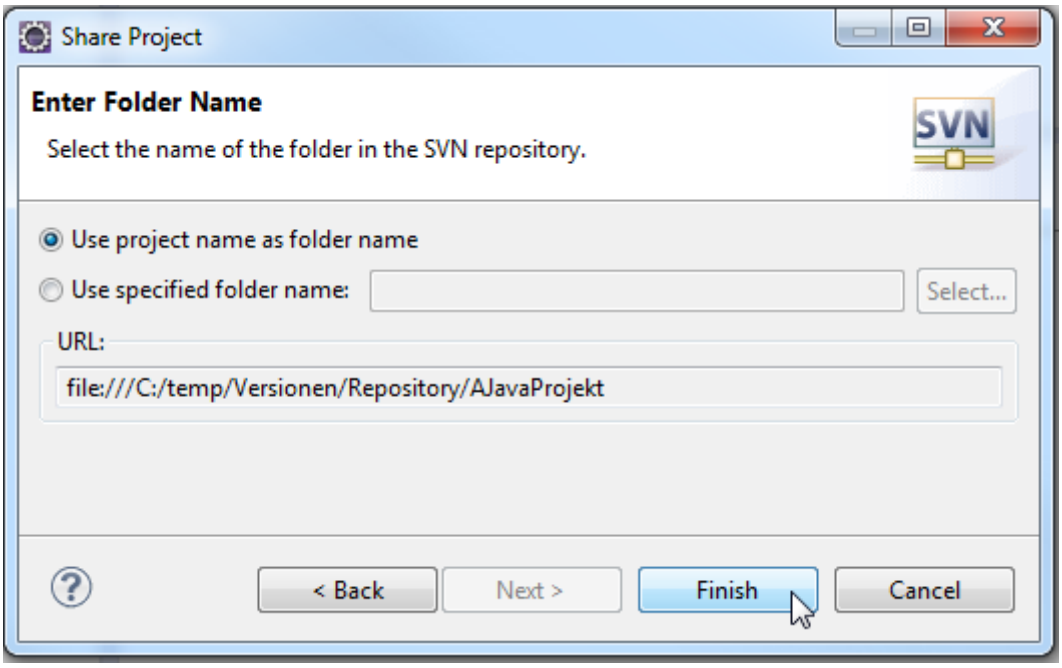
Im Beispiel soll ein neues Repository gewählt werden, genauer jenes, das mit TortoiseSVN eingerichtet wurde.



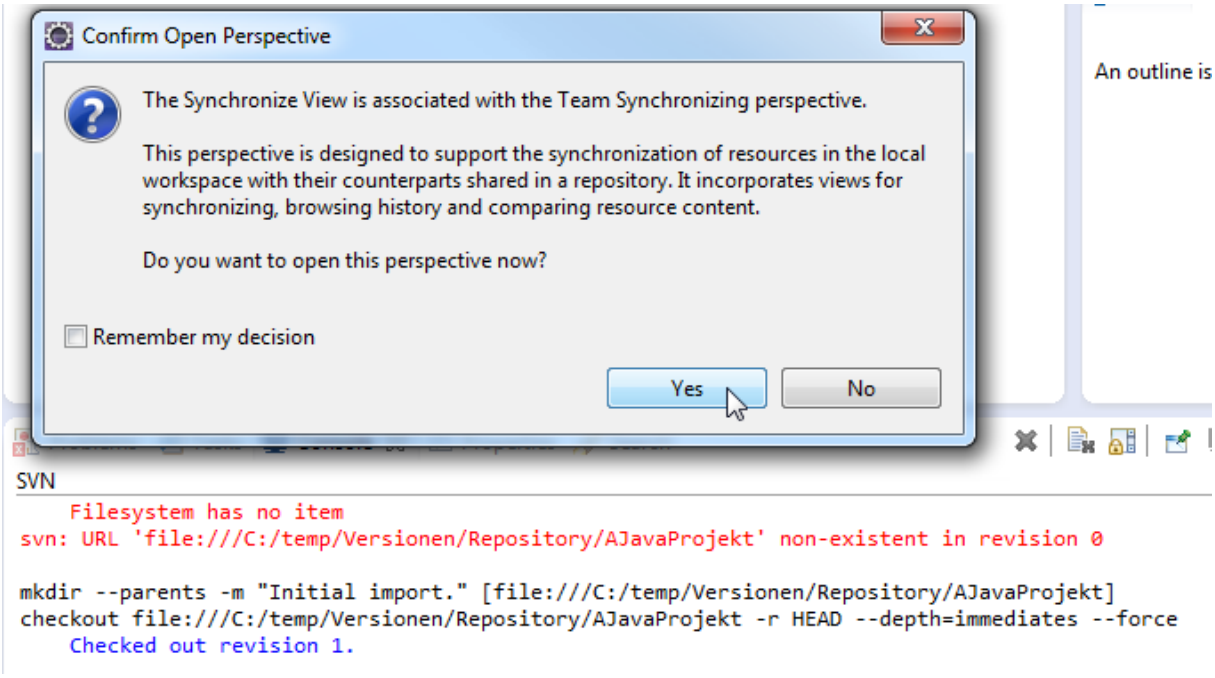
Die URL des Repositorys wird angegeben, im Spezialfall beginnend mit file:///.



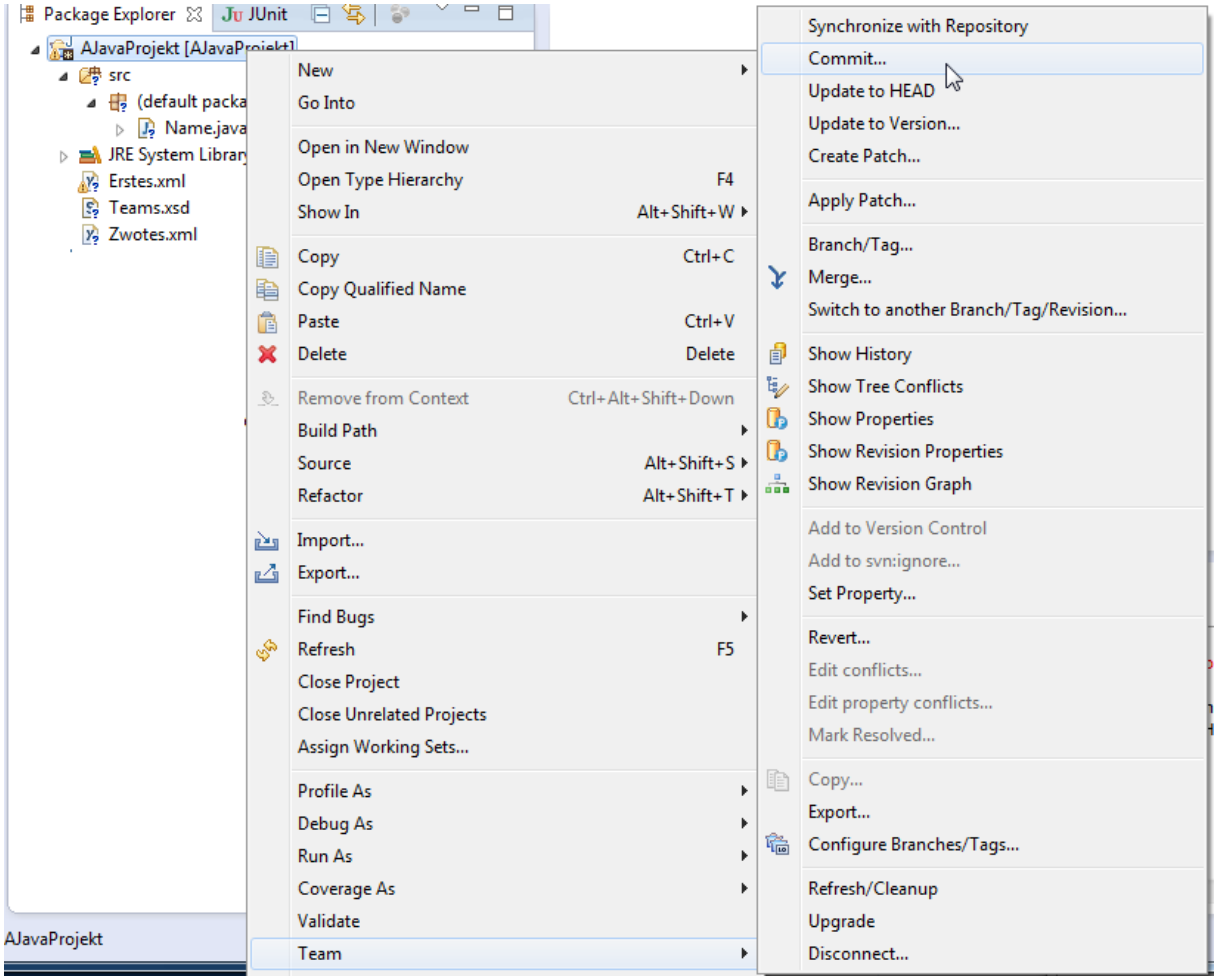
Der Projektname kann angepasst werden, wird es in diesem Fall aber nicht.



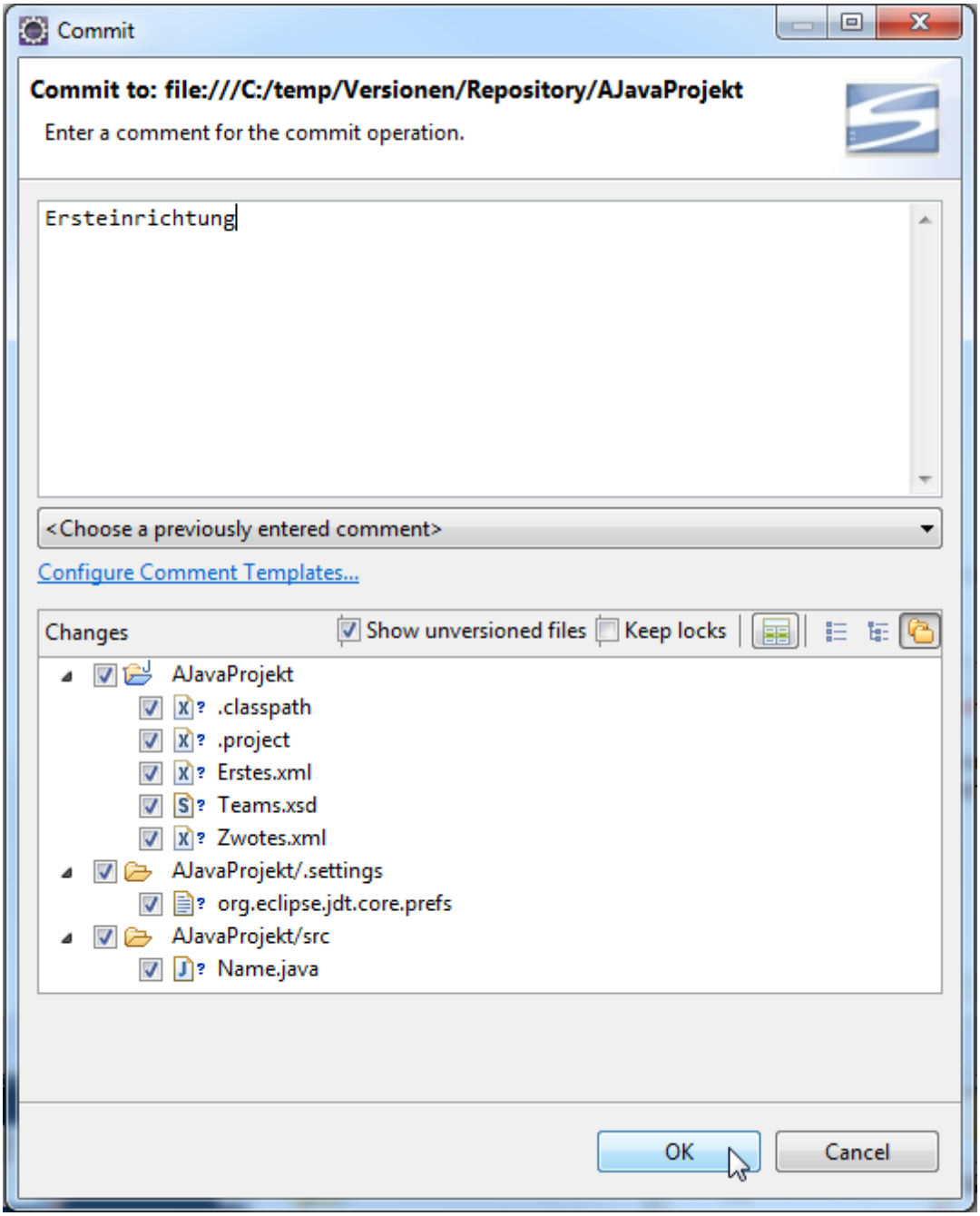
Man erkennt, dass das Projekt angelegt wird.



