Eclipse 3.1.0

Eine Einführung

Inhaltsverzeichnis

1 Einführung in Eclipse

1.1 Download und erster Start

Die Web-Seite von Eclipse finden Sie unter *http://www.eclipse.org/*. Die aktuelle Version von Eclipse ist die 3.1 (27.07.2005). Eclipse laden Sie über den Link *Eclipse SDK 3.1* unter *http://www.eclipse.org/downloads/index.php* oder direkt unter

```
http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops/R-3.1-200506271435/eclipse-SDK-3.1-win32.zip
```

Eclipse benötigt mindestens ein installiertes JRE 1.4.2.

Die Installation erfolgt durch das einfache Entpacken der ZIP-Datei in einen Ordner Ihrer Wahl. In diesem Ordner wird ein Verzeichnis *..\eclipse* erzeugt. Darin finden Sie die Datei *eclipse.exe*, welche Eclipse unter Windows startet.

Nach dem erstmaligen Start von Eclipse werden Sie gefragt, wo Ihr Arbeitsplatz eingerichtet werden soll. Standardmäßig wird das Verzeichnis *C:\Dokumente und Einstellungen\[Benutzername]\workspace* vorgeschlagen. Damit Sie nicht jedes mal beim Start danach gefragt werden, markieren Sie die Option USE THIS AS THE DEFAULT AND DO NOT ASK AGAIN.

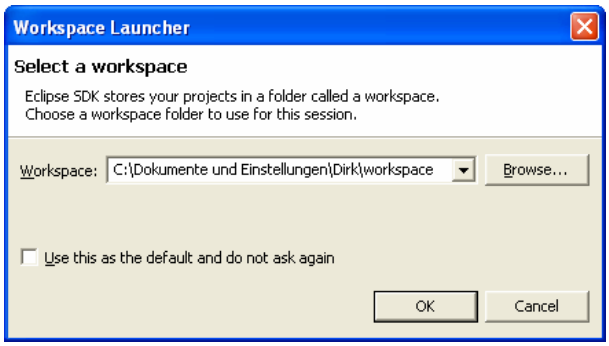


Abb. 1.1 Workspace auswählen

Nach der Bestätigung mit OK wird Eclipse mit der Welcome-Seite geöffnet.

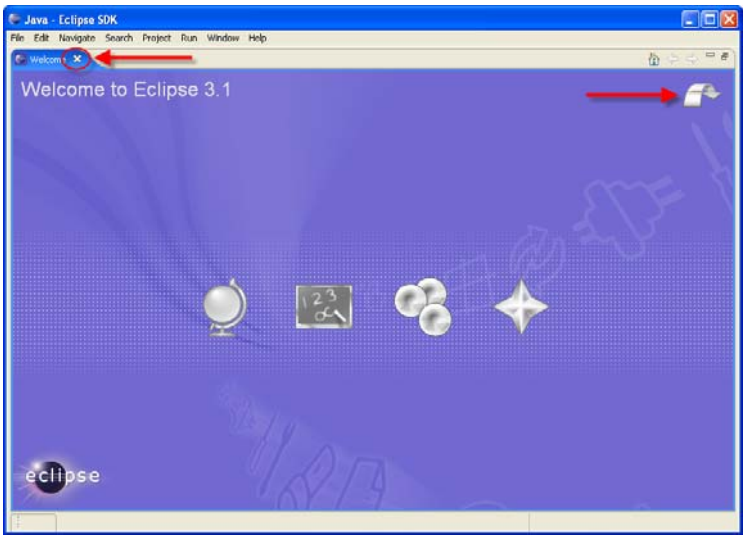
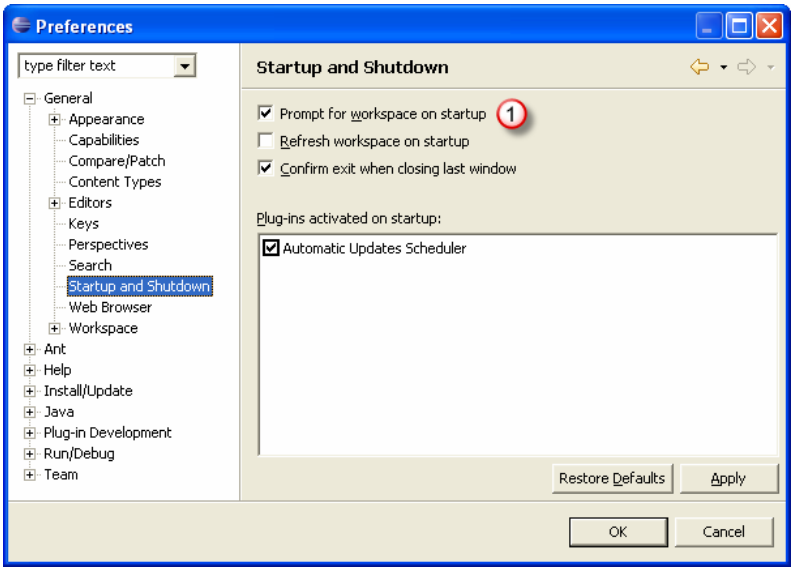


Abb. 1.2 Eclipse-Willkommenseite

Die Anzeige des Dialogfeldes zur Auswahl des Workspaces stellen Sie unter dem Menüpunkt WINDOW - PREFERENCES, Kategorie WORKBENCH - STARTUP AND SHUTDOWN ein.



Die Welcome-Seite enthält in der Mitte 4 Symbole, über die Sie erste Kontakte mit Eclipse knüpfen können. Die Willkommenseite schließen Sie entweder über das Kreuz auf dem Registerreiter links oder den Pfeil oben rechts.

Die Willkommenseite können Sie jederzeit über den Menüpunkt HELP - WELCOME anzeigen.

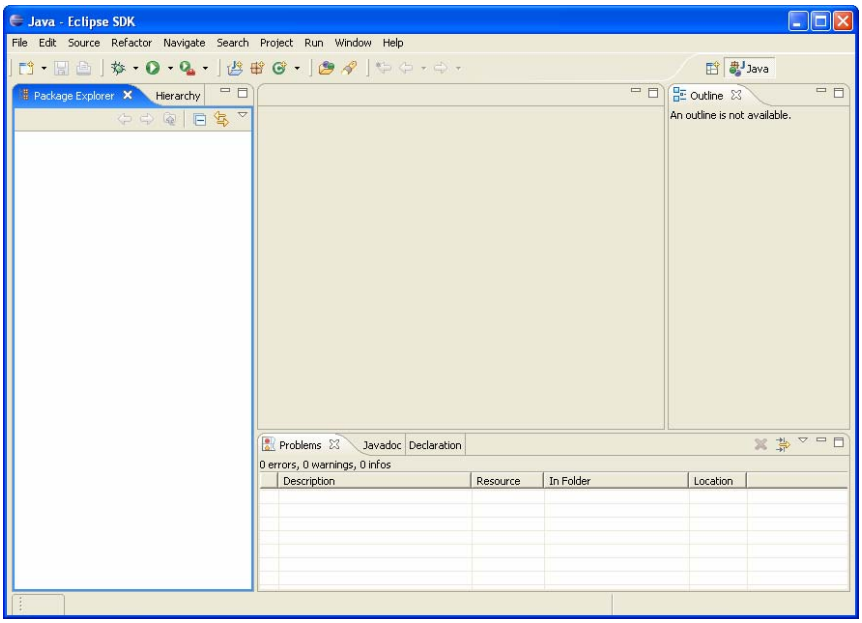


Abb. 1.3 Eclipse-IDE ohne ein geöffnetes Projekt

Die Anzeige der Workbench ist nach dem ersten Start sehr aufgeräumt. Um sie mit Leben zu füllen, müssen Sie ein Projekt erstellen. Vorher werden aber erst ein paar Einstellungen vorgenommen.

Den aktuellen Workspace (d.h. das Arbeitsverzeichnis) ändern Sie über den Menüpunkt FILE - SWITCH WORKSPACE.

1.2 Eclipse konfigurieren

Rufen Sie den Menüpunkt WINDOW - PREFERENCES auf.

Kategorie / Subkategorie	**Einstellung - Beschreibung**
GENERAL - WORKSPACE	BUILD AUTOMATICALLY - Beim Speichern werden die Projekte automatisch aktualisiert / übersetzt.
GENERAL	ALWAYS RUN IN BACKGROUND - Lang anhaltende Operationen werden im Hintergrund ausgeführt.
JAVA - CODE STYLE - FORMATTER	Klicken Sie auf SHOW um einen erweiterten Dialog anzuzeigen. Interessant ist die Angabe der Tabulatorbreite auf dem Register INDENTION und die Angabe der Ausrichtung der Klammern unter BRACES. Nachdem Sie die Änderungen durchgeführt haben werden Sie aufgefordert, einen neuen Namen für das Profil anzugeben.
GENERAL - EDITORS - TEXT EDITORS	DISPLAY TAB WIDTH - Tabulatorbreite
	SHOW LINE NUMBERS - Zeilennummern anzeigen
JAVA - INSTALLED JRES	Hier können Sie ein weiteres JRE hinzufügen.

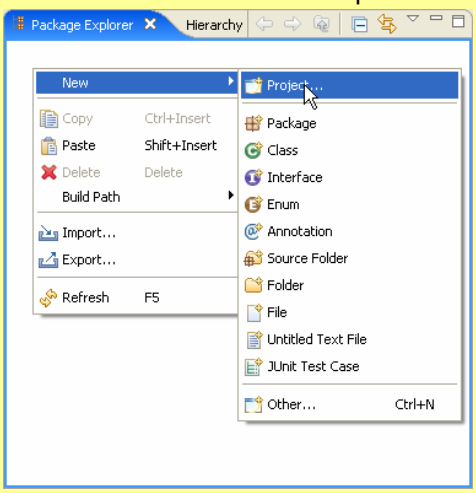
Über die Menüpunkte DATEI - IMPORT und DATEI - EXPORT können die Einstellungen (und anderes) für andere Mitarbeiter bereitgestellt werden.

1.2.1 Begrifflichkeiten

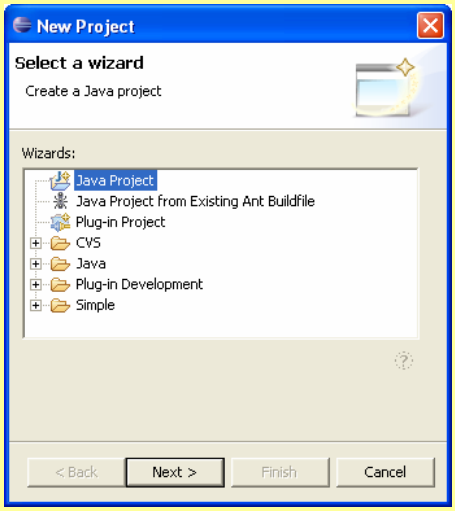
Workbench	Die Eclipse-IDE wird als Workbench bezeichnet.
Workspaces	Ein Workspace ist ein Verzeichnis, in dem Ihre Projekte verwaltet werden.
Perspektiven	Eine Perspektive ist eine Sammlung von Einstellungen und Fensteranordnungen in der Workbench.
Projekte	Ein Projekt kann separat konfiguriert werden und ist Bestandteil des aktuellen Workspaces.

2 Projekte erstellen

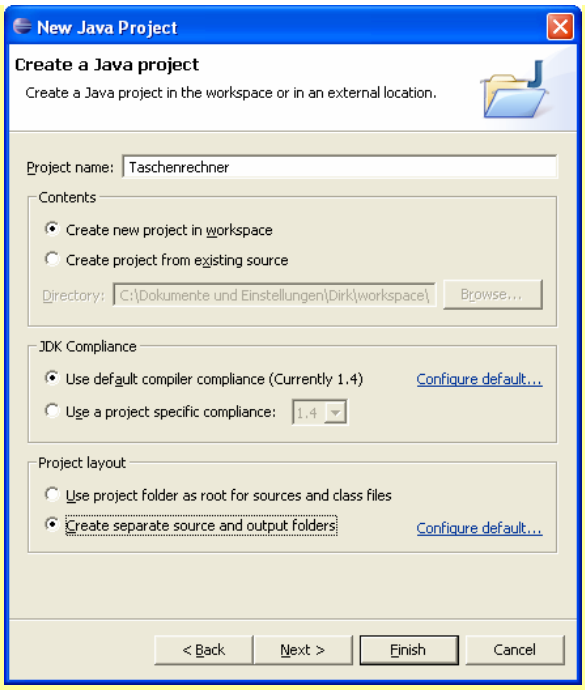
- Klicken Sie im Kontextmenü des Navigatorfensters auf den Menüpunkt NEW - PROJECT.
- Alternativ rufen Sie den Menüpunkt FILE - NEW - PROJECT des Hauptmenüs auf.



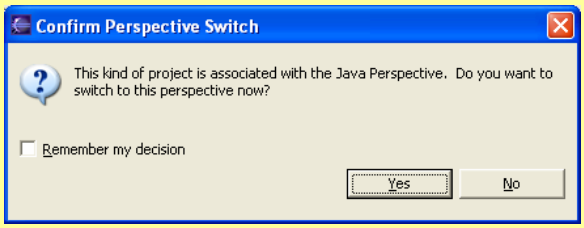
- Markieren Sie im Fenster NEW PROJECT den Eintrag JAVA PROJECT und klicken Sie auf NEXT.



- Vergeben Sie einen Projektnamen (z.B. Taschenrechner), markieren Sie die Option CREATE NEW PROJECT IN WORKSPACE, um das Projekt im aktuellen Workspace zu erstellen und markieren Sie außerdem die Option CREATE SEPARATE SOURCE AND OUTPUT FOLDERS, um getrennte Ordner für den Java-Code und die Class-Dateien zu erhalten.



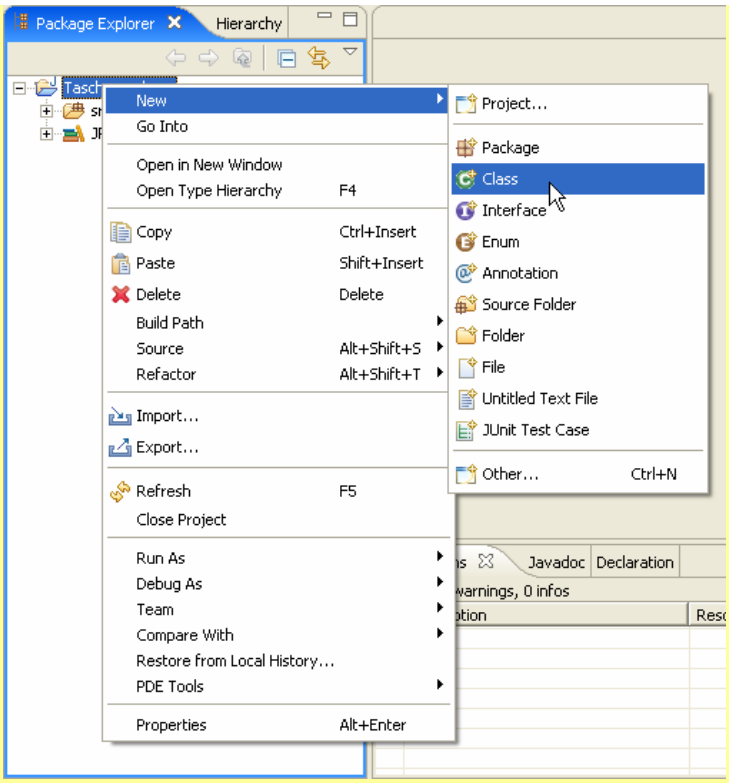
- Klicken Sie auf FINISH.
- Es wird eventuell ein Hinweisfenster angezeigt, dass dieser Projekttyp mit der Java-Perspektive verknüpft ist. Bestätigen Sie mit YES.



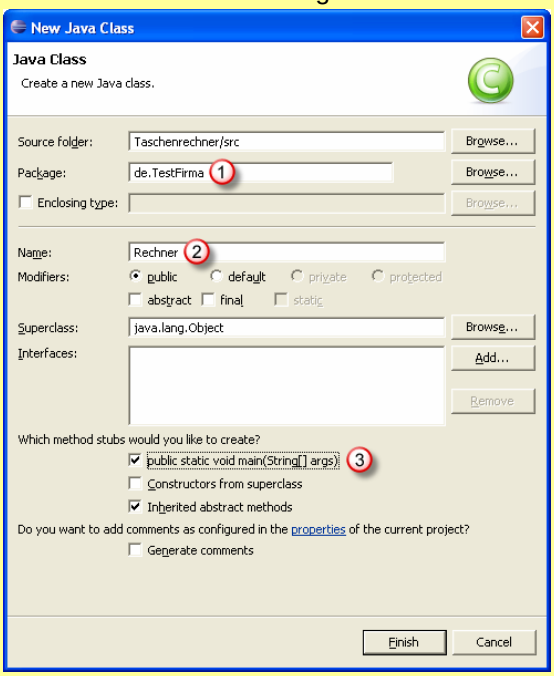
- Damit ist die Projekterstellung abgeschlossen. Bevor die weiteren Teile der Workbench besprochen werden, soll eine Klasse erzeugt werden.

2.1 Klasse erstellen

- Rufen Sie im Package Explorer den Kontextmenüpunkt NEW - CLASS auf.
- Alternativ verwenden Sie den Menüpunkt FILE - NEW - CLASS.



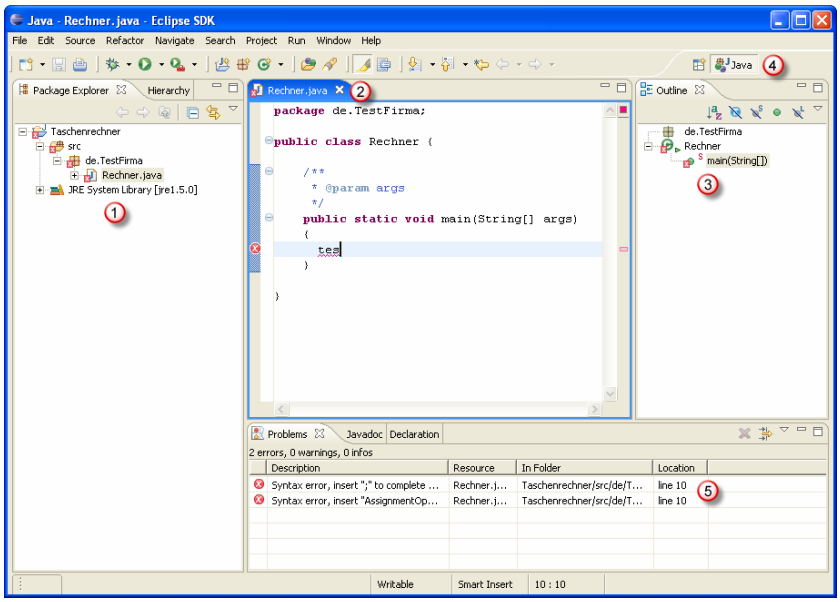
- Im Dialogfenster NEW JAVA CLASS geben Sie bei (1) optional einen Packagenamen an und unter (2) den Klassennamen. Wenn die Klasse eine `main()`-Methode besitzen soll, machen Sie bei (3) noch einen Haken. Bestätigen Sie mit FINISH.



- Die neue Klasse wird im Java-Editor geöffnet. Jetzt ist ein guter Zeitpunkt, die einzelnen Fenster der Workbench näher zu betrachten.

2.2 Aufbau der Workbench

Ist ein Projekt geöffnet bzw. sind Projekte im Workspace verfügbar, kommt etwas mehr Leben in die Workbench. Die folgende Abbildung zeigt die Fenster, die in der Java-Perspektive sichtbar sind. Im aktiven Fenster ist das Register blau gefärbt.



Unter (4) können Sie zwischen verschiedenen Perspektiven umschalten. Weitere Perspektiven können Sie unter dem Menüpunkt WINDOW - OPEN PERSPECTIVE oder über das linke Symbol unter (4) öffnen.

2.2.1 Package Explorer

Der Package Explorer zeigt alle Projekte und Dateien des aktuellen Workspaces an. Neue Klassen werden im aktuellen Projekt im Verzeichnis *src* angelegt, da beim Erstellen des Projekts die Option CREATE SEPARATE SOURCE AND OUTPUT FOLDERS markiert wurde. Dadurch ist jetzt die folgende Verzeichnisstruktur entstanden:

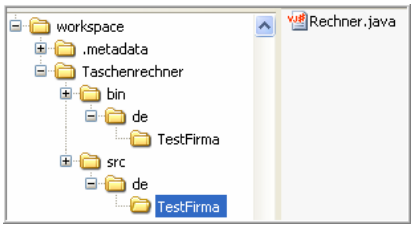


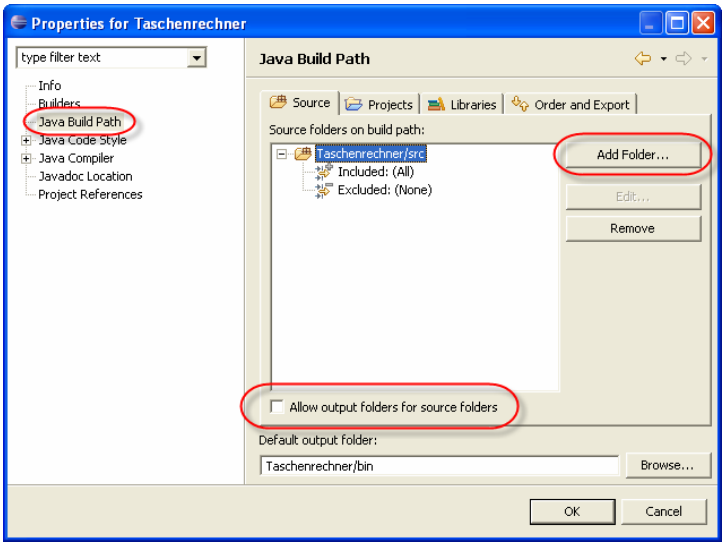
Abb. 2.4

- Im Workspace-Ordner wurde ein neuer Ordner erstellt, der den Projektnamen trägt.
- Darunter wurden zwei Verzeichnisse *src* und *bin* erstellt, welche die Source-Dateien und die übersetzten Class-Dateien enthalten. Damit haben Sie eine gute Trennung zwischen den Dateien, die evt. in einer Versionsverwaltung gesichert werden müssen, und den Dateien, die Ihre Anwendung ausmachen.
- Da im Quelltext momentan ein Fehler vorliegt, erhalten die Symbole im Package Explorer ein rotes Symbol links unten.

Der Package Explorer zeigt nur die für das Projekt notwendigen Dateien an. Erzeugte Class-Dateien oder das *bin*-Verzeichnis werden nicht angezeigt.

Unter dem Menüpunkt PROJECT - PROPERTIES können Sie die Ordnerstruktur für Source- und Ausgabeordner jederzeit ändern. Wechseln Sie in das Register SOURCE und markieren Sie die Option ALLOW OUTPUT ..., um keine getrennten Ordner zu verwenden. Haben Sie dagegen nur einen Ordner angelegt, können Sie über ADD FOLDER einen neuen Ordner anlegen.

Eclipse erzeugt dann automatisch einen *bin*-Ordner für die Ausgabe.



2.2.2 Java-Editor

Der Java-Editor dient zur Eingabe des Quelltextes. Sind mehrere Dateien geöffnet, werden standardmäßig mehrere Registerkarten angezeigt.

2.2.3 Outline-Fenster

Dieses Fenster dient zur Navigation im Source-Code im aktuell geöffneten Editorfenster. Es werden die Typen (Klassen, Interfaces), Methoden usw. angezeigt. Klicken Sie auf einen Eintrag, wird er im Editor selektiert. Umgekehrt wird der Fensterinhalt mit der aktuellen Position im Editor synchronisiert.

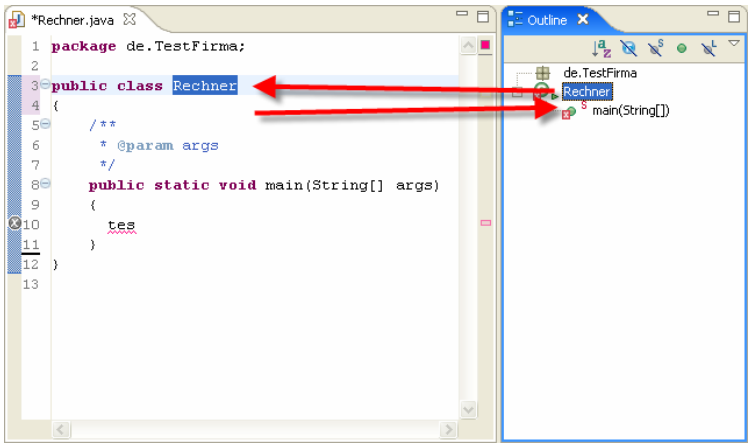


Abb. 2.5

2.2.4 Navigator-Fenster

In der Java-Perspektive wird das Navigatorfenster nicht angezeigt. Dazu wechseln Sie entweder in die Perspektive *Resource* oder Sie rufen den Menüpunkt WINDOW - SHOW VIEW - NAVIGATOR auf. Das Fenster zeigt alle Dateien des Workspaces an. Über das Kontextmenü können Sie diese auch bearbeiten, z.B. umbenennen.