Im nächsten Schritt werden die Projektdaten in das SVN-Repository eingespielt. Dies passiert nicht im ersten Schritt, da man oft einzelne Dateien, wie Binärdateien, von der Versionierung ausnehmen, möchte. Mit einem Rechtsklick auf dem Projekt wird „Team > Commit…“ gewählt.



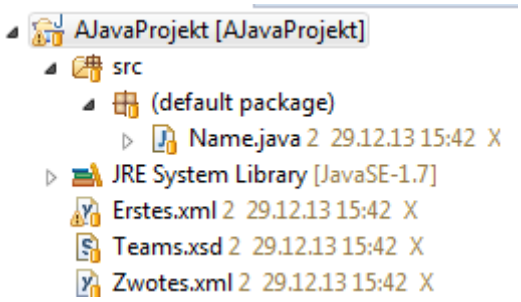
Nun kann man auswählen, welche Dateien unter Versionsverwaltung stehen sollen. Generell sollte beim Einchecken immer ein Kommentar angegeben werden.



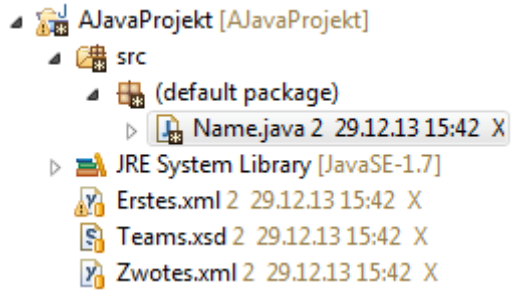
Man erkennt, dass die Dateien in das Repository kopiert werden.

```
SVN
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/.project
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/.settings
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/.settings/org.eclipse.jdt.core.prefs
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/Erstes.xml
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/Teams.xsd
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/Zwotes.xml
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/src
     Adding          F:/workspaces/workspace_SoSe14/AJavaProjekt/src/Name.java
     Transmitting file data ...
     Committed revision 2.
```

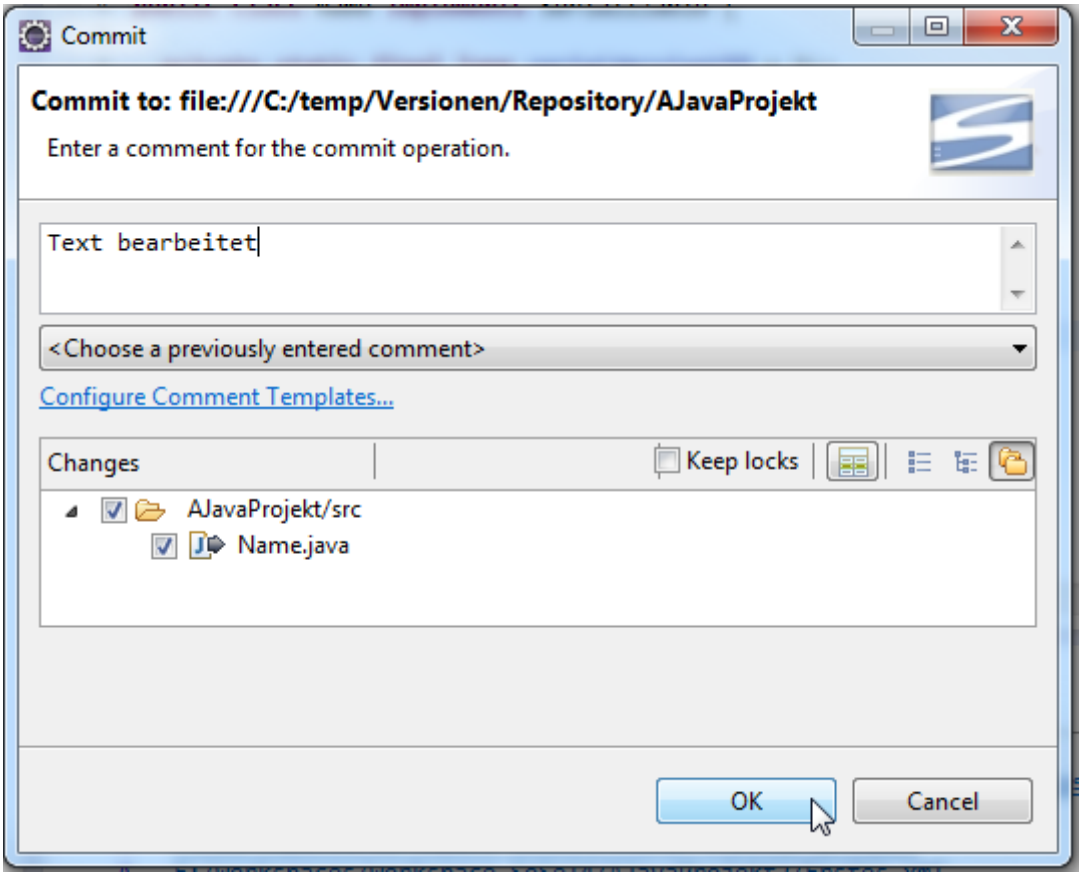
Bei aktuellen Daten werden die Dateien mit einer kleiner Tonne rechts unten im Icon markiert. Man bedenke, dass man vor dem Beginn der Arbeit immer ein „Update“ der Daten von SVN besorgen sollte/muss.



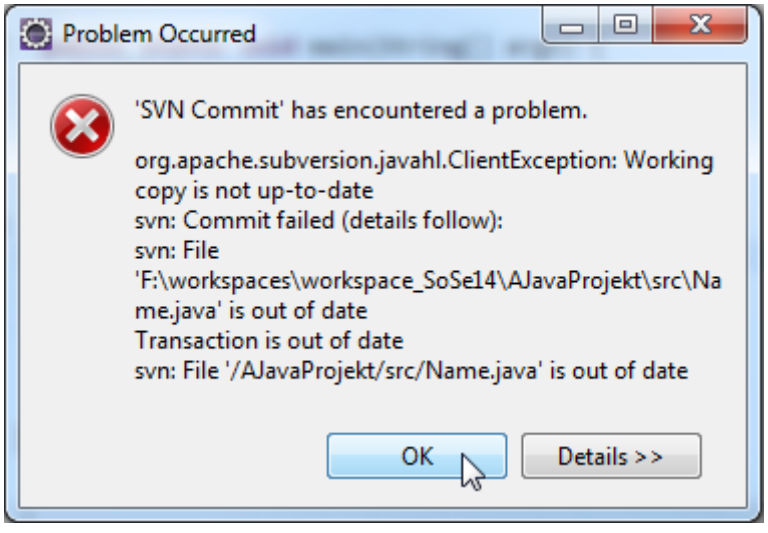
Falls Dateien bearbeitet wurden, wird die Datei und die Pfade zu dieser Datei mit einem Stern im schwarzen Kreis gekennzeichnet.



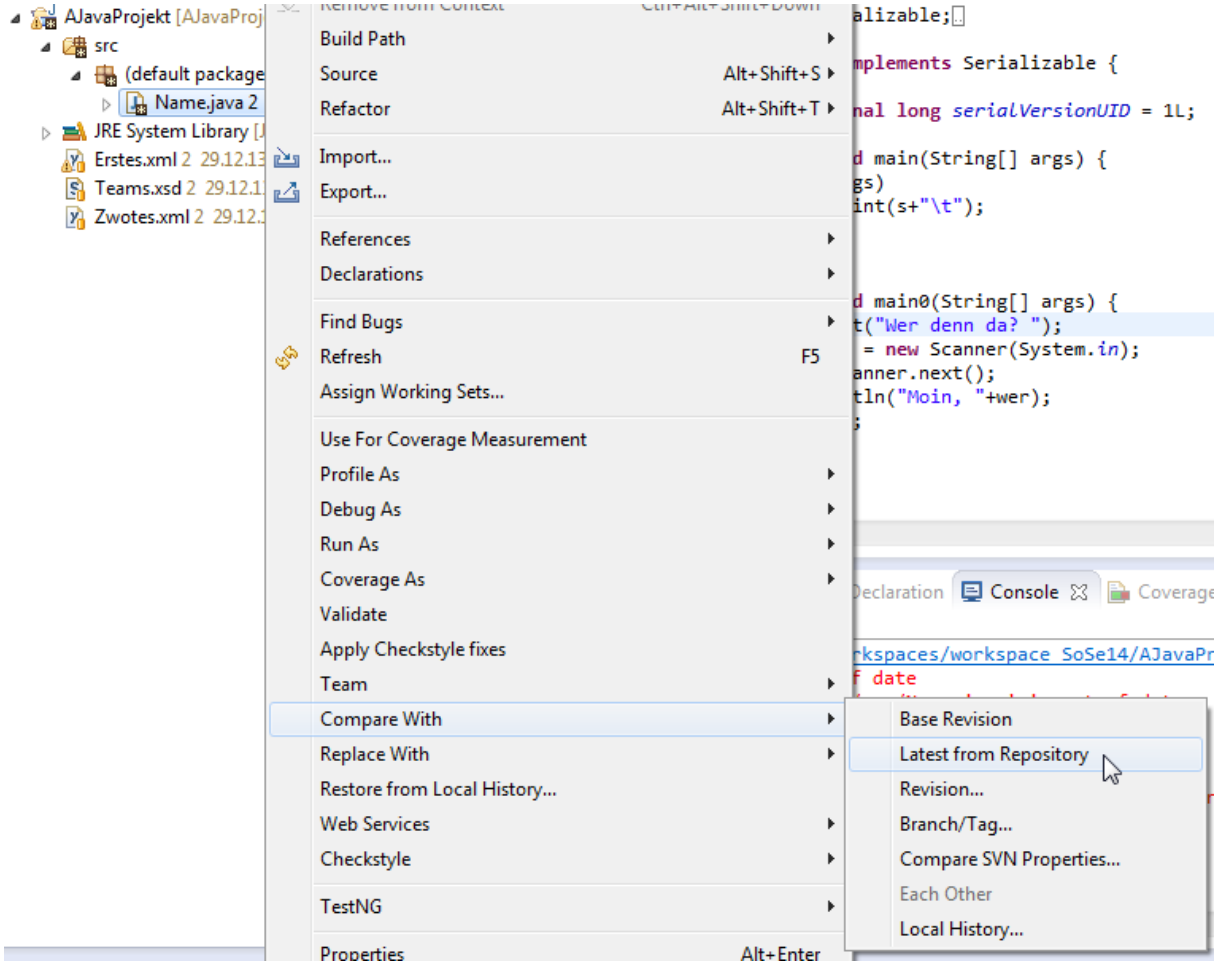
Nach den Änderungen wird wieder ein Commit mit einem Kommentar ausgeführt.