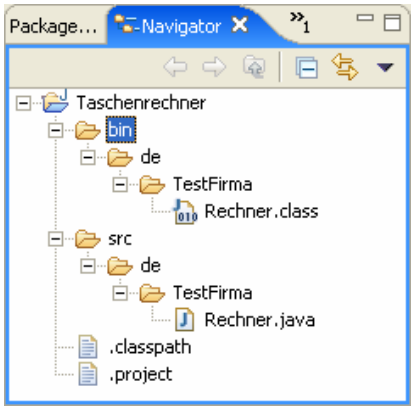


Abb. 2.6

2.2.5 Problems-Fenster

Wurden Fehler festgestellt, werden diese in diesem Fenster angezeigt. Durch einen Klick auf die Meldung gelangen Sie direkt zum fehlerhaften Code.

2.3 Sonstiges

Projekte können zwar geschlossen werden (Menüpunkt FILE - CLOSE), sie werden im Package Explorer aber immer noch im zugeklappten Zustand angezeigt. Allerdings beanspruchen geschlossene Projekte weniger Speicher.

3 Anwendungen übersetzen und ausführen

Fügen Sie in die Methode `main()` den Code `System.out.println("Hallo");` ein.

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

Info Obwohl der Code jetzt fehlerfrei ist, werden immer noch die Fehlermeldungen angezeigt. Erst wenn Sie die Datei speichern wird automatisch der Code im Hintergrund übersetzt und die Anzeige aktualisiert. Dadurch liegt auch sofort ein fertig übersetztes Programm vor.

3.1 Anwendungen übersetzen

Haben Sie unter WINDOW - PREFERENCES, Kategorie GENERAL - WORKSPACE die Option BUILD AUTOMATICALLY aktiviert, wird bei jedem Speichern das Projekt aktualisiert und übersetzt. Alternativ können Sie auch den Haken unter dem Menüpunkt PROJECT - BUILD AUTOMATICALLY setzen. Ist der Haken nicht gesetzt, stehen die Menüpunkt PROJECT - BUILD ALL und PROJECT - BUILD PROJECT zur Verfügung.

Info Wenn Sie nicht den automatischen Build nutzen, müssen Sie evt. noch unter WINDOWS - PREFERENCES, Kategorie GENERAL - WORKSPACE die Option SAVE AUTOMATICALLY BEFORE BUILD aktivieren.

3.2 Anwendungen ausführen

Damit eine Anwendung ausgeführt werden kann, muss normalerweise eine Konfiguration dazu erstellt werden. Allerdings kann eine Anwendung auch ohne diesen "Umweg" ausgeführt werden.

3.2.1 Anwendung schnell ausführen

Klicken Sie auf einen Knoten im Package Explorer und rufen Sie den Kontextmenüpunkt RUN AS - JAVA APPLICATION auf. Eclipse sucht nun in allen Dateien des Knotens und seinen Unterknoten nach Class-Dateien, die eine `main()`-Methode enthalten, also ausgeführt werden können. Wenn Sie z.B. auf den Projektknoten klicken, wird dazu das Projekt aber auch das gesamte JRE durchsucht.

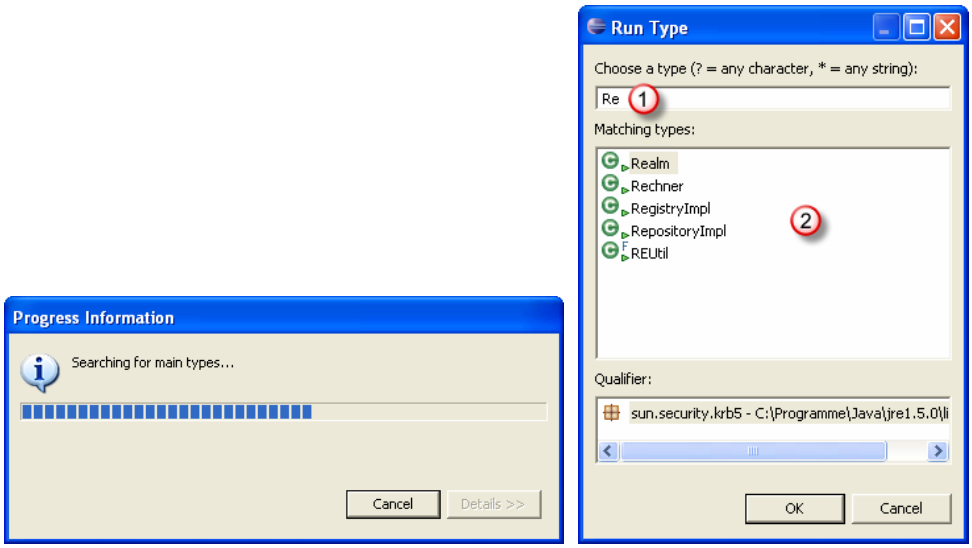


Abb. 3.7

Deshalb ist es besser, gleich die entsprechende Source-Datei auszuwählen und die Anwendung darüber zu starten. Wenn Sie im Fenster *Run Type* unter (1) Text eingeben, wird die Auswahl (2) darunter eingeschränkt.

3.2.2 Anwendung über eine Konfiguration starten

Haben Sie auf den Run-Button () in der Symbolleiste geklickt oder den Menüpunkt RUN - RUN aufgerufen, wird das folgende Fenster geöffnet.

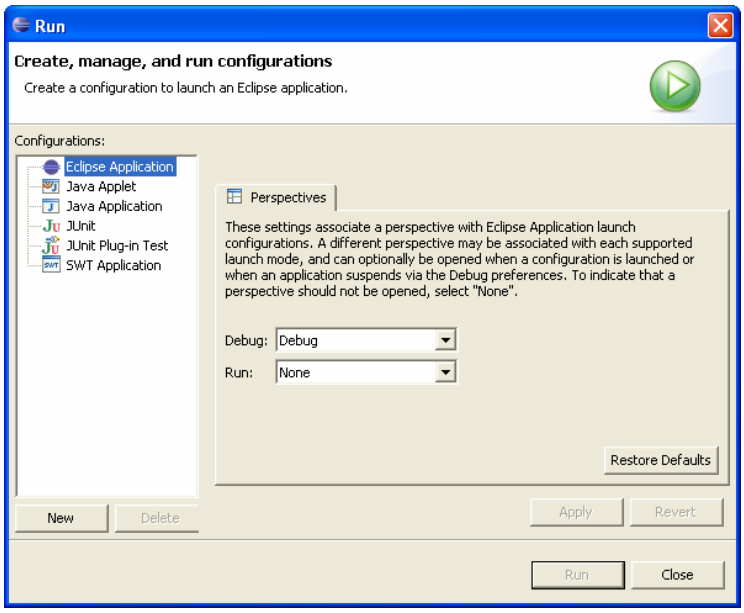


Abb. 3.8

Haben Sie das Projekt bereits vorher einmal gestartet befindet sich bereits eine Konfiguration unter dem Eintrag JAVA APPLICATION.

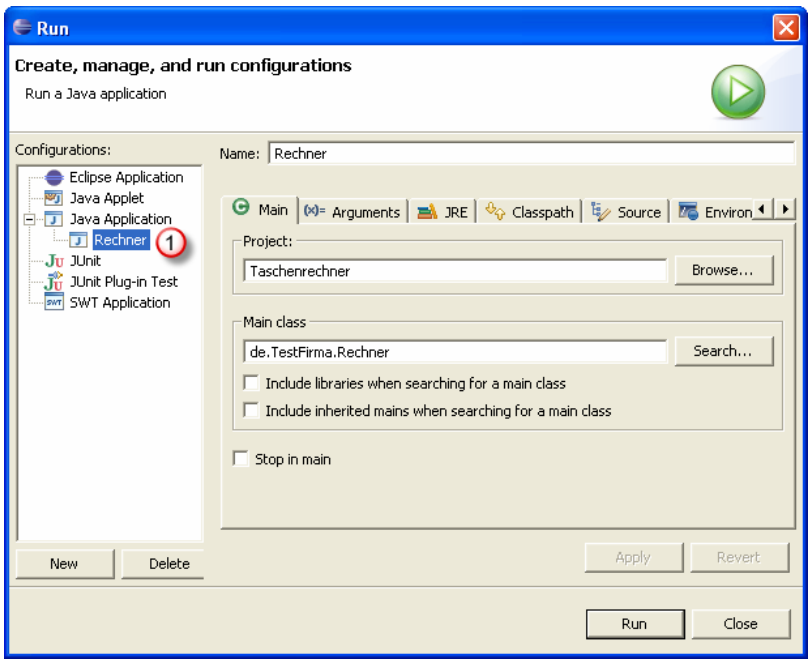
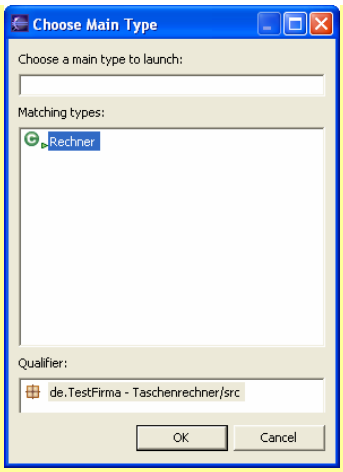
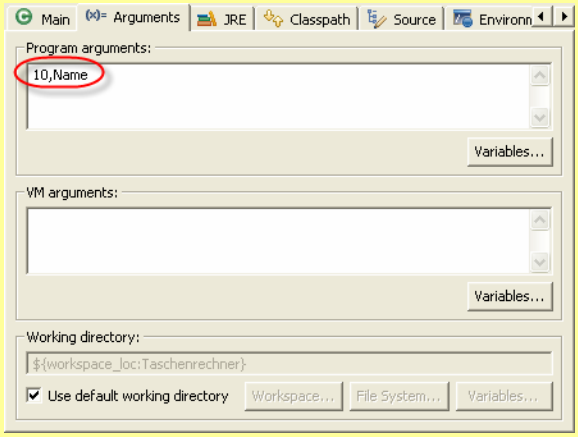


Abb. 3.9

- Klicken Sie im Dialog aus Abbildung 3.8 auf den Eintrag JAVA APPLICATION.
- Klicken Sie auf NEW.
- Vergeben Sie im Feld NAME einen Namen für die Konfiguration, z.B. *Taschenrechner*.
- Klicken Sie auf SEARCH, um die Hauptklasse Ihrer Anwendung auszuwählen und bestätigen Sie mit OK.



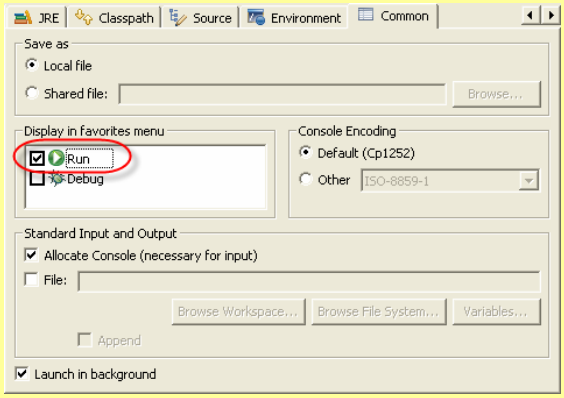
- Über die Register ARGUMENTS, CLASSPATH und SOURCE können Sie Parameter an die Anwendung übergeben, den Klassenpfad definieren und weitere Ordner mit SourceCode hinzufügen. Geben Sie 2 Argumente 10 und Name, getrennt durch Komma ein.



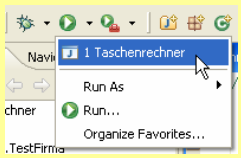
- Ändern Sie dazu den Code der Methode `main()` in

```
public static void main(String[] args)
{
  System.out.println("Hallo");
  for(int i = 0; i < args.length; i++)
    System.out.println(args[i]);
}
```

- Im letzten Register COMMON markieren Sie die Option RUN, damit die Konfiguration als Menüpunkt bei der Schaltfläche RUN erscheint.



=>



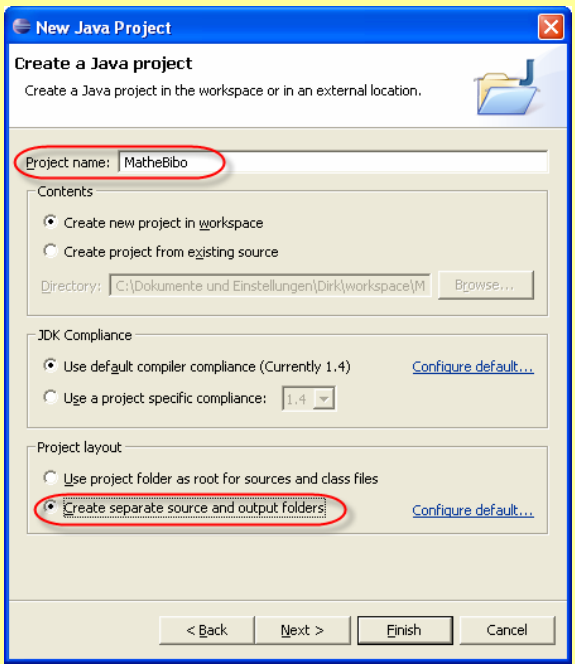
- Starten Sie jetzt die Anwendung. Da Sie Ausgaben auf der Console durchgeführt haben, wird im unteren Bereich ein weiteres Fenster CONSOLE geöffnet. Darin wird die Ausgabe durchgeführt.

4 Eine Bibliothek erstellen und verwenden

Jetzt soll ein Archiv erstellt werden, das eine Klasse `Mathe` enthält, die Methoden zum Addieren und Subtrahieren enthält. Das Archiv soll in der Anwendung *Rechner* genutzt werden.

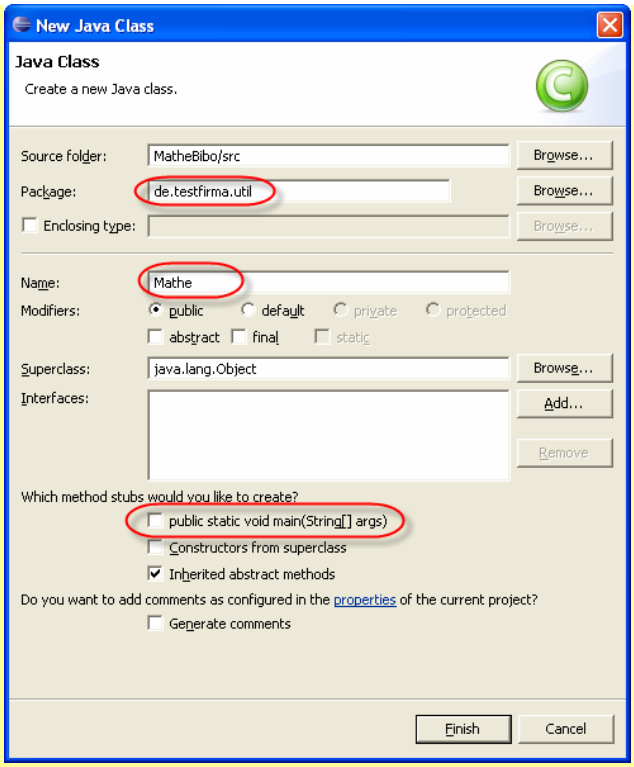
4.1 Projekt erstellen

- Rufen Sie den Menüpunkt FILE - NEW - PROJECT auf.
- Wählen Sie JAVA PROJECT und klicken Sie auf NEXT.
- Nennen Sie das Projekt MATHEBIBO, markieren Sie CREATE SEPARATE ... und bestätigen Sie mit FINISH.



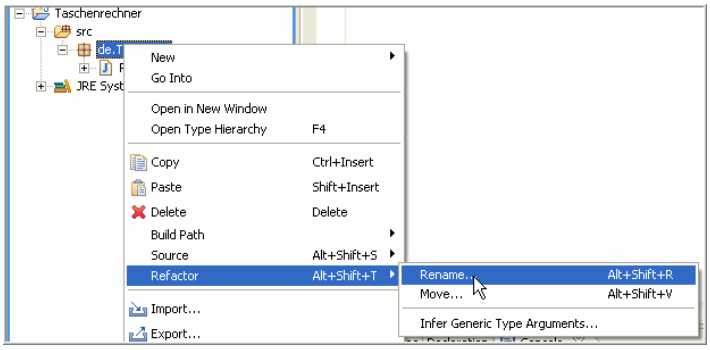
4.2 Klasse erstellen

- Rufen Sie im Kontextmenü des neuen Projekts im Package Explorer den Menüpunkt NEW - CLASS auf.
- Vergeben Sie den Packagenamen `de.testfirma.util` und benennen Sie die Klasse `Mathe`. Entfernen Sie gegebenenfalls die Markierung zum Erstellen der Methode `main()`.

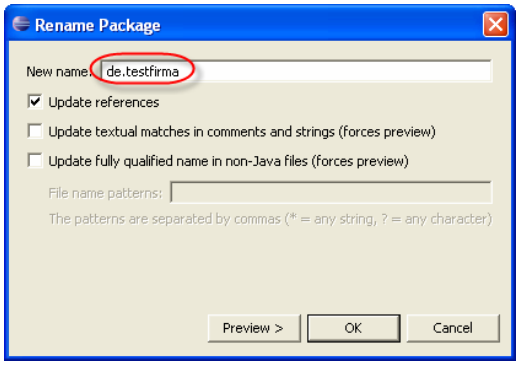


- Bestätigen Sie mit FINISH.

Beim Erstellen der Klasse `Rechner` ist Ihnen eventuell aufgefallen, dass der Packagename der Hauptanwendung nicht den allgemeinen Konventionen für Packagenamen entspricht. Diese sollten nämlich immer klein geschrieben werden. Klicken Sie auf das Packagesymbol `de.TestFirma` und rufen Sie den Menüpunkt REFACTOR - RENAME auf.



Geben Sie `de.testfirma` ein und bestätigen Sie mit OK.



Ignorieren Sie die angezeigte Meldung und bestätigen Sie mit CONTINUE. Neben den Verzeichnisnamen wurde auch die `package`-Anweisung in der Datei *Rechner.java* aktualisiert.

Nach diesem kleinen Ausflug ins das Refactoring geht es weiter mit der Erstellung der Bibliothek.

- Implementieren Sie die Klasse `Mathe` wie folgt.

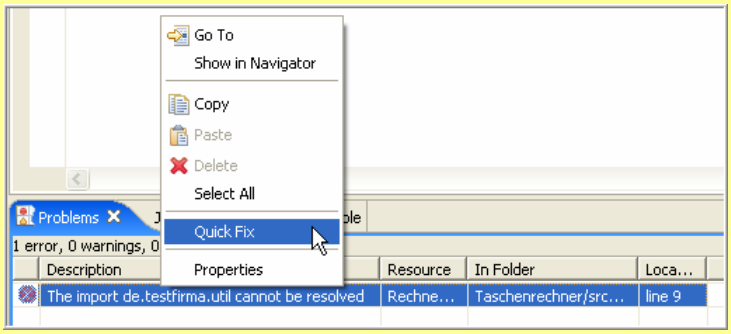
```
public class Mathe
{
  public static int Add(int zahl1, int zahl2)
  {
    return zahl1 + zahl2;
  }
  public static int Minus(int zahl1, int zahl2)
  {
    return zahl1 - zahl2;
  }
}
```

- Fügen Sie in der Klasse `Rechner` des Projekts *Taschenrechner* die folgende `import`-Anweisung ein.

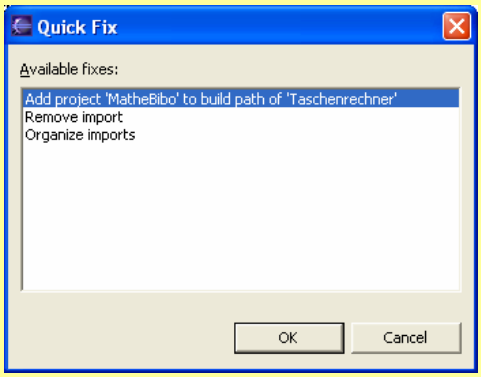
```
import de.testfirma.util.*;
```

  und speichern Sie die Datei. Im *Problems*-Fenster wird ein Fehler angezeigt, da sich das Projekt mit der Klasse `Mathe` nicht im Klassenpfad befindet.

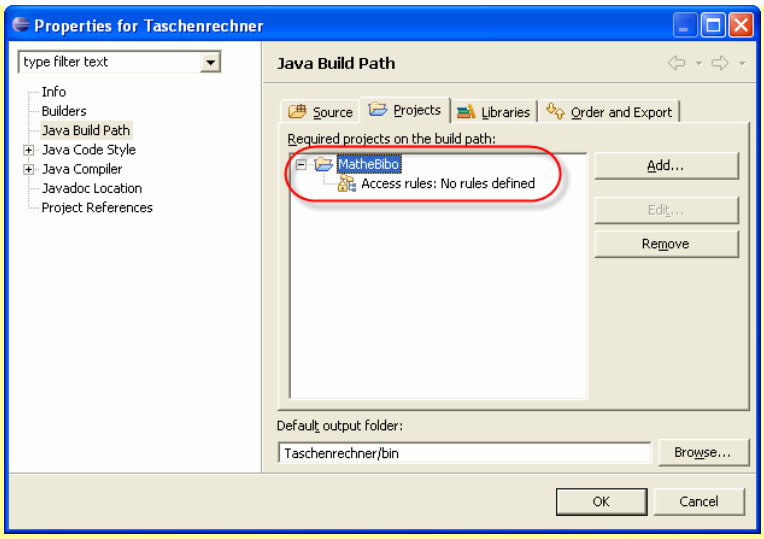
- Klicken Sie mit der rechten Maustaste auf die Fehlermeldung und rufen Sie den Kontextmenüpunkt QUICKFIX auf.



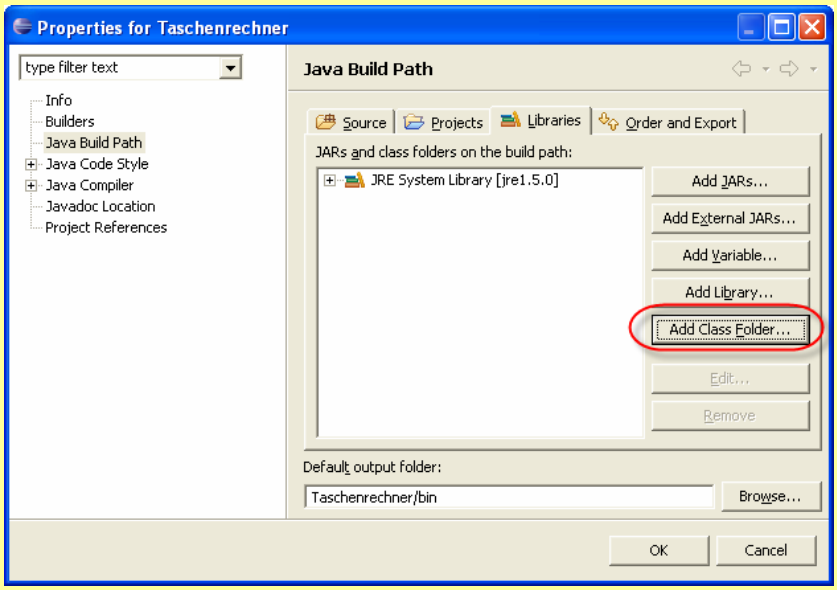
- Im Dialogfenster QUICK FIX werden drei Lösungsmöglichkeiten für das Problem vorgeschlagen. Die markierte Lösungsmöglichkeit beseitigt das Problem. Bestätigen Sie mit OK.



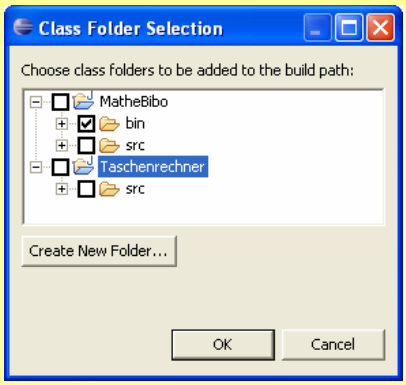
- Als Ergebnis wurde in den Projektoptionen (Menüpunkt PROJECT - PROPERTIES) im Register PROJECTS das Projekt *MatheBibo* hinzugefügt.



- Alternativ können Sie auch in das Register LIBRARIES wechseln und klicken dort auf ADD CLASS FOLDER.



- Setzen Sie einen Haken vor das *bin*-Verzeichnis des *Mathe*-Projekts und bestätigen Sie mit OK.



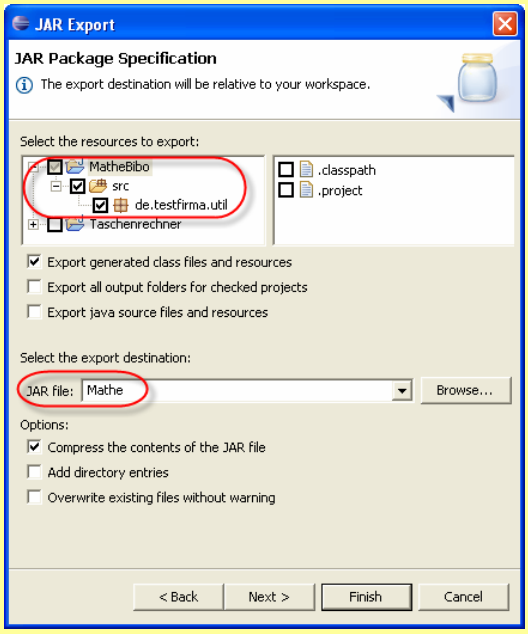
- Fügen Sie in die `main()`-Methode noch die beiden Anweisungen ein:

```
System.out.println(Mathe.Add(10, 11));
System.out.println(Mathe.Minus(11, 34));
```

4.3 JAR-File erstellen

- Rufen Sie im Package Explorer den Kontextmenüpunkt EXPORT auf.
- Wählen Sie als Exportformat JAR FILE und klicken Sie NEXT.