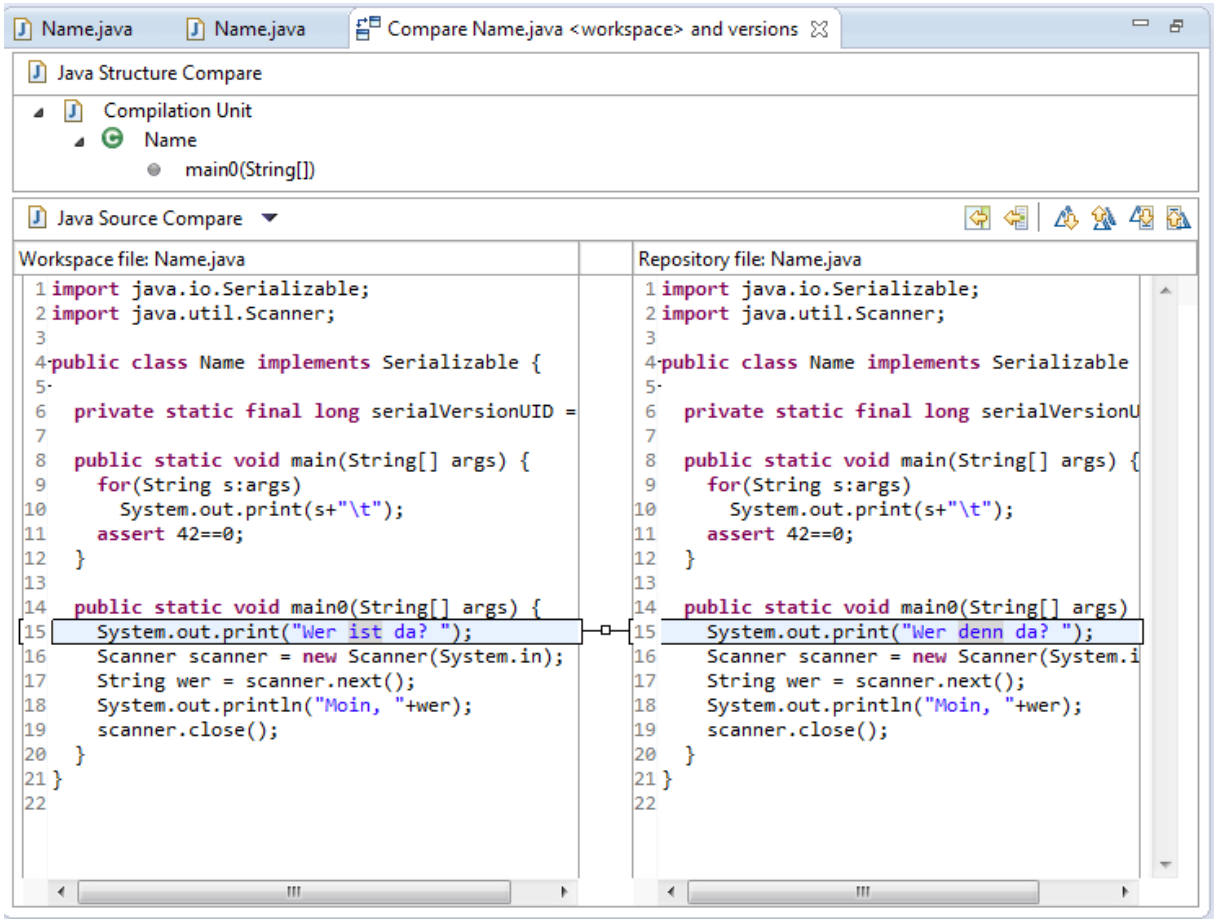
Sollte die Datei von jemand anderen zwischenzeitlich bearbeitet worden sein, erhält man eine Fehlermeldung.



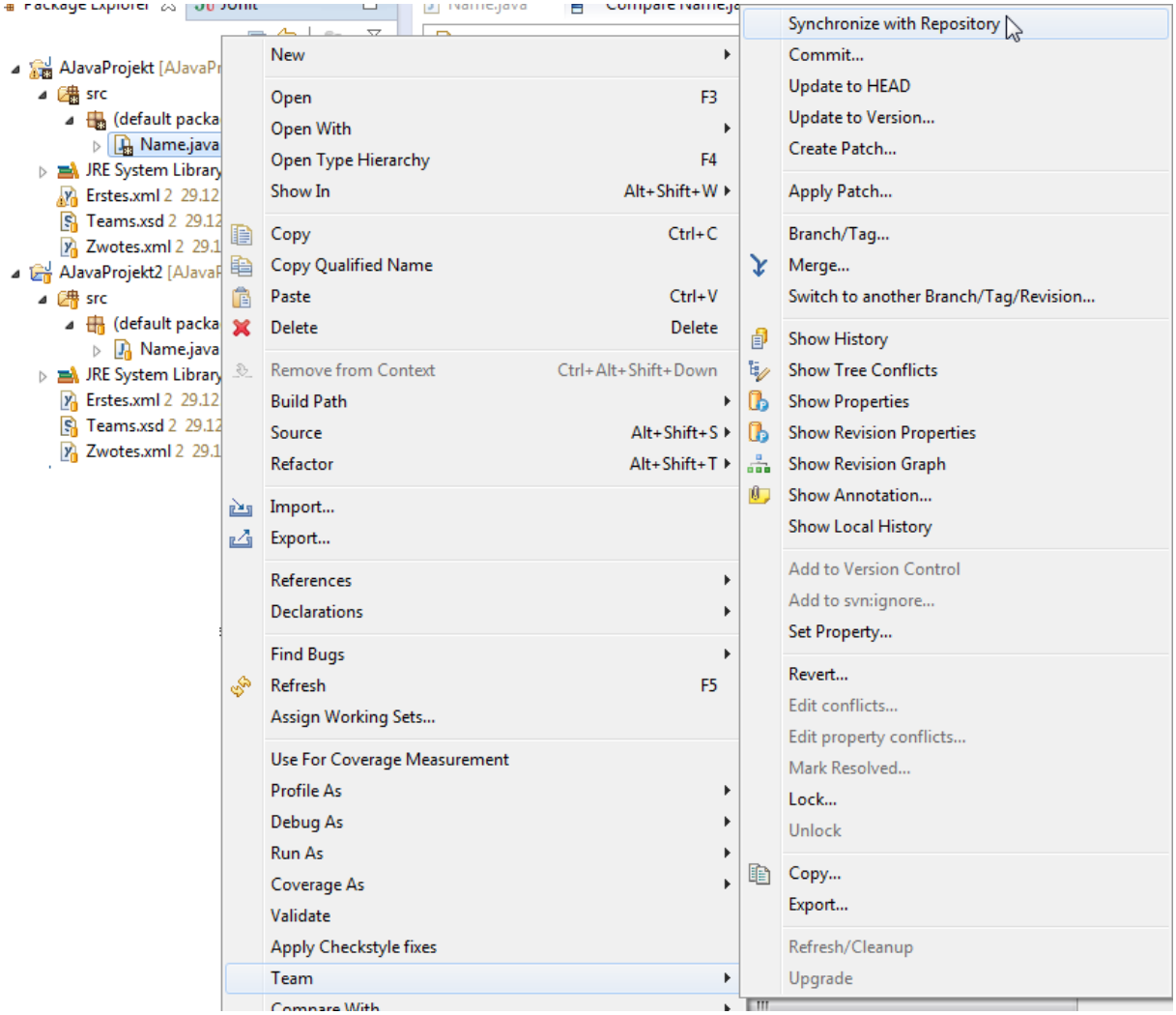
Der nächste Schritt ist meist ein Vergleich der eigenen Version mit der neuen aktuellen Version über einen Rechtsklick und der Auswahl „Compare With > Latest vom Repository“.



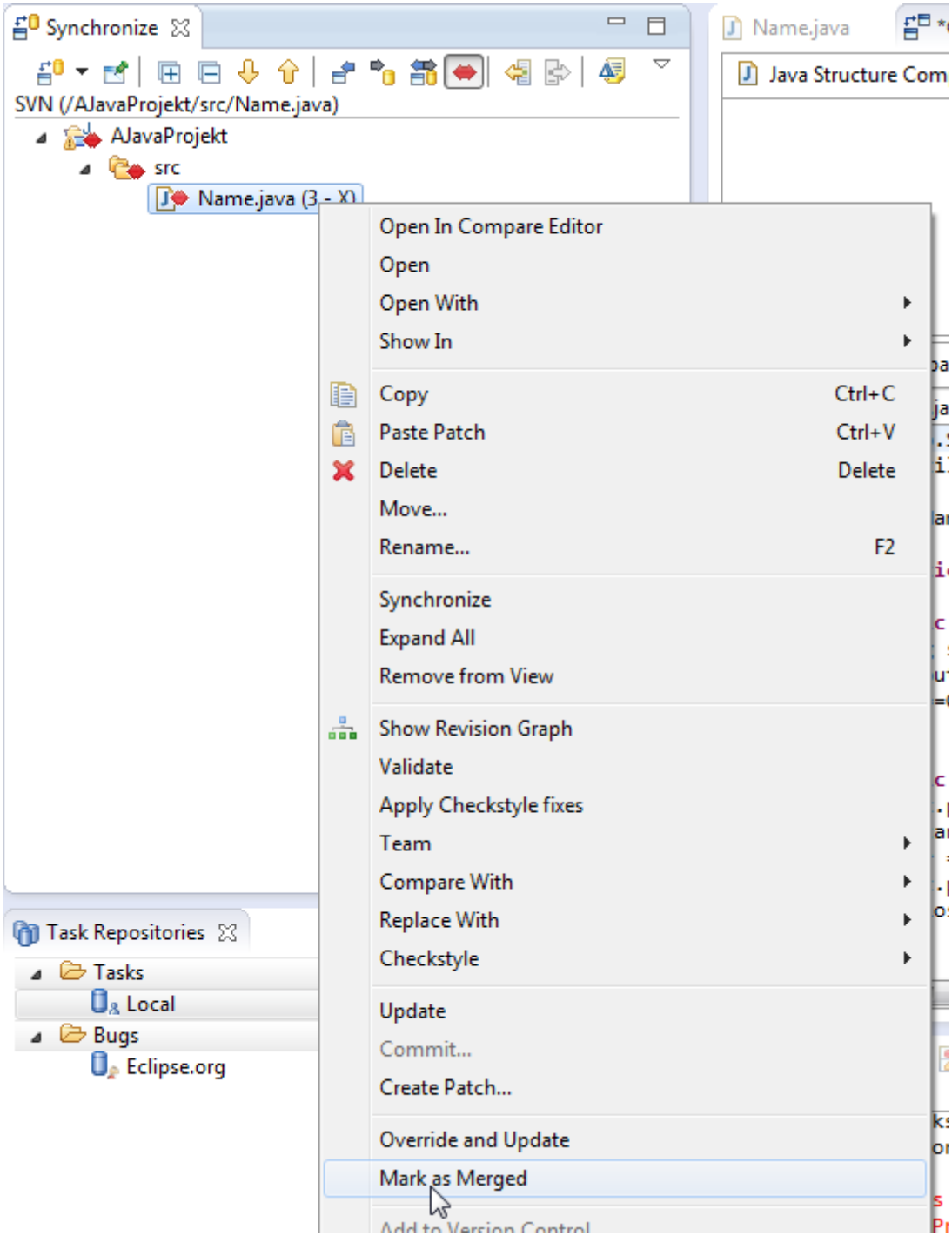
Unterschiede werden meist klar angezeigt.



Man kann dann z. B. Änderungen vornehmen oder einfach die eigene Lösung verwerfen und ein Update der neuen Version laden. Möchte man seine Änderungen übernehmen, macht man einen Rechtsklick auf der Datei und wählt „Team > Synchronize with Repository“ mit dem auch in den Repository-View gewechselt wird.



Hier kann u.a. mit einem Rechtsklick „Mark as Merged“ gewählt werden.



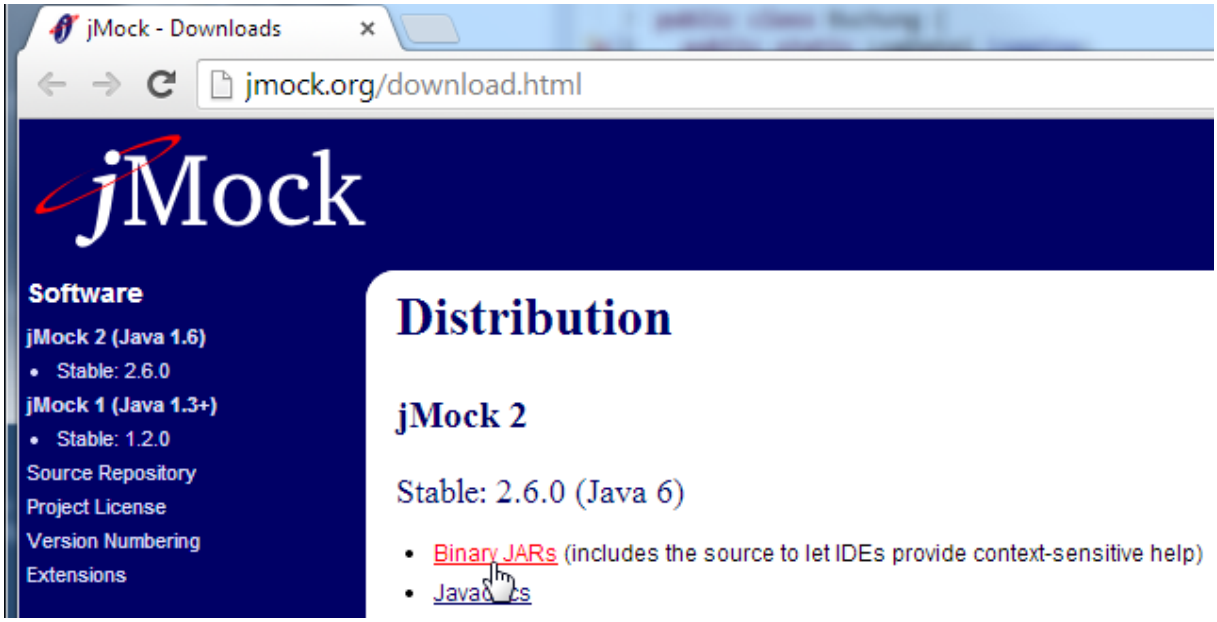
Danach kann z. B. in der Java-Perspektive ein Commit durchgeführt werden.