- Markieren Sie das *..\src*-Verzeichnis des *Mathe*-Projekts. Geben Sie in das Feld JAR FILE den Namen des neuen Archivs ein, z.B. *Mathe.jar*. Deaktivieren Sie gegebenenfalls die Markierungen vor *.classpath* und *.project*.



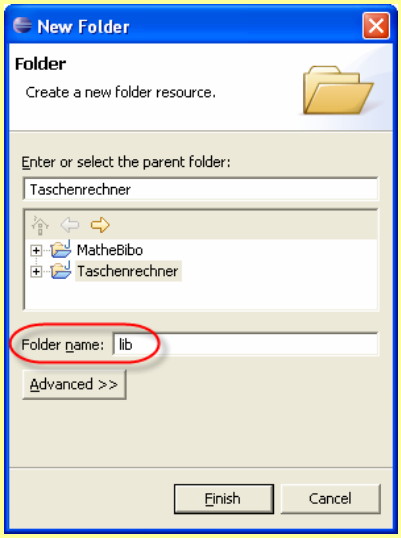
- Klicken Sie auf FINISH. Die JAR-Datei wird im Workspace-Verzeichnis erstellt.

Ein umfangreicheres Buildmanagement ist mit Ant möglich.

4.4 Verwenden von JAR-Dateien

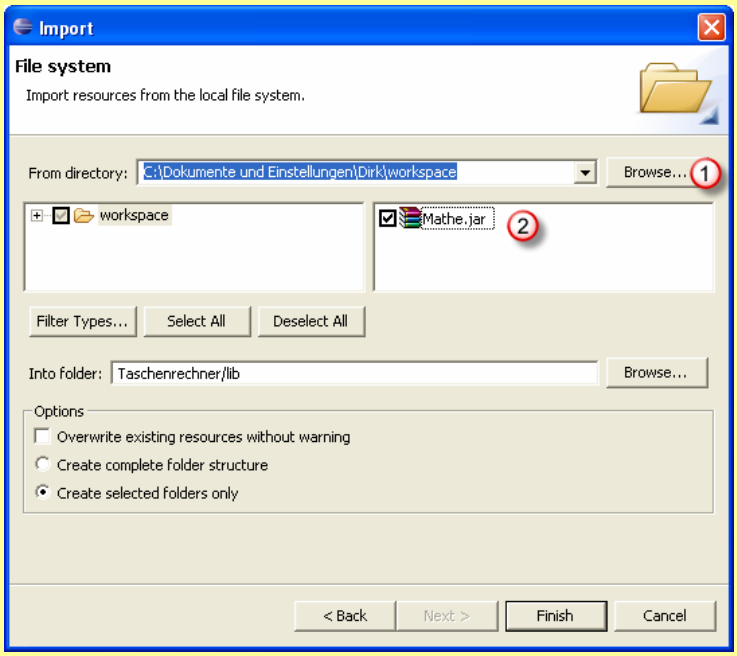
Statt nun das Verzeichnis der Class-Dateien des *Mathe*-Projekts zu referenzieren, soll das erstellte JAR-Archiv genutzt werden. Dies ist beispielsweise der Standardvorgang, wenn Sie Archive von Fremdherstellern nutzen wollen.

- Entfernen Sie aus dem *Rechner*-Projekt die Referenzen auf das *Mathe*-Projekt.
- Erstellen Sie über den Package Explorer einen neuen Ordner *..\lib* im *Rechner*-Projekt. Rufen Sie dazu den Menüpunkt NEW - FOLDER des Projekts auf.



- Rufen Sie im Package Explorer den Kontextmenüpunkt IMPORT des Ordner *lib* im Rechner-Projekts auf.
- Wählen Sie den Eintrag FILE SYSTEM und klicken Sie auf NEXT.

- Wählen Sie nach dem Klick auf BROWSE den Ordner mit dem Archiv aus (1) und markieren Sie es (2).

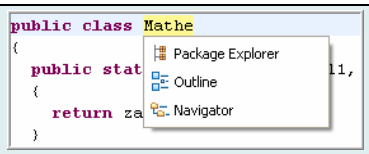
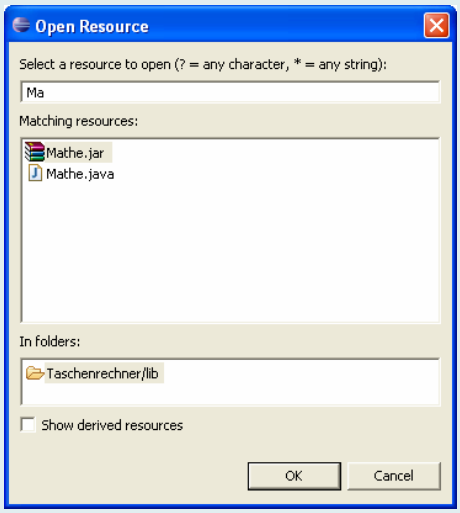


- Bestätigen Sie mit FINISH.

- Jetzt muss noch das Archiv dem Klassenpfad des Projekts hinzugefügt werden.
- Öffnen Sie die Projektoptionen über den Menüpunkt PROJECT - PROPERTIES, markieren Sie die Kategorie JAVA BUILD PATH und wechseln Sie in das Register LIBRARIES.
- Klicken Sie auf ADD JARS. Wählen Sie dann das importierte Archiv aus und bestätigen Sie mit OK.

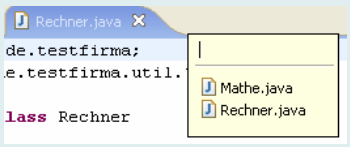
5 Tipps

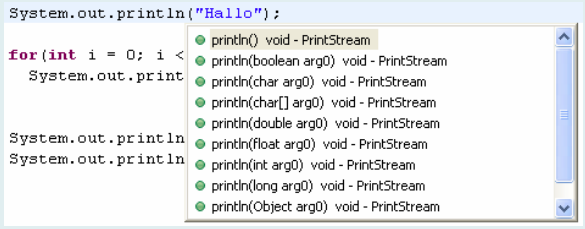
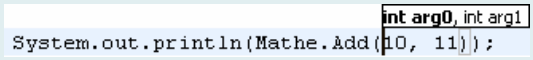
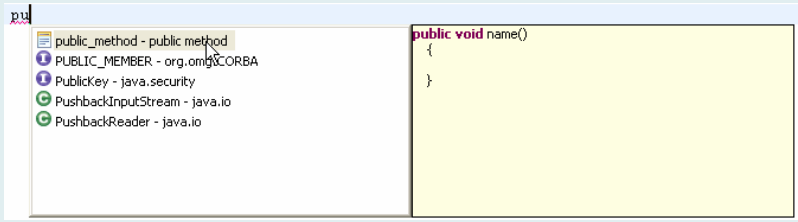
Navigationshistorie	Über ALT + ⇦ und ALT + ⇨ können Sie sich im Editor vorwärts und rückwärts entsprechend Ihrer eigenen Navigation bewegen.
Inkrementelle Suche	Über STRG + J starten Sie die inkrementelle Suche. In der Statuszeile wird Incremental Find angezeigt. Geben Sie jetzt die nacheinander die Anfangsbuchstaben des Suchbegriffes ein. Diese werden dann in der Statuszeile angezeigt und im Editor wird die erste Fundstelle markiert. Mit STRG + J bewegen Sie sich zur nächsten Fundstelle.
Synchronisation	Wenn Sie sich auf einem Begriff befinden klicken Sie UMSCHALT + ALT + W. Es wird ein Popup angezeigt, über das Sie mit dem jeweiligen View synchronisieren können.

	
Resource-Suche	Um schnell eine Ressource zu finden, betätigen Sie STRG + UMSCHALT + R oder rufen den Menüpunkt NAVIGATE - OPEN RESOURCE auf. Geben Sie nun die Anfangsbuchstaben der Ressource ein. 
Mehrere Editorfenster	Ziehen Sie das Register des Editors an eine "Kante". Dann können Sie mehrere Editorfenster nebeneinander sehen. Um es wieder anzudocken, ziehen Sie es auf den Tab eines anderen Fensters.
Alles zuklappen	Klicken Sie im Package Explorer auf  , um alle Knoten einer Ansicht zu schließen.

Javadoc-Kommentare	Geben Sie die Zeichenfolge /** ein und betätigen Sie RETURN. Es wird automatisch der Code analysiert und der Kommentar abgeschlossen, z.B. wird für die Methode `Minus(int zahl1m, int zahl2)` der folgende Kommentar erzeugt:
	`/**`
	`*`
	`* @param zahl1`
	`* @param zahl2`
	`* @return`
	`*/`

ToDo-Listen	Geben Sie in einem JavaDoc-Kommentar TODO gefolgt von einem Text ein und speichern Sie die Datei. Dieser Text wird im TASK-Fenster angezeigt (WINDOW - SHOW VIEW - OTHER - BASIC - TASKS).
	`/**`
	`* TODO Hier ist noch was zu tun...`

Editorliste	Über STRG + E wird eine Liste mit allen geöffneten Editorfenstern angezeigt. 
Syntaxhilfe im Editor	STRG + LEERTASTE betätigen

	
Inhaltshilfe	In einer Anweisung betätigen Sie STRG + LEERTASTE. Es werden die möglichen Zuweisungen angezeigt.
Parameterhilfe	STRG + UMSCHALT + LEERTASTE in der Parameterliste einer Methode betätigen, der aktuelle Parameter wird fett dargestellt. 
Codevorlagen	Geben Sie die Anfangsbuchstaben ein und betätigen sie STRG + LEERTASTE.  Die Codevorlagen werden unter WINDOW - PREFERENCES, JAVA - EDITOR - TEMPLATES definiert.
Interfaces implementieren	Rufen Sie in der Klasse den Kontextmenüpunkt SOURCE - OVERRIDE/IMPLEMENT METHODS auf.

6 Refactoring

Über Refactoring verändern Sie die Struktur Ihres Codes, ohne dessen Funktionalität zu ändern. Ziel ist es, den Code besser wartbar, lesbar und wieder verwendbarer zu gestalten. Typische Refactorings sind:

- Aussagekräftige Variablennamen
- Aufteilen langer Codepassagen in mehrere Methoden

6.1 Bezeichner umbenennen

- Begeben Sie sich zur Deklaration des Bezeichners und rufen Sie den Menüpunkt REFACTOR - RENAME auf.
- Alternativ rufen Sie den gleichnamigen Kontextmenüpunkt auf.
- Vergeben Sie im Dialogfeld einen neuen Namen. Entspricht der Name nicht den Konventionen für die Namensgebung unter Java, wird außerdem eine Meldung angezeigt.
- Durch die Markierung der Option UPDATE REFERENCES werden auch Referenzen auf den Bezeichner (Methode, Variable, Klasse etc.) umbenannt.

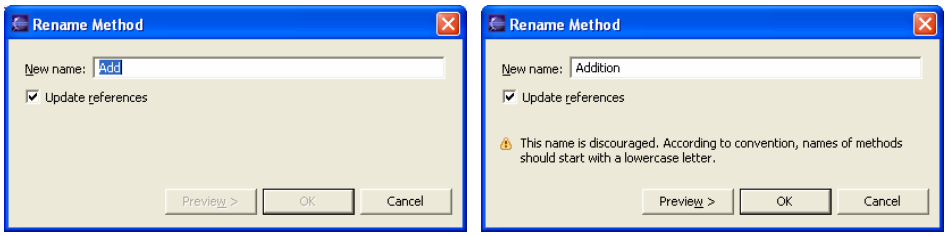
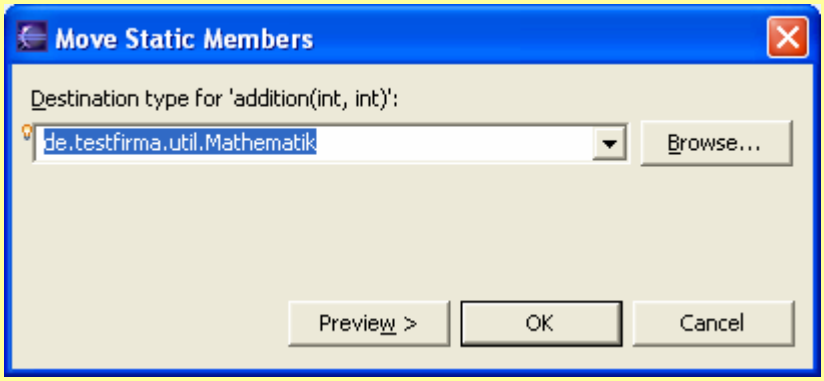


Abb. 6.10

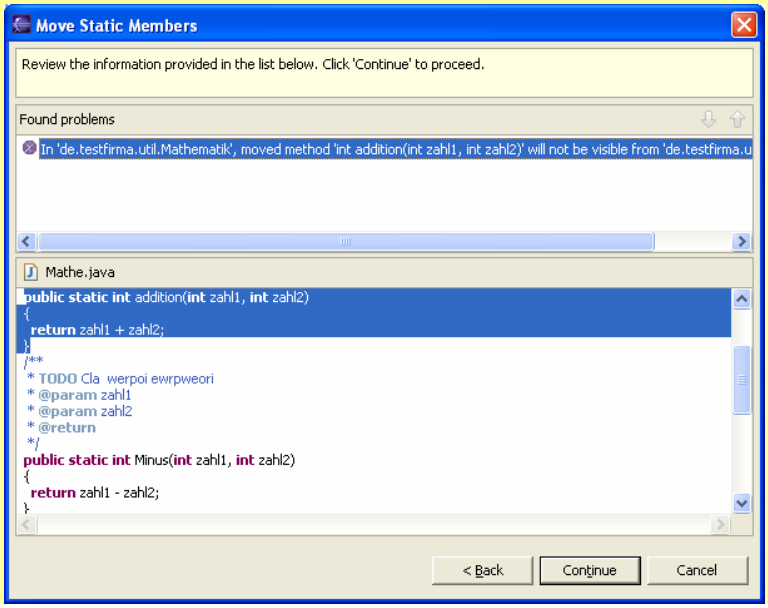
6.2 Methoden verschieben

Hiermit können Sie Methoden einer Klasse in eine andere verschieben.

- Markieren Sie die Methodendeklaration in rufen Sie den Menüpunkt REFACTOR - MOVE auf. Geben Sie den Typ (die Zielklasse) an oder wählen Sie den Typ über BROWSE aus.



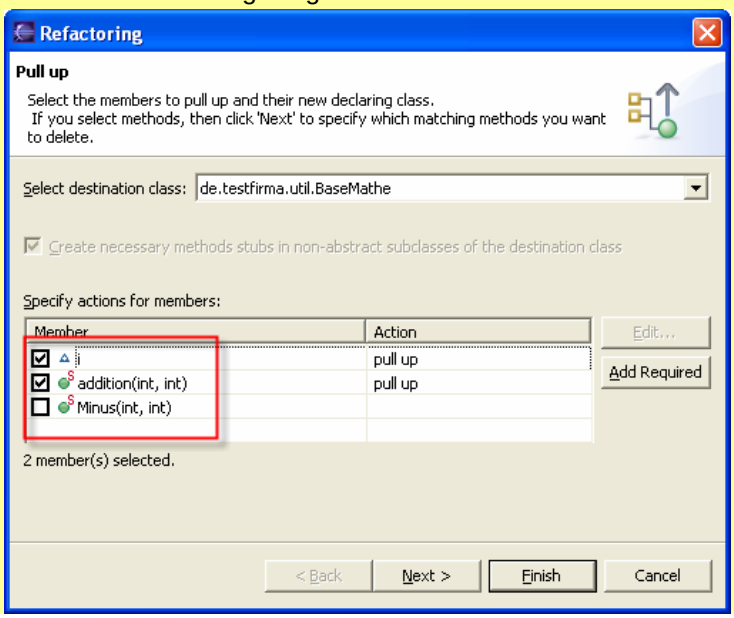
- Gibt es beim Verschieben Probleme mit existieren Code (im Beispiel ist die neue Klasse nicht für andere Klassen sichtbar, welche die Methode bereits verwenden), wird ein Dialog mit weiteren Informationen angezeigt.



6.3 Pull up - Member in Basisklassen verschieben

Mit diesem Refactoring verschieben Sie ausgewählte Member einer Klasse in eine Basisklasse.

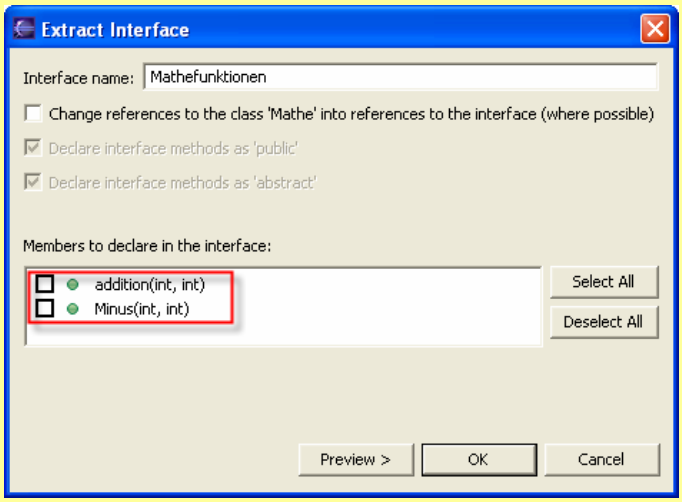
- Sie können sich dazu an einer beliebigen Stelle innerhalb der Klasse befinden und rufen den Menüpunkt REFACTOR - PULL UP auf.
- Markieren Sie im angezeigten Fenster die zu verschiebenden Member.



6.4 Schnittstellen extrahieren

Aus einer Klasse lassen sich für ausgewählte Methoden Interfaces extrahieren. Die Methoden müssen dazu `public` und nicht `static` sein.

- Rufen Sie innerhalb der Klasse den Menüpunkt REFACTOR - EXTRACT INTERFACE auf.
- Markieren Sie im Dialog die Methoden, die in das Interface aufgenommen werden sollen und vergeben Sie einen Namen für das Interface.



- Standardmäßig wird das Interface im selben Package wie die Klasse erzeugt. Außerdem wird die Klassendeklaration um `implements` erweitert.

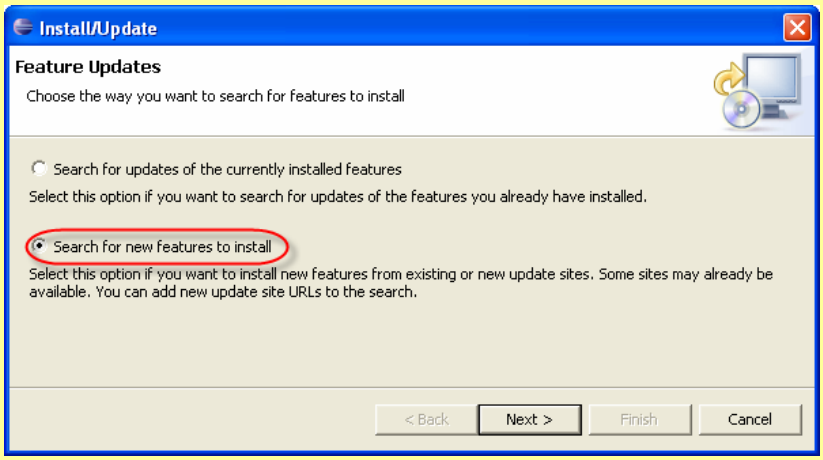
7 Neue Plug-Ins installieren

Erweiterungen von Eclipse werden durch Plug-Ins durchgeführt. Eclipse selbst stellt im Prinzip eine einzige Menge von Plug-Ins dar. Im Folgenden soll das CheckStyle-Plug-In installiert werden, über dass Sie Ihren Code auf die Einhaltung von Regeln (Schreibweise von Bezeichnern) überprüfen können.

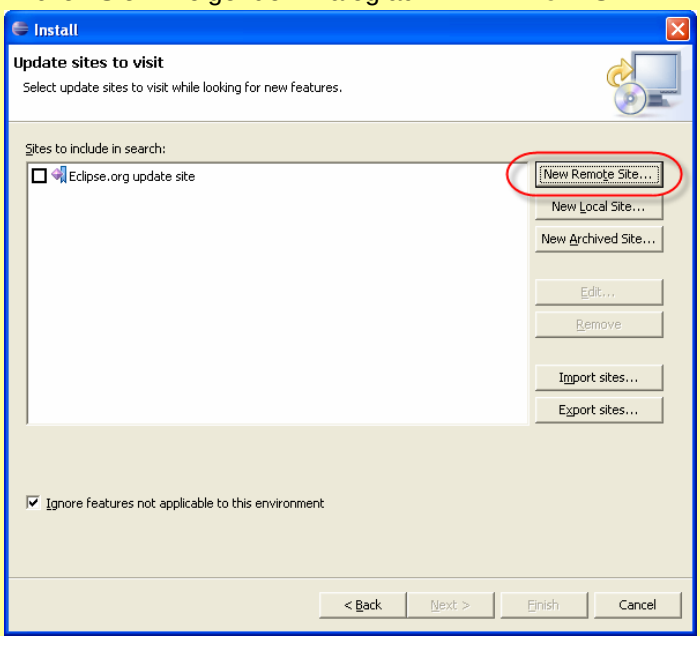
CheckStyle finden Sie unter der URL *http://eclipse-cs.sourceforge.net/*.

- Laden Sie die die aktuelle Version 4.0.0 beta 4 von *http://sourceforge.net/projects/eclipse-cs/*.
- Beenden Sie Eclipse.
- Extrahieren Sie die Datei *com.atlassw.tools.eclipse.checkstyle_4.0.0b4b-bin.zip* in das Eclipse-Installationsverzeichnis. Es wird ein neuer Ordner *com.atlassw.tools.eclipse.checkstyle_4.0.0* unter *..\plugins* erzeugt.
- Starten Sie Eclipse neu. Verwenden Sie dazu die Option `-clean`, also: `eclipse -clean`. Ansonsten kann es sein, dass CheckStyle nicht erkannt wird und damit nicht zur Verfügung steht.

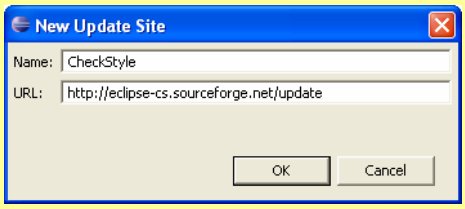
- Alternativ installieren Sie CheckStyle über den Menüpunkt HELP - SOFTWARE UPDATES - FIND AND INSTALL. Markieren Sie die Option SEARCH FOR NEW FEATURES TO INSTALL und klicken Sie auf NEXT.



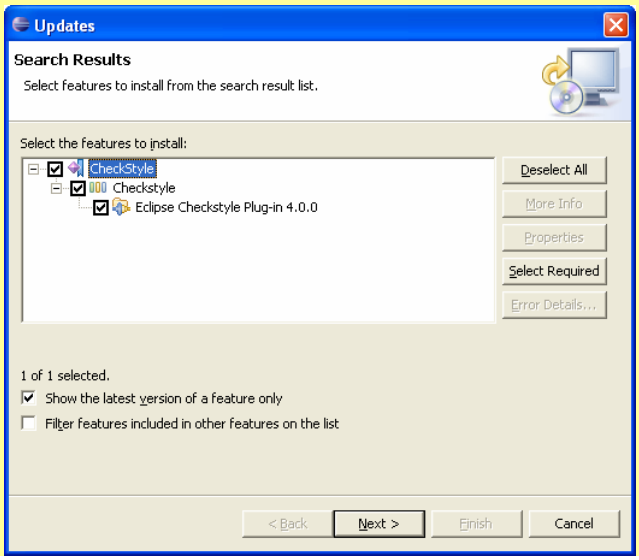
- Klicken Sie im folgenden Dialog auf NEW REMOTE SITE.



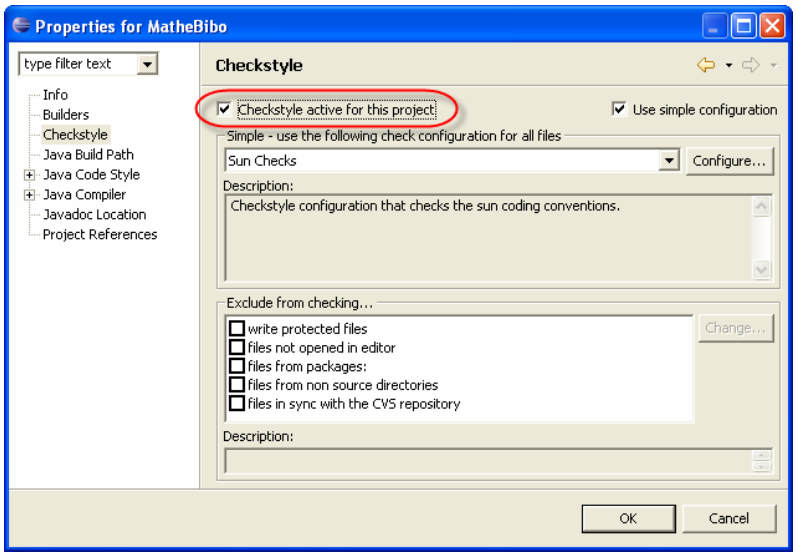
- Geben Sie einen Namen für das neue Plug-In und die URL ein. Klicken Sie auf OK und danach auf FINISH.