E JMock (2.6.0)

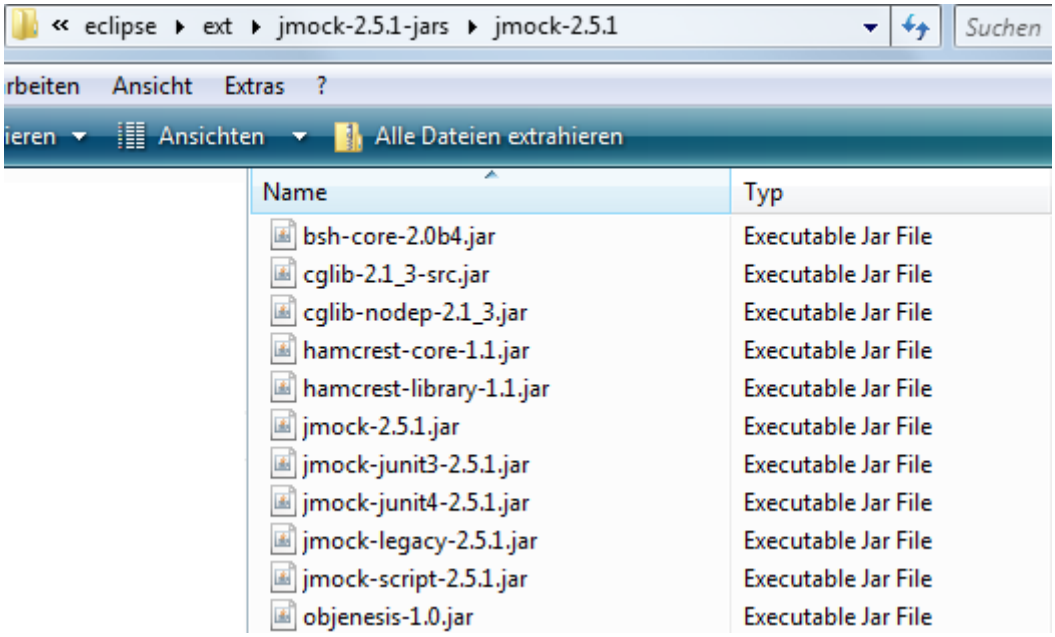
Möchte man seine eigene Software testen, ist diese häufig von Klassen abhängig, die von anderen Entwicklern stammen, die ggfls. die Programmierung noch nicht abgeschlossen haben. In diesem Fall muss man für die eigenen Tests sogenannte Mock-Klassen erstellen, also minimale Implementierungen der fehlenden 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");
		}
	}
}
```

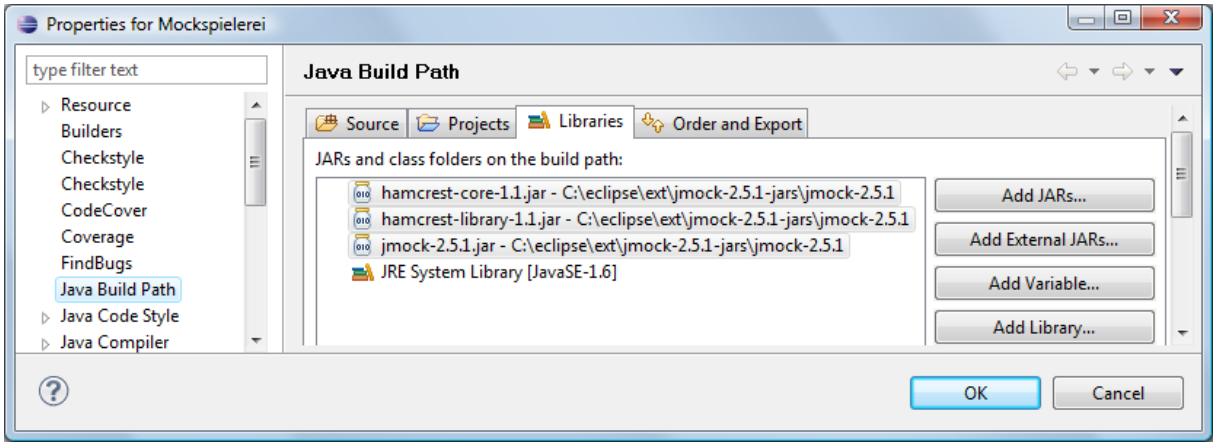
Statt die Mock-Klassen von Hand zu implementieren, was auch eine sinnvolle Variante ist, kann man hier Hilfswerkzeuge nutzen, die es Einem erlauben, die Mock-Erzeugung dynamisch innerhalb von Testfällen durchzuführen. Ein solches Werkzeug ist JMock, das unter http://www.jmock.org/download.html erhältlich ist.



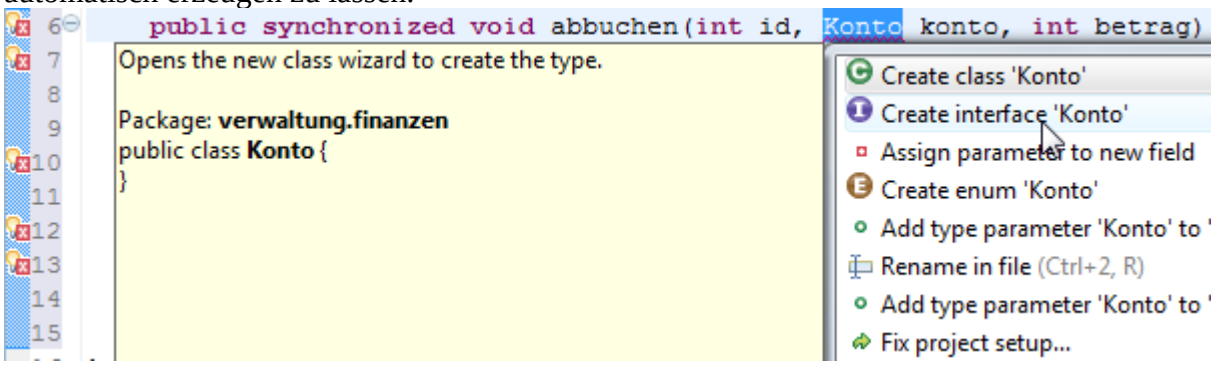
Man beachte, dass sich einige Jar-Dateien nach dem Auspacken im Verzeichnis jmock-2.5.1 (Nummer abhängig von der benutzten Version) befinden, die ggfls. ins Java-Projekt mit eingebunden werden müssen.



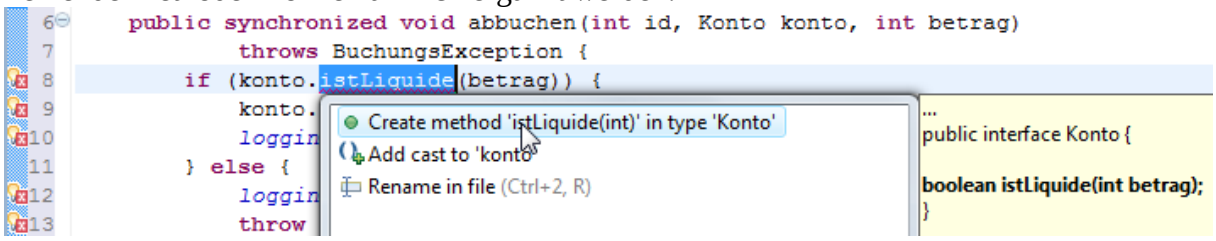
Die Möglichkeiten Jar-Dateien einzubinden wurden bereits im Abschnitt 7.10 beschrieben, das folgende Bild zeigt die minimal für das Beispiel benötigten Bibliotheken.



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.

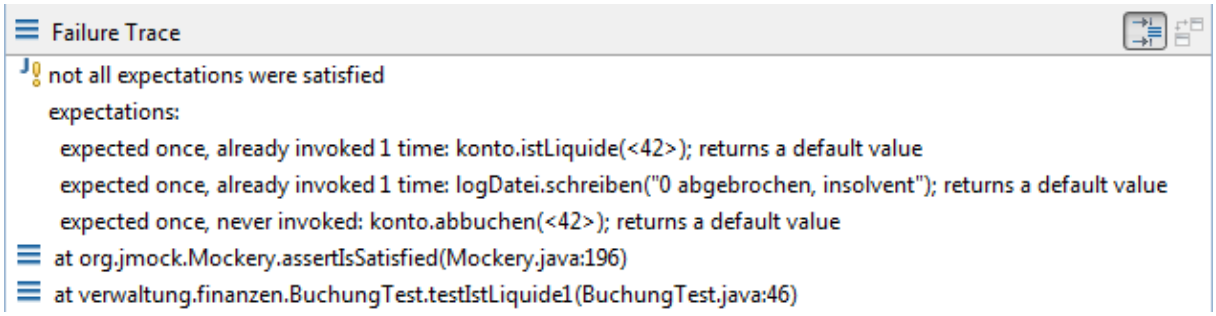


Das folgende Beispiel zeigt eine sehr kurze Einführung in die JMock-Nutzung, die detaillierter im Cookbook von der JMock-Seite nachgelesen werden kann.

```
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();
    }
  }
}
```

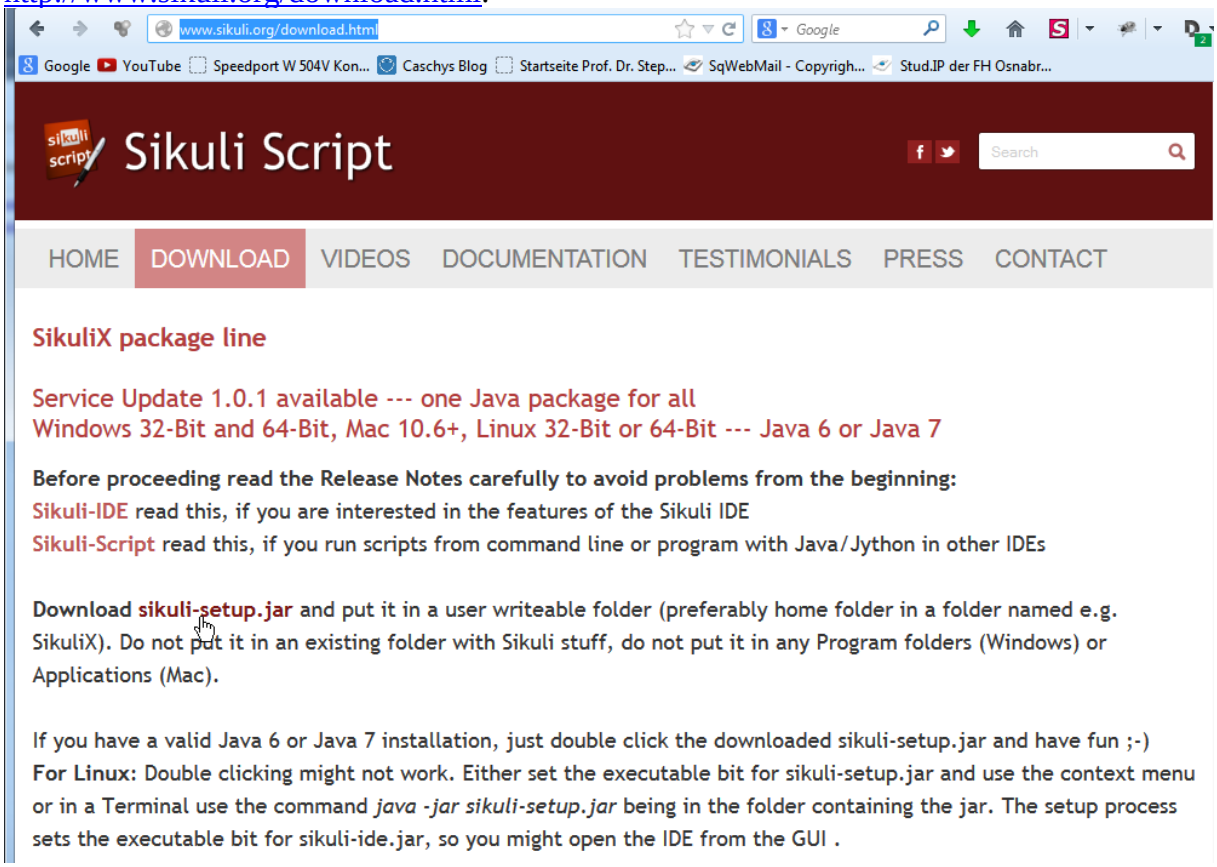
Man erkennt das zentrale Mockery-Objekt zur Steuerung der Mock-Nutzung. Weiterhin kann mit der Methode mock(Class-Object) ein beliebiges Mock-Objekt einer Klasse, genauer eines Interfaces, angelegt werden. Erwartete Methodenaufrufe von Mock-Objekten sind vor ihrem Auftreten als Expectations-Objekt anzugeben. 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-Methode 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.


Failure Trace
not all expectations were satisfied
expectations:
expected once, already invoked 1 time: konto.istLiquide(<42>); returns a default value
expected once, already invoked 1 time: logDatei.schreiben("0 abgebrochen, insolvent"); returns a default value
expected once, never invoked: konto.abbuchen(<42>); returns a default value
at org.jmock.Mockery.assertIsSatisfied(Mockery.java:196)
at verwaltung.finanzen.BuchungTest.testIstLiquide1(BuchungTest.java:46)

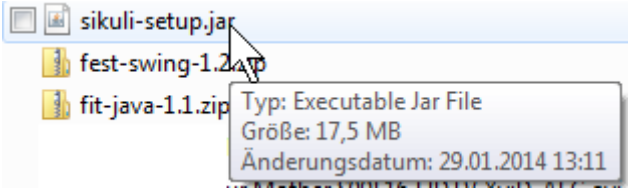
F Sikuli (1.0.1)

Mit Sikuli sind mit der Maus gesteuerte Abläufe automatisierbar. Dabei werden dem Werkzeug Bilder übergeben, die auf dem aktuellen Arbeitsplatz gesucht und dann von Sikuli angeklickt werden können. Dieses Werkzeug eignet sich sehr gut zum Testen von Oberflächen, da die Testausführung in Java steuerbar ist.
Die Installation ist leider etwas kompliziert, da sie auf verschiedenen Rechnern zu Problemen führen kann. Lösungsansätze werden auf der zugehörigen Webseite http://www.sikuli.org/ vorgestellt.