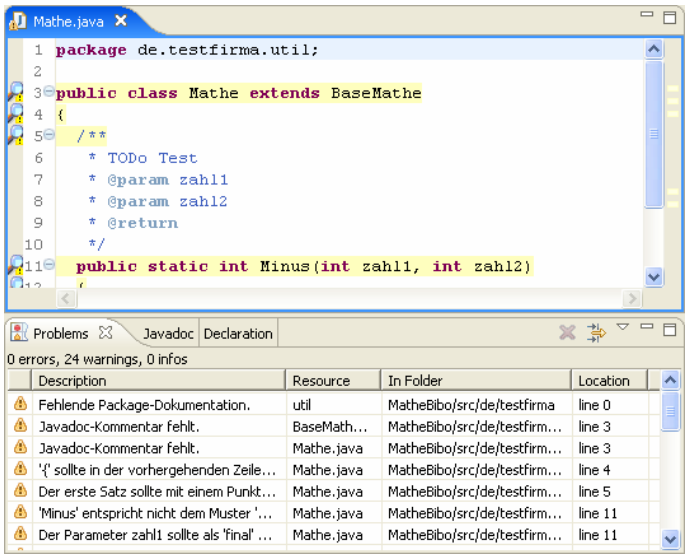
- Der Update-Manager wird gestartet. Markieren Sie im Ergebnis das CheckStyle-Plug-In. Akzeptieren Sie die Lizenzbedingungen und klicken Sie am Ende FINISH. Die Installation wird durchgeführt.



Unter dem Menüpunkt WINDOWS - PREFERENCES gibt es nun eine neue Kategorie CHECKSTYLE über die Sie eigene, angepasste Check-Konfigurationen erstellen können. Für ein Projekt aktivieren Sie CheckStyle in den Projekteigenschaften (Kontextmenüpunkt PREFERENCES eines Projekts). Markieren Sie dazu die Option CHECKSTYLE ACTIVE FOR THIS PROJECT.



Der Quellcode wird nun bei jedem Speichervorgang geprüft und Verstöße gegen die CheckStyle-Richtlinien werden im Problems-Fenster angezeigt.



8 SWT - Standard Widget Toolkit

Eclipse besitzt in der Standardinstallation keinen grafischen Editor. Es stehen aber mehrere Editoren als Plug-In zur Verfügung, von denen nun der VE (Visual Editor) verwendet wird. Unter der URL *http://www.eclipse.org/downloads/index.php* klicken Sie zum Download auf den Link des Editors.

Vorher sind noch das EMF Build 2.1.0 und GEF Build 3.1 zu installieren (zusammen ca. 4MB). Dazu extrahieren Sie beide ZIP-Dateien *GEF-runtime-3.1.zip* und *emf-sdo-runtime-2.1.0.zip* unterhalb des Verzeichnisses ..*eclipse*. Danach sollte Eclipse auf der Kommandozeile mit der Option `-clean` aufgerufen werden:

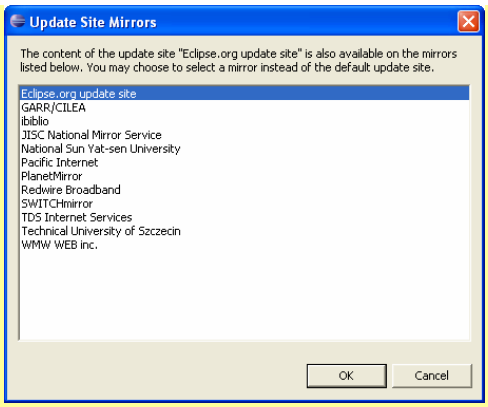
```
eclipse -clean
```

EMF - Eclipse Modeling Framework
GEF - Graphical Modeling Framework

Jetzt können Sie die Datei *VE-runtime-1.1.zip* entpacken und starten Eclipse noch einmal mit der Option `-clean`.

Alternativ verwenden Sie wieder das Update-Feature. In diesem Fall werden alle Einstellungen automatisch vorgenommen.

- Klicken Sie auf den Menüpunkt HELP - SOFTWARE UPDATES - FIND AND INSTALL.
- Wählen Sie dann SEARCH FOR NEW FEATURES TO INSTALL.
- Markieren Sie ECLIPSE.ORG UPDATE SITE.



Markieren Sie diese Option im folgenden Fenster noch mal.

- Markieren Sie EMF, GEF und VE und los gehts.

Einstellungen zum VE nehmen Sie unter WINDOWS - PREFERENCES, Kategorie JAVA - VISUAL EDITOR vor.

8.1 Vergleich AWT, SWT, Swing

SWT	- portable Bibliothek - es wird eine Implementierung für jedes Betriebssystem benötigt - evt. Umsetzungsprobleme, wenn die Funktionsweise unter verschiedenen Systemen anders ist - keine Standardbibliothek des JDK - Zugriff auf Betriebssystem über C-Routinen und das JNI - Look&Feel wie ein natives Programm - schnell
AWT	- auf Standardkomponenten aller Betriebssysteme beschränkt - kaum komplexe Elemente wie Trees - schnell - läuft überall - Standardbibliothek seit der ersten Version des JDK
Swing	- werden unabhängig vom Betriebsystem gerendert - etwas langsamer als AWT - läuft prinzipiell überall, evt. mit anderem Erscheinungsbild - Standardbibliothek seit dem JDK 1.2

8.1.1 Klassenübersicht des SWT

Shell	Fenster
Display	GUI - Prozess (Verbindung zwischen Anwendung und Betriebssystem)
Widget	Basisklasse aller Kontrollelemente
Control	davon abgeleitete Klasse für die GUI-Elemente
Composite	Container für andere Controls

8.1.2 Neue Fenster

Unter WINDOWS - SHOW VIEW - OTHER finden Sie in den Katagorien BASIC und JAVA neue Einträge.

- Palette - Komponenten
- Properties - Eigenschaften der Komponenten
- Java Beans - Hierarchie der Komponenten
- Editorfenster mit 2 Sichten (Design und Source)

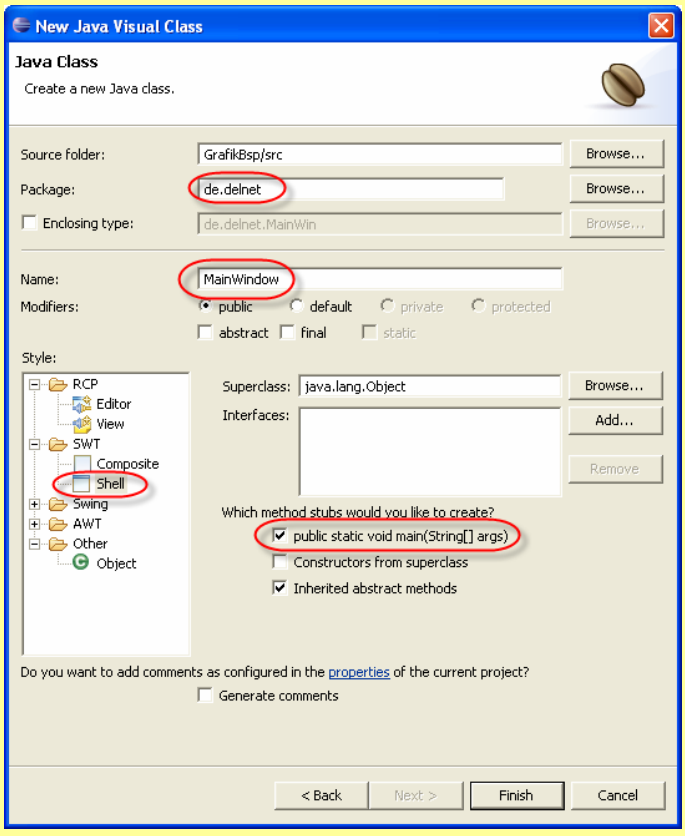
8.2 SWT-Rahmen erstellen

Projekt erstellen

- Erstellen Sie über FILE - NEW - PROJECT ein neues JAVA PROJEKT. Betätigen Sie nach Angabe des Projektnamens NEXT.
- Wechseln Sie in den JAVA SETTINGS in das Register LIBRARIES und klicken Sie auf ADD LIBRARY.
- Wählen Sie den Eintrag STANDARD WIDGET TOOLKIT (SWT) und bestätigen Sie mit NEXT.
- Wählen Sie als Plattform die Plattform Ihrer laufenden IDE und klicken Sie zweimal FINISH.

Klasse erstellen

- Klicken Sie nun auf den Menüpunkt FILE - NEW - OTHER.
- Markieren Sie unter der Kategorie JAVA den Eintrag VISUAL CLASS.



- Markieren Sie SWT - SHELL und vergeben Sie einen Klassen- und Packagenamen. Klicken Sie dann auf FINISH.

Beispielanwendung:

```
import org.eclipse.swt.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
```

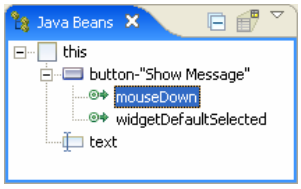
```
import org.eclipse.swt.widgets.*;

public class MainWin extends Composite
{
  private Button button = null;
  private Text text = null;
  public static void main(String[] args)
  {
    // GUI-Prozess starten
    Display display = Display.getDefault();
    // shell als Hauptfenster festlegen - Übergabe von display
    Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(new Point(300, 200));
    MainWin thisClass = new MainWin(shell, SWT.NONE);
    shell.open();
    // Nachrichtenschleife initialisieren
    // wird beendet, wenn das Hauptfenster geschlossen wird
    while(!shell.isDisposed())
    {
      if(!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }
  public MainWin(Composite parent, int style)
  {
    super(parent, style);
    initialize();
  }
  // der Name der Initialisierungsmethode kann kanfiguriert werden
  private void initialize()
  {
    button = new Button(this, SWT.NONE);
    button.setText("Show Message");
    button.setAlignment(org.eclipse.swt.SWT.LEFT);
    text = new Text(this, SWT.BORDER);
    this.setLayout(new RowLayout());
    setSize(new org.eclipse.swt.graphics.Point(476,325));
  }
}
```

8.2.1 Layoutmanager

- FillLayout - gleichmäßiges Auffüllen (hintereinander, z.B. in einer Toolbar)
- RowLayout - ähnlich FillLayout, Abstände und Ausrichtung einstellbar
- FormLayout - relative Abstände zwischen Elementen einstellbar
- GridLayout - Tabelle mit zahlreichen Einstellmöglichkeiten

8.3 Ereignisbehandlung

Ereignisse können über den Kontextmenüpunkt EVENTS einer Komponente oder im Kontextmenü einer Komponente in der JavaBeans-Sicht behandelt werden.



EventInterface	Methoden - EventTyp, Adapter
KeyListener	keyPressed, keyReleased - KeyEvent, KeyAdapter
MouseListener	mouseDown, mouseUp, mouseDoubleClick - MouseEvent, MouseAdapter
MouseMotionListener	mouseMove - MouseEvent
MouseTrackListener	mouseEnter, mouseExit, mouseHover - MouseEvent,
ModifyListener	modifyText - ModifyEvent
VerifyListener	verifyText - VerifyEvent
SelectionListener	widgetSelected - SelectionEvent

8.3.1 Generischer Listener

Mit generischen Listenern können Sie verschiedene Ereignisse in einer Methode über `switch-case`-Anweisungen verarbeiten.

```
Listener l = new Listener()
  {
    public void handleEvent(Event e)
    {
      switch(e.type)
      {
        case SWT.Paint: ... break;
        ...
      }
    }
  }
shell.addListener(SWT.Paint, l);
```

8.4 Weitergabe und Besonderheiten

Zur Weitergabe einer SWT-Anwendung unter verschiedenen Betriebssystemen muss die Anwendung selbst nicht verändert bzw. angepasst werden. Lediglich das verwendete SWT-Archiv muss ausgetauscht werden.

Es ist keine Garbage Collection für SWT-Komponenten möglich, da die Bibliothek sehr systemnah ist. Inhalte eines Fensters werden aber Komponenten durch die Parent-Beziehung automatisch freigegeben.