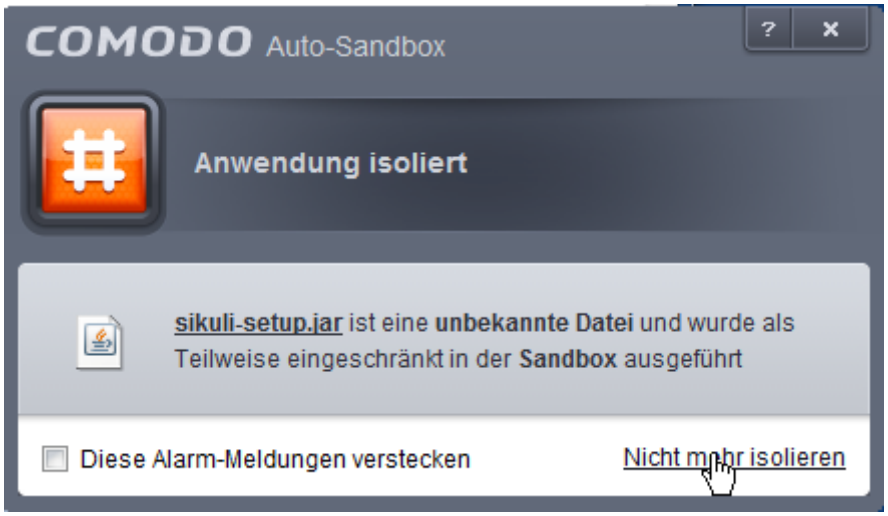
Der Download beginnt mit einem Klick auf sikuli-setup.jar auf der Webseite http://www.sikuli.org/download.html.



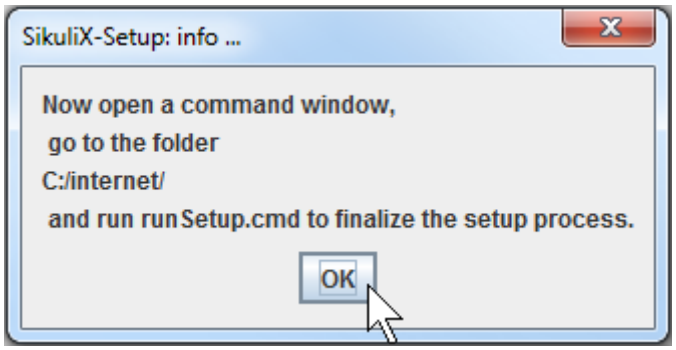
Die Installation beginnt unter Windows mit einem Doppelklick auf der heruntergeladenen Datei. Ansonsten wird das Programm in einer Konsole mit davor stehendem „java – jar“ aufgerufen. Die Datei sollte sich dabei bereits in dem Ordner befinden, in dem Sikuli installiert werden soll, im Beispiel C:\internet, wobei C:\sikuli sicherlich ein bessere Name ist.



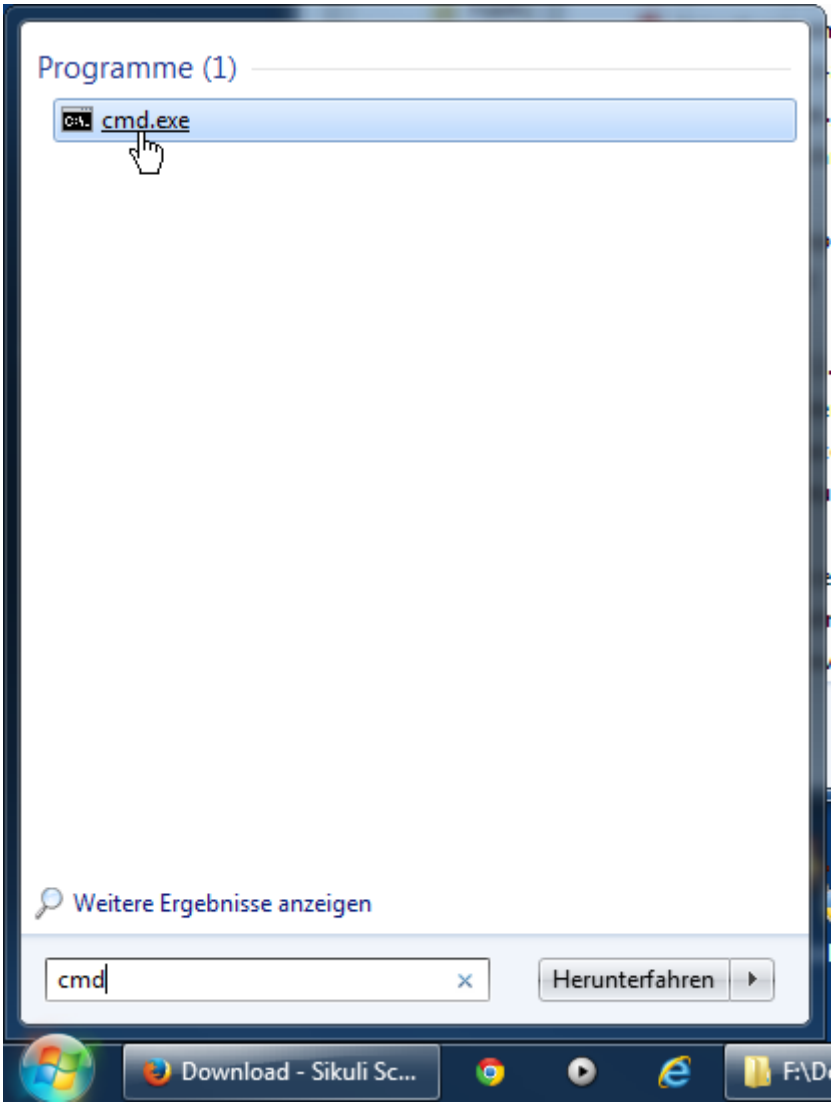
Auf Meldungen von Schutzprogrammen wird hier nicht eingegangen.



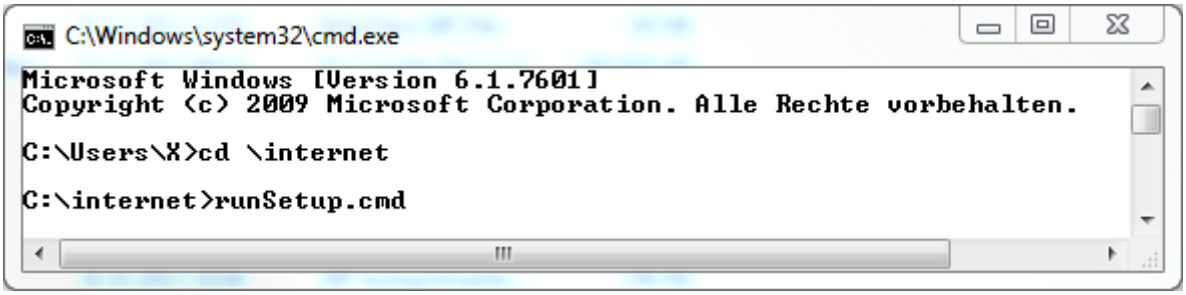
Die aufgerufene Datei befand sich in einem Ordner C:\internet, in den man jetzt mit einem Konsolenfenster wechseln soll.



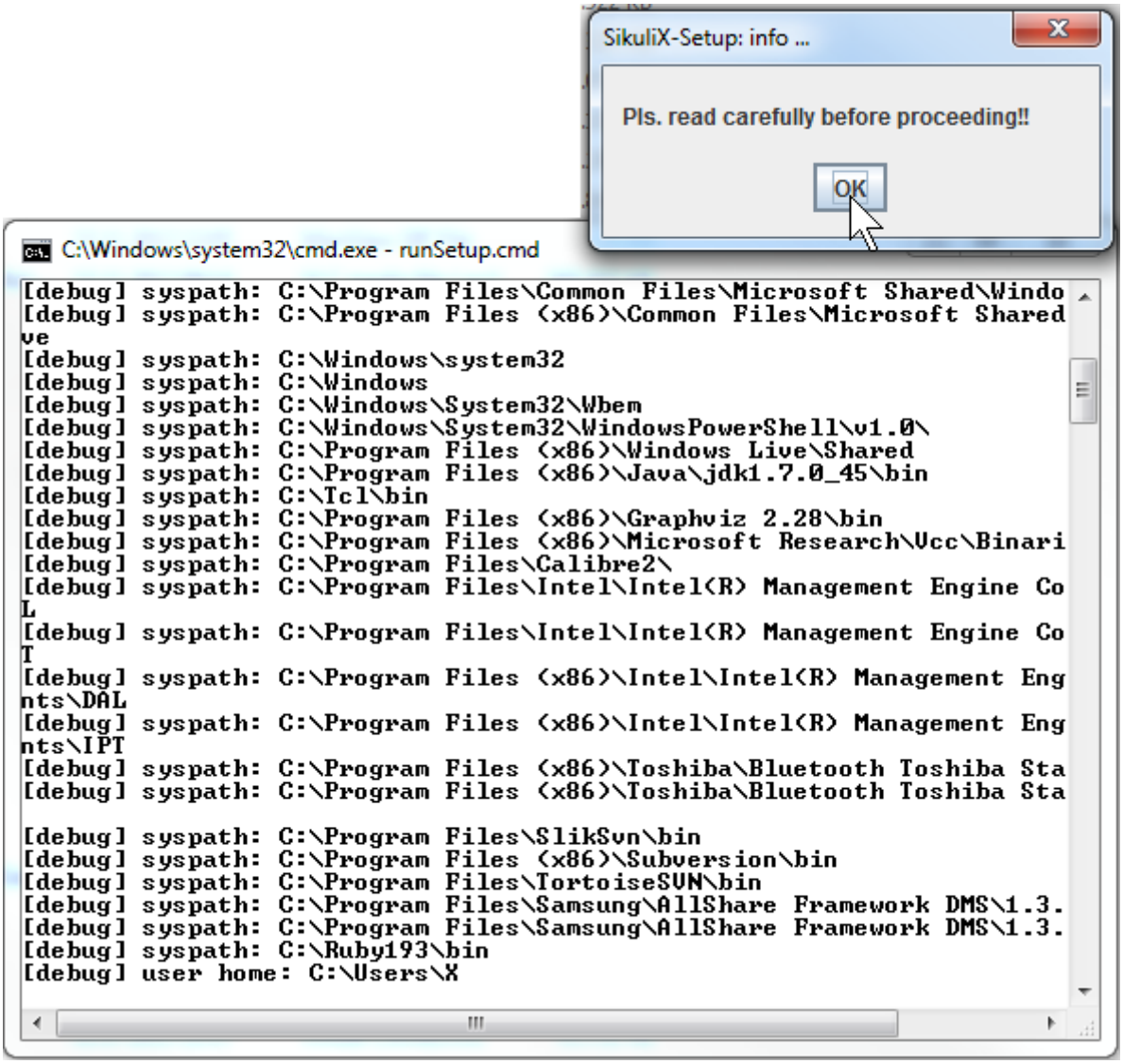
Dazu wird z. B. der Start.Knopf gedrückt, unten „cmd“ eingetippt und dann cmd.exe gestartet.



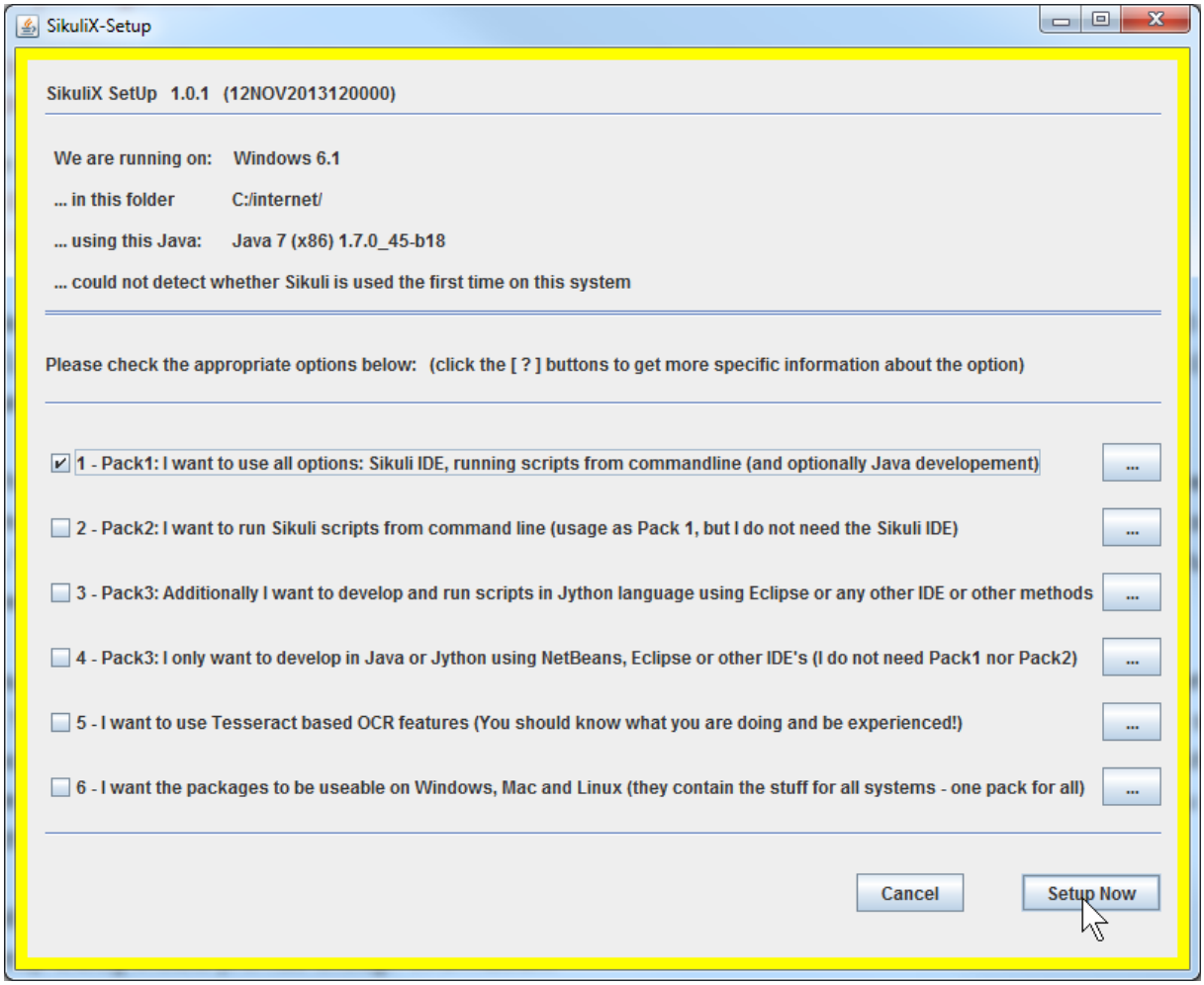
Es wird das Programm runSetup.cmd ausgeführt, bei Problemen sollte man dies als Administrator durchführen.



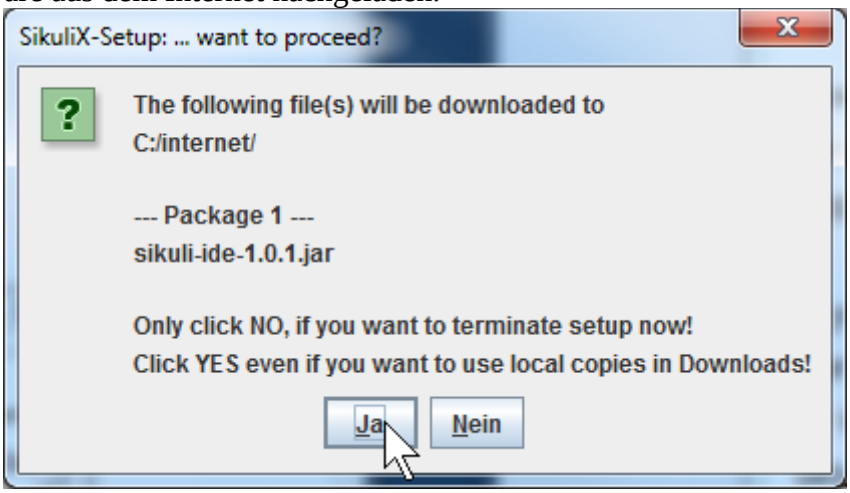
Man sieht einige Meldungen und wird aufgefordert diese anzusehen und dies mit „OK“ zu bestätigen. Die Meldungen zeigen im Wesentlichen die Einträge der PATH-Variablen.



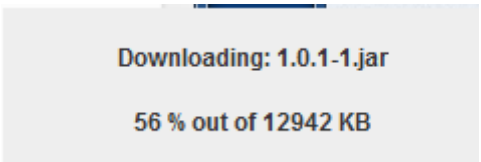
Man kann dan auswählen, was genau installiert werden soll, typischerweise wird Punkt 1 und dann „Setup Now“ angeklickt.



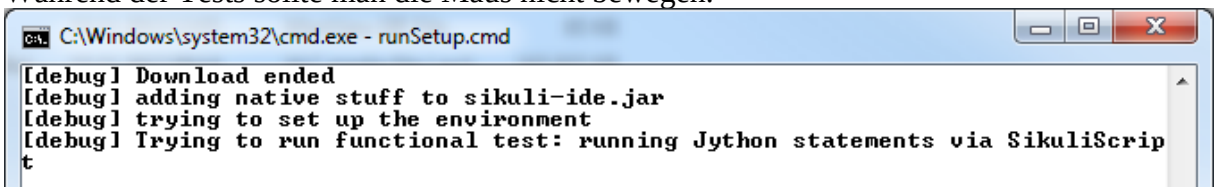
Es wird Software aus dem Internet nachgeladen.



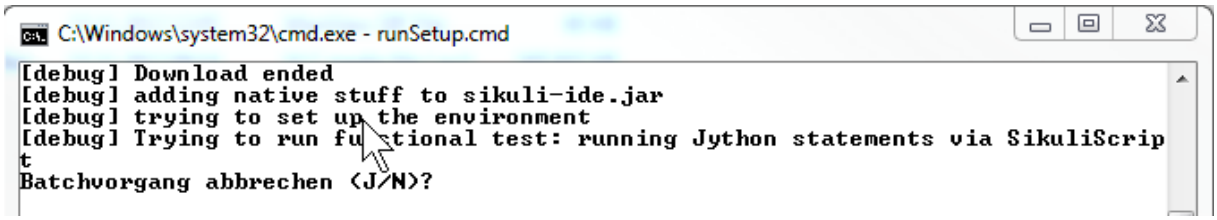
Der Download dauert etwas



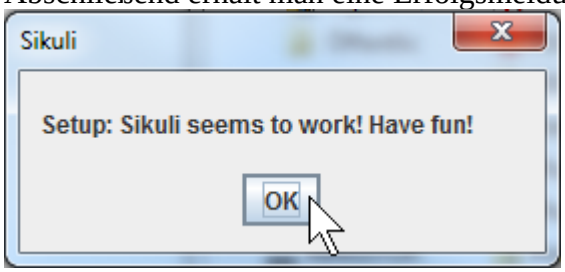
Es werden einige Tests durchgeführt, über die man auch in der Konsole informiert wird. Während der Tests sollte man die Maus nicht bewegen.



Bei Problemen drückt man in der Konsole Strg-C, um das Installationsprogramm abzubrechen und neu zu starten.



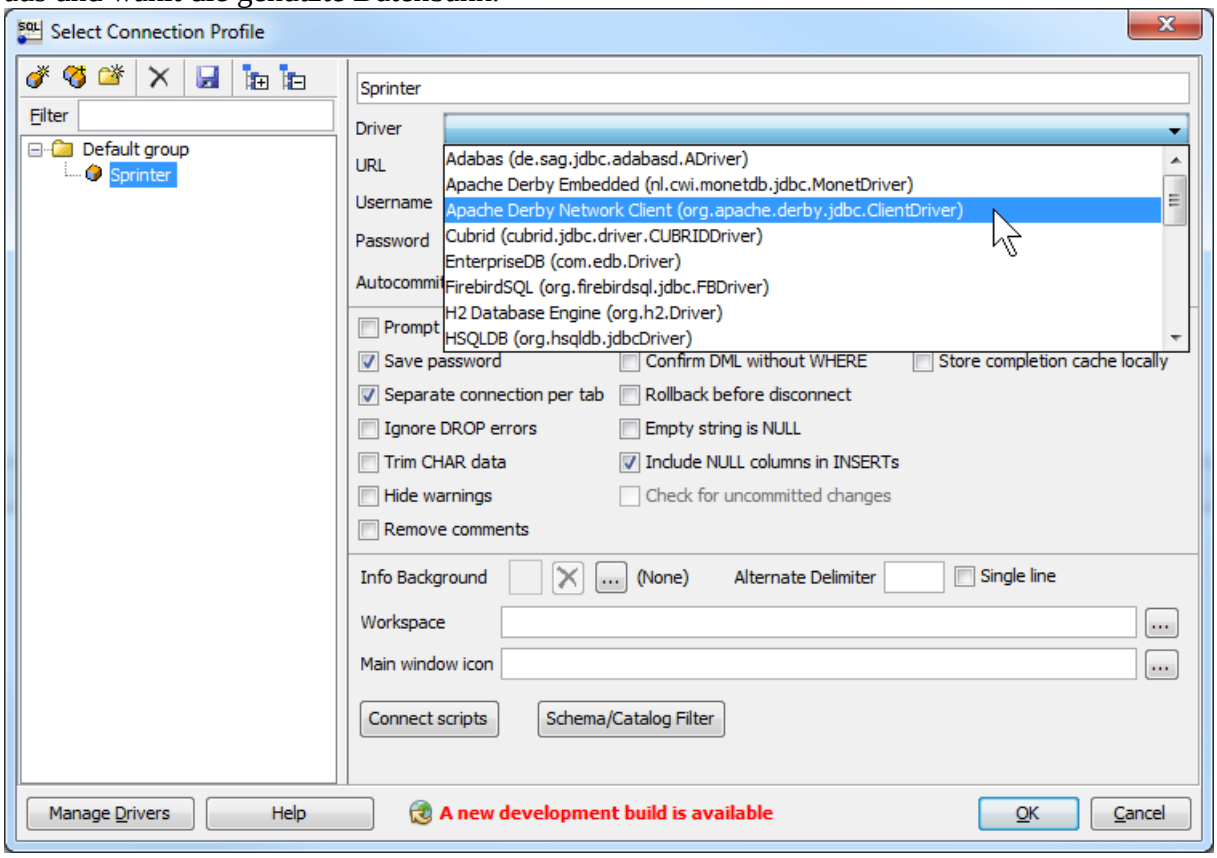
Abschließend erhält man eine Erfolgsmeldung.



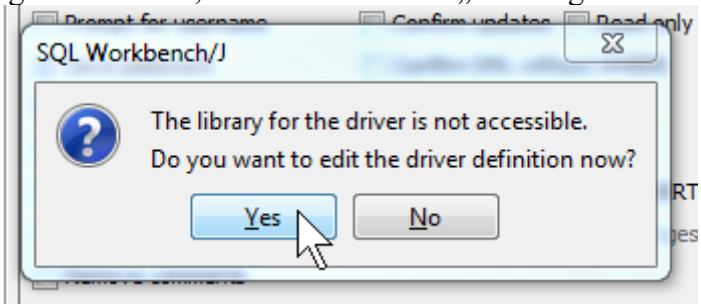
Zum Start des Programms wird ein Doppelklick auf „sikuli-die.jar“ gemacht,

G SQL Workbench / J (Build 116)

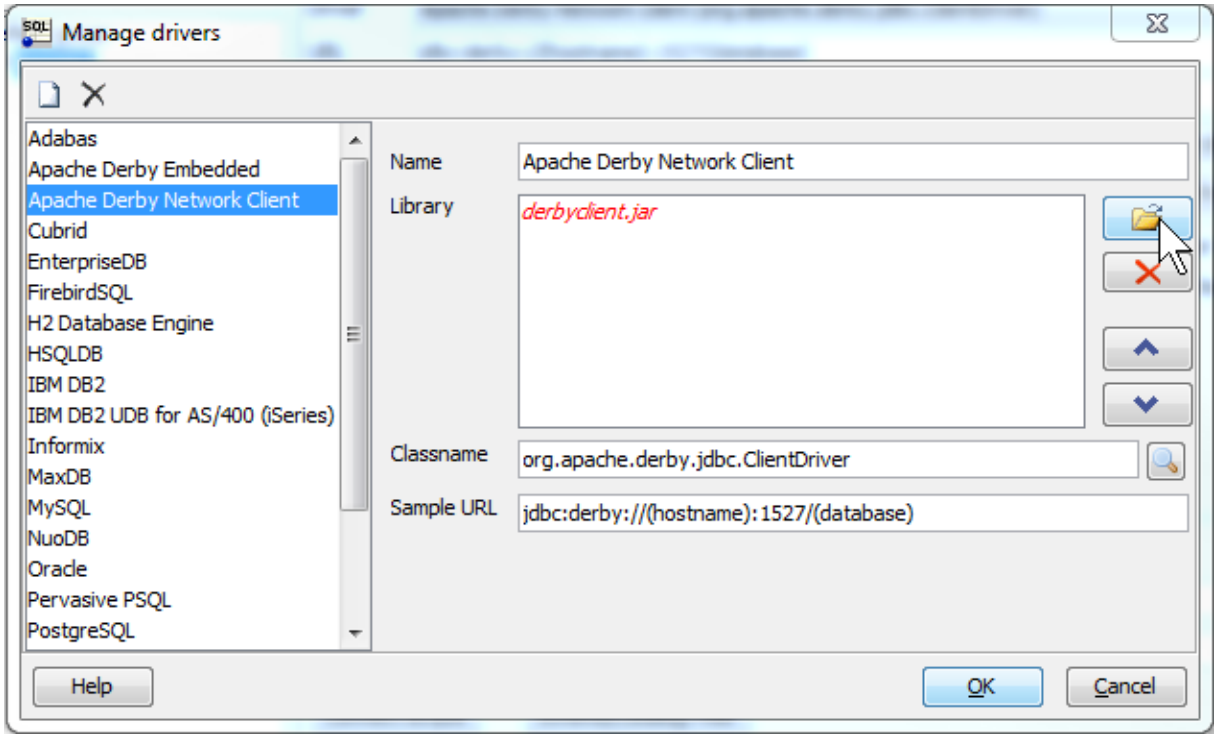
SQL Workbench / J (http://www.sql-workbench.net/) ist ein Datenbankwerkzeug, mit dem man u. a. sich recht einfach über existierende Tabellen und deren Inhalte informieren kann, was auch im Fokus dieser Beschreibung steht.
Nach dem ersten Start muss eine Verbindung aufgebaut werden. Dazu sollte bekannt sein, wo sich der JDBC-Treiber der genutzten Datenbank befindet. Man klappt zunächst das Feld Driver aus und wählt die genutzte Datenbank.



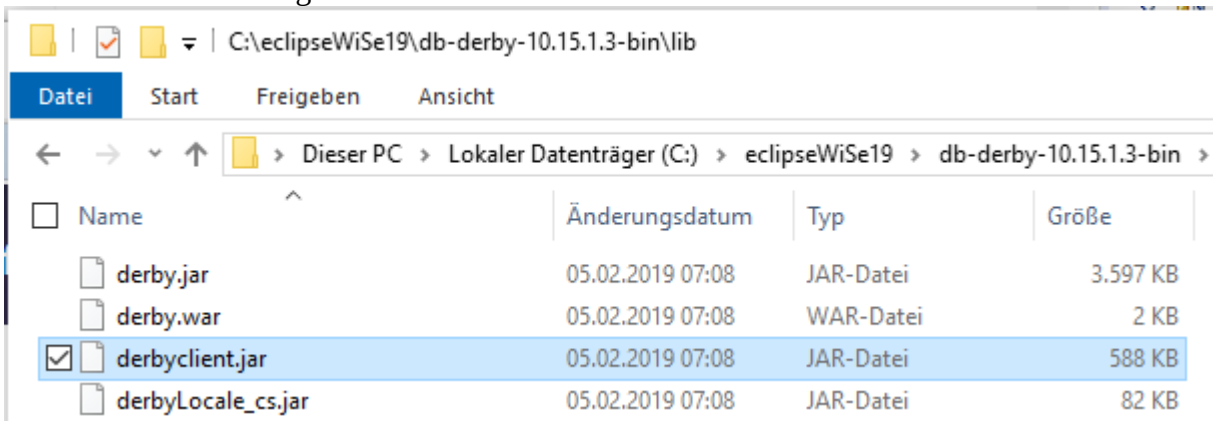
Da der Treiber nicht gefunden wird, kann man ihn über „Yes“ ergänzen.



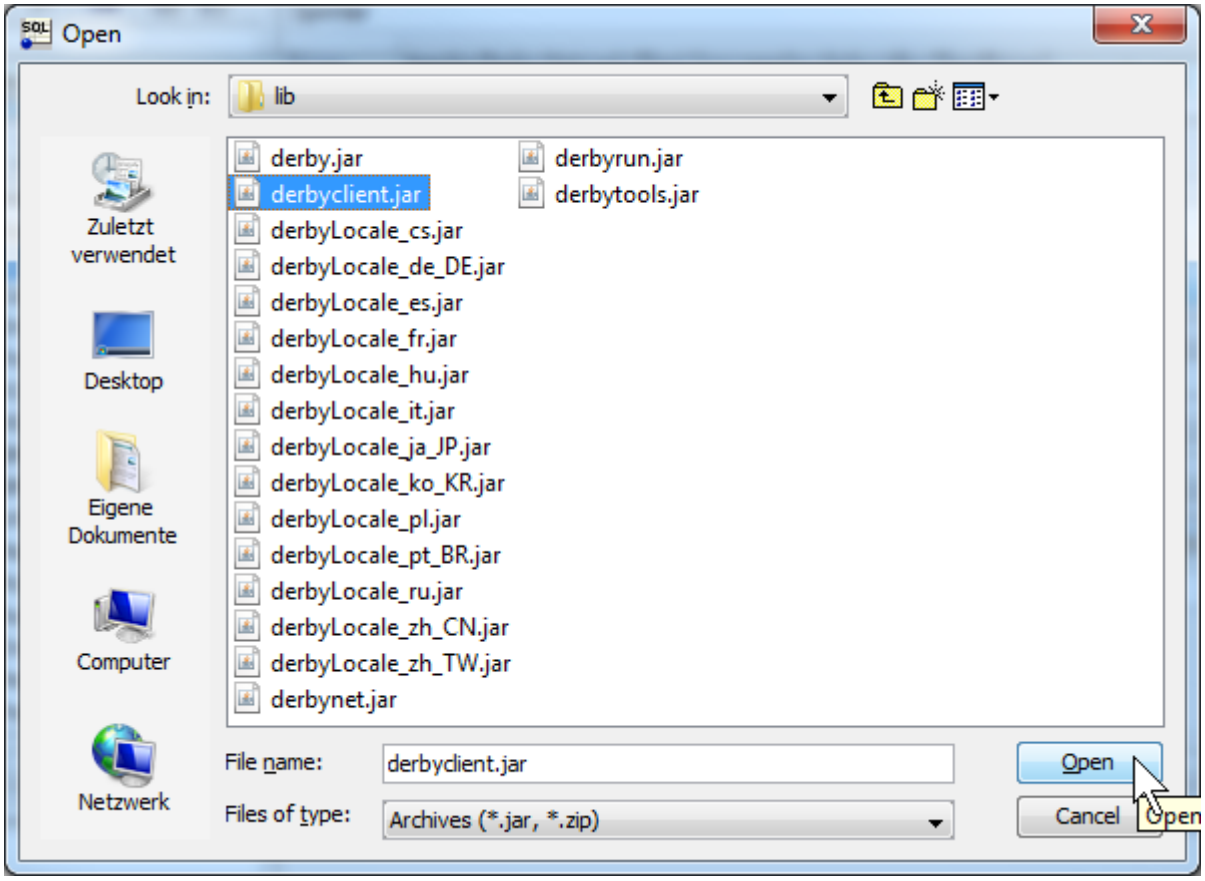
Über das Ordner-Symbol rechts-oben wird dann zum passenden JDBC-Treiber manövriert.



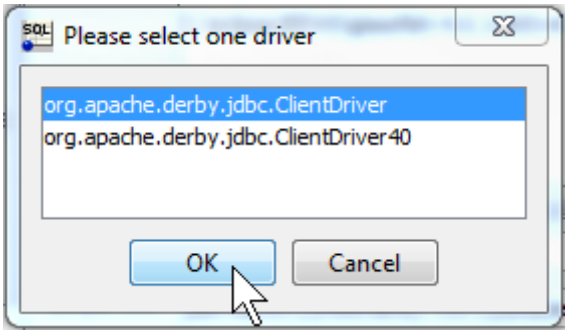
Im Installationsordner gibt es ein Verzeichnis lib mit dem Treiber.



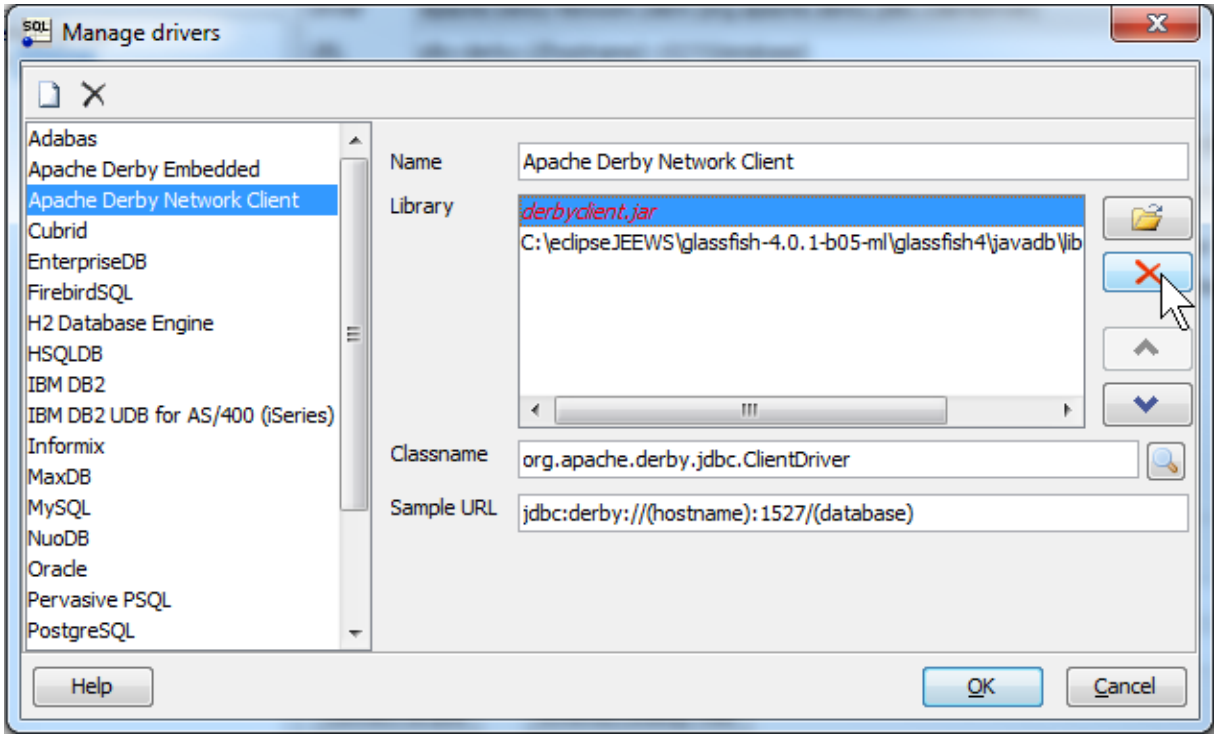
Beim GlassFish gibt es z. B. einen Unterordner javadb/lib, in dem sich ebenfalls der Treiber befindet.



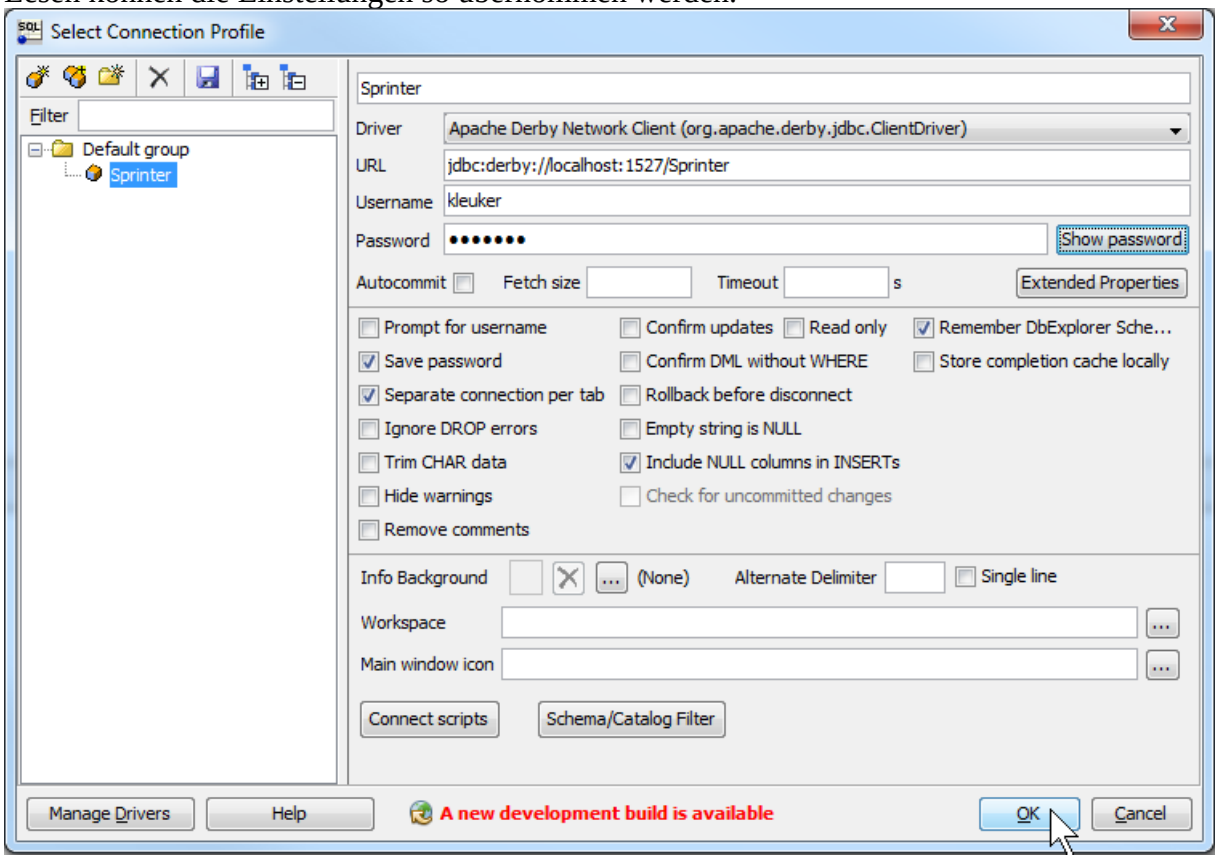
Man wählt den ersten Treiber.



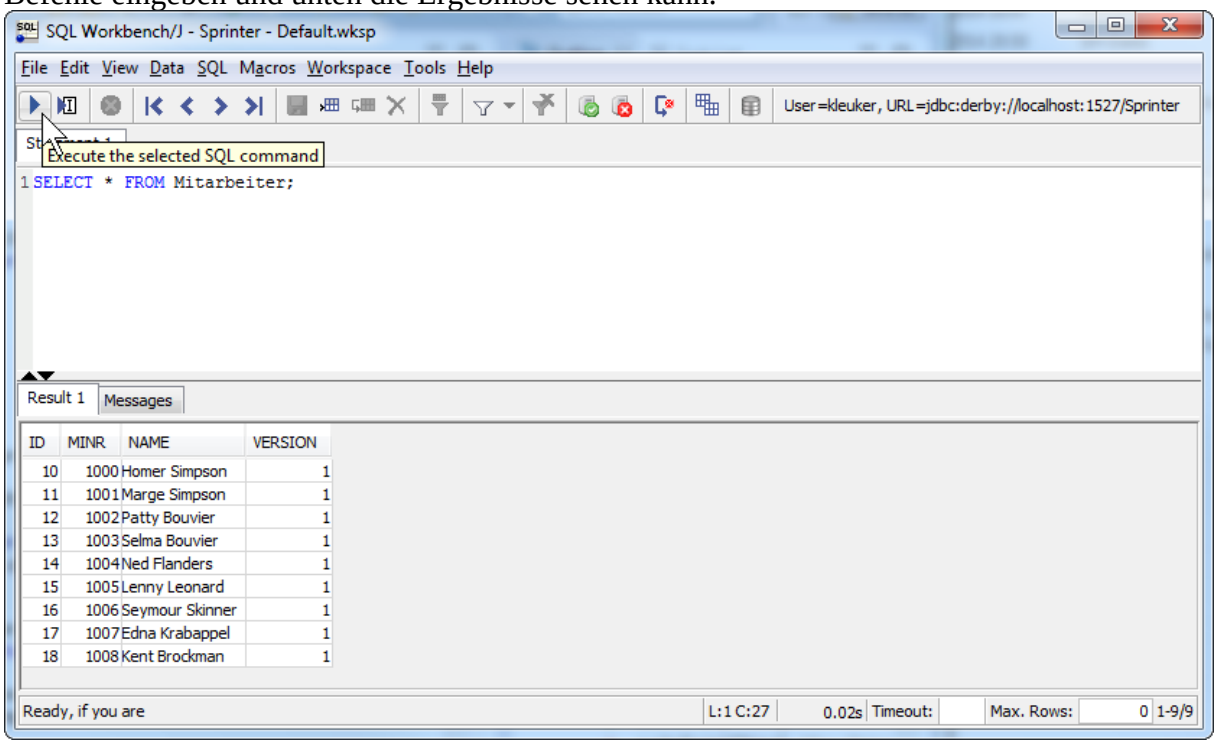
Dann wird der Eintrag in roter kursiver Schrift mit dem X am rechten Rand gelöscht und „OK“ gedrückt.



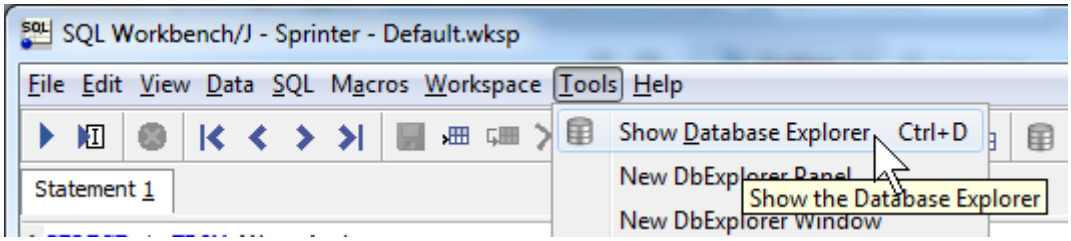
Nun wird die URL aktualisiert und Username sowie Password eingetragen. Die anderen Einstellungen hängen davon ab, wie man mit der Datenbank arbeiten möchte. Zum einfachen Lesen können die Einstellungen so übernommen werden.



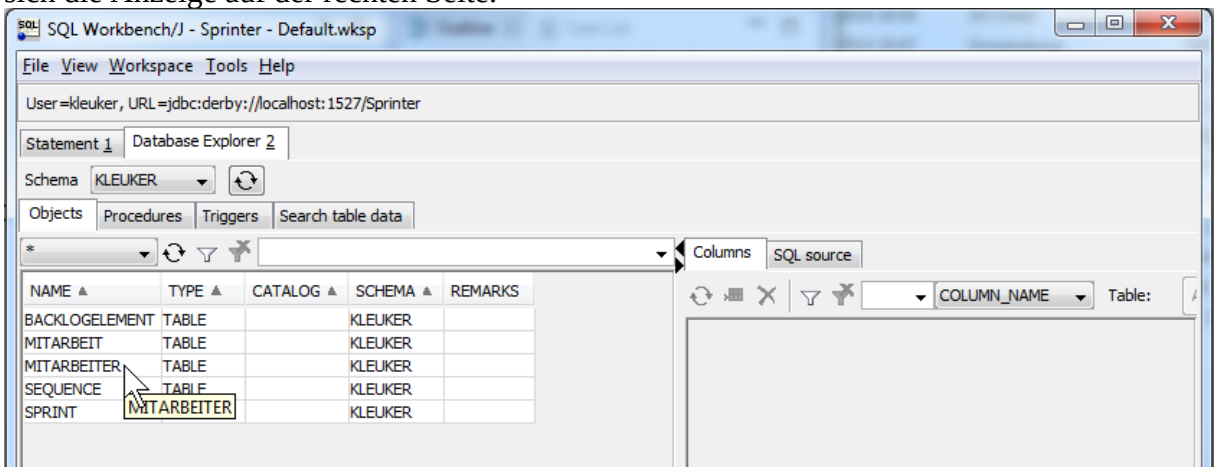
Danach wird die Verbindung aufgebaut und man erhält eine Arbeitsfläche, in der man oben Befehle eingeben und unten die Ergebnisse sehen kann.



Zum reinen Betrachten der Tabellen wird unter „Tools“ dann „Show Database Explorer“ genutzt.



Auf der linken Seite werden alle Tabellen angezeigt, Klickt man auf eine der Tabellen, ändert sich die Anzeige auf der rechten Seite.



Der mittlere Balken ist etwas holperig über die kleinen Pfeile steuerbar, so dass auch die folgende Abbildung möglich wird. Dabei kann man auf der rechten Seite durch die oberen Reiter die Informationen auswählen, die man betrachten möchte. Das kleine kreisartige Symbol links auf der rechten Seite ermöglicht ein „Refresh“, mit dem die Daten aktualisiert werden.