Zur Ausführung von SWT-Anwendungen müssen Sie die notwendigen Archive unter *http://www.eclipse.org/downloads/index.php* laden und die Datei *swt.jar* zum Klassenpfad hinzufügen. Weiterhin muss das Verzeichnis zur SWT-Dll über die Systemeigenschaft `java.library.path` angegeben werden.

```
java -classpath swt.jar;TestApp.jar -Djava.library.path=C:\Lib\swt MainApp
```

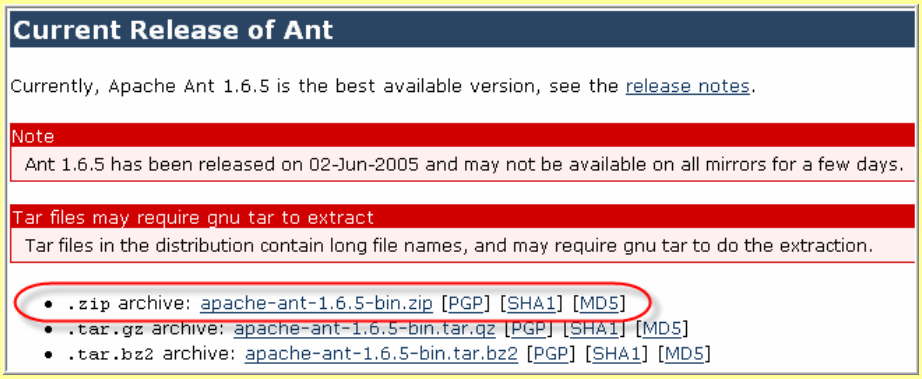
9 Build-Management mit Ant

Mit dem Übersetzen und Ausführen einer Java-Anwendung ist es meist nicht getan. Dateien müssen gesichert werden, z.B. in einer Versionsverwaltung. Für die Weitergabe müssen zusätzliche Dateien und Bibliotheken zusammengestellt werden usw. Das Tool Ant (Another Neat Tool) hilft Ihnen dabei, diesen Prozess zu automatisieren.

Zur Ausführung von Ant verfügen Sie idealerweise über ein JDK 1.4.2 oder neuer. Ansonsten wird noch ein XML-Parser benötigt. Das JRE genügt nicht, da z.B. kein Compiler enthalten ist.

Ant finden Sie unter der URL *http://ant.apache.org/*.

- Laden Sie die die aktuelle Version 1.6.5 von *http://ant.apache.org/bindownload.cgi*.
- Verwenden Sie die ZIP-Version. Diese ist ca. 9.6MB groß.



- Entpacken Sie die Datei in einem beliebigen Ordner. Es entsteht das Verzeichnis *..\apache-ant-1.6.5*.
- Namen Sie das *..\bin*-Verzeichnis der Ant-Installation in den Klassenpfad auf. Dadurch kann Ant jederzeit aufgerufen werden.
- Setzen Sie optional die Umgebungsvariable `ANT_HOME` auf das Installationsverzeichnis von Ant. Dies ist aber eigentlich nicht notwendig, außer die Skripte arbeiten nicht korrekt.
- Setzen Sie außerdem - falls noch nicht geschehen - die Umgebungsvariable `JAVA_HOME` auf das Installationsverzeichnis Ihres JDKs. Dies ist z.B. für die Ausführung von Ant unter Eclipse notwendig.

Das besondere an Ant ist, das es Java-basierend ist und XML-Dateien als Eingabe entgegennimmt. Java-basierend heißt, dass Ant selbst mit Java geschrieben wurde und auch darüber erweitert wird. Dies macht das Erlernen einer weiteren Skriptsprache unnötig. Außerdem ist Ant dadurch weites gehend Plattformunabhängig einzusetzen (es werden zahlreiche Einstellungen für eine verbesserte Plattformunabhängigkeit bereitgestellt). Um Ant zu verwenden, gehen Sie folgendermaßen vor:

- Installieren Sie Ant
- Erstellen Sie den Source für Ihr Java-Projekt (auch andere Projektarten wären möglich).
- Legen Sie eine geeignete Verzeichnisstruktur an.
- Erstellen Sie eine Konfigurationsdatei *build.xml*.
- Erstellen Sie optionale Erweiterungen für Ant.
- Rufen Sie Ant auf.

9.1 Beispielanwendung

Als Beispielanwendung werden zwei Dateien in der folgenden Dateistruktur verwendet. Wurzel des Klassenpfades ist *..\src*. Im Verzeichnis *..\classes* werden die erzeugten CLASS-Dateien erstellt.

..\src\de\delnet\rechner\Rechner.java
..\src\de\delnet\util\Mathe.java
..\classes

Übersetzt werden die Sourcen, ausgehend vom *..\src* und *..\classes* übergeordneten Verzeichnis durch:

```
> javac -d classes -cp .\src src\de\delnet\rechner\*.java
```

Rechner.java:

```
package de.delnet.rechner;
import de.delnet.util.*;
public class Rechner
{
  public Rechner()
  {
    System.out.println(Mathe.add(10, 11));
  }
  public static void main(String args[])
  {
    new Rechner();
  }
}
```

Mathe.java:

```
package de.delnet.util;
public class Mathe
{
  public static int add(int zahl1, int zahl2)
  {
    return zahl1 + zahl2;
  }
}
```

9.2 Aufruf von Ant

Ant wird über Batch-Dateien oder Skripte gestartet, die sich im *..\bin*-Verzeichnis der Installation befinden. Wird Ant ohne Angabe von Parametern gestartet, wird im aktuellen Verzeichnis nach einer Datei *build.xml* gesucht und diese als Konfiguration für den Buildprozess genutzt.

```
ant [Optionen] [Target1] [Target2] ...
```

Über Targets können bestimmte Buildziele angegeben werden. Diese Ziele werden in der Konfigurationsdatei definiert.

Optionen	Beschreibung

`-version`	gibt die Ant-Version aus
`-quiet`	wenig Ausgaben anzeigen
`-verbose`	viele Informationen ausgeben
`-lib`	Pfad zu Verzeichnissen die Archive und CLASS-Dateien beinhalten
`-l [Datei]`	Log-Datei angeben
`-f [Datei]`	Build-Konfigurationsdatei angeben
`-DName=Wert`	Übergeben Sie Ant Name/Wert-Paare von Eigenschaften. Diese können in Ant ausgewertet werden bzw. die bereits definierten Werte überschreiben.
`-propertyfile [Datei]`	Eigenschafswerte über eine Datei übergeben

Standardmäßig wird nach Archiven im *..\lib*-Verzeichnis von Ant gesucht.

```
ant -f MyBuild.xml -lib D:\myLibs
ant -DZielVer=D:\Ziel
```

9.3 Aufbau der XML-Konfigurationsdatei

- Die Angabe der XML-Version leitet die XML-Datei ein.
- Der Inhalt wird in ein `<project>`-Element eingeschlossen.
- Über `<property>`-Elemente können Eigenschaften definiert werden, auf die später über `${Name}` zugegriffen werden kann.
- Über `<target>`-Elemente werden bestimmten Ziele definiert, die z.B. den Kompiliervorgang oder das Erstellen der Dokumentation beinhalten. Ziele können voneinander abhängig sein.
- Innerhalb von Targets kommen so genannte Tasks zum Einsatz. Tasks führen die eigentlichen Operationen durch, z.B. das Kompilieren oder das Erstellen/Löschen eines Verzeichnisses.

Da die Targets benannt sind, kann Ant mit der Angabe eines oder mehrerer Targets (wie bereits erwähnt) gestartet werden. Es wird dann dieses Target ausgeführt. Ist es von anderen abhängig, werden diese vorher ausgeführt.

```
<?xml version="1.0"?>
<project name="Ant-Test" default="Main" basedir=".">
  <property name="src" location="src" />
  <property name="build" location="build" />
  <property name="dist" location="dist" />
  <property name="docs" location="docs" />

  <target name="clean">
    <delete dir="${build}" />
    <delete dir="${docs}" />
    <delete dir="${dist}" />
  </target>

  <target name="compile" depends="clean">
    <mkdir dir="${build}" />
    <javac srcdir="${src}" destdir="${build}" />
  </target>
```

```
  <target name="docs" depends="compile">
    <mkdir dir="${docs}" />
    <javadoc packagenames="de.delnet.*" sourcepath="${src}" destdir="${docs}" />
  </target>

  <target name="jar" depends="compile">
    <mkdir dir="${dist}" />
    <jar destfile="${dist}\MainApp.jar" basedir="${build}" >
      <manifest>
        <attribute name="Main-Class" value="de.delnet.rechner.Rechner" />
      </manifest>
    </jar>
  </target>

  <target name="Main" depends="jar, docs">
    <description>Das Hauptziel...</description>
  </target>
</project>
```

Da es sich um eine XML-Datei handelt, können über die Zeichenfolgen `<!--` und `-->` Kommentare eingefügt werden.

Neben den hier beschriebenen Dingen gibt es in Ant u.a. die Möglichkeit

- Targets aufgrund von Bedingungen auszuführen
- Zugriff auf vordefinierte Eigenschaften
- einbinden anderer Konfigurationsdateien

9.3.1 Verzeichnisstruktur eines Ant-Projekts

Die Verwaltung der Dateien in geeigneten Verzeichnissen erleichtert die Erstellung von Datensicherungen und die Weitergabe. Folgender Struktur wird häufig in Projekten verwendet. Die Verzeichnisse *..\build* und *..\dist* und evt. auch *..\doc* können jederzeit wiederhergestellt werden und müssen deshalb nicht gesichert werden.

Verzeichnis	Inhalt
bin	Binäre Zusatzdateien wie Dlls, Skripte o.a.
build	Enthält die übersetzten CLASS-Dateien.
dist	Enthält die Dateien, die für eine Weitergabe des Projekts notwendig sind. Dies sind unter Umständen nicht nur die CLASS-Dateien.
doc	Hier wird die JavaDoc- und sonstige Dokumentation abgelegt.
lib	Enthält alle benötigten Archive
src	Enthält die Java-Sourcen

Sichern sie neben den Sourcen auch sonstige Anwendungen, die zum Erstellen der Projektdateien notwendig sind, z.B. das verwendete JDK, die Ant-Version etc.

9.3.2 <project>-Element

Dieses Element schließt alle anderen Elemente ein.

Attribut	Beschreibung

basedir	Basisverzeichnis auf das sich alle relativen Verzeichnisangaben beziehen, sonst ist es das Verzeichnis der Build-Datei
default	Definiert das Standardziel, wird es nicht angegeben (seit Version 1.6.0 möglich), werden alle TopLevel-Targets ausgeführt.
name	Beschreibung zu Auswertungszwecken

9.3.3 <target>-Element

Jedes Projekt besteht aus einer beliebigen Anzahl von Targets. Durch die Angabe eines Targets beim Aufruf von Ant, kann jedes beliebige Target ausgeführt werden. Über das folgende Kommando können Sie sich alle Targets einer Konfiguration anzeigen lassen.

```
ant -projecthelp
```

Attribut	Beschreibung
name	Damit ein Ziel verwendet werden kann, muss ihm über dieses Pflichtattribut ein Name gegeben werden.
depends	Hier geben Sie die Namen von Targets an, die vorher abgearbeitet werden müssen. Mehrere Targets werden durch Komma getrennt.
description	Hat ein Target dieses Attribut, wird es als Main Target identifiziert, z.B. beim Aufruf `ant -projecthelp`

Typische Targets (Ziele) sind eines Ant-Projekts sind:

Name	Beschreibung
all	Es werden alle Ziele ausgeführt.
build	Es wird eine Kompilierung durchgeführt.
clean	Die Ausgabeverzeichnisse (*..\build* und *..\dist*) werden geleert.
deploy	Erstellt ein Weitergabeprojekt und führt sie evt. durch (auf einen Server o.a.)
docs	Erstellt die Dokumentation.
fetch	Holt die aktuellen Dateien aus der SourceCode-Verwaltung.
init	Initialisierung, z.B. Variablen setzen
jar	Archive erstellen
main	Hauptziel - z.B. Ausführung von clean, build, docs, fetch
test	Ausführung von Unit-Tests

9.3.4 Tasks

Ein Target besteht aus mehreren Tasks. Da Ant sehr viele Tasks bereitstellt, soll im Folgenden nur eine kurze Auswahl gezeigt werden. Sie finden die Dokumentation unter *..\docs\manual\index.html* und dem Link ANT TASKS auf der linken Seite.

Tasks	Beschreibung
javac	Geben Sie über die Attribute `srcdir` das Basisverzeichnis und mit `destdir` das Ziel für die CLASS-Dateien an.
mkdir	Im Attribut `dir` wird das zu erstellende Verzeichnis angegeben.

javadoc	Es wird die Dokumentation der angegebenen Packages (mit SubPackages) der Dateien im Quellpfad `sourcepath` erstellt und im Verzeichnis `destdir` abgelegt.
delete	Hiermit können Sie Dateien oder Verzeichnisse löschen.
jar	JAR-Archive lassen sich mit dieser Task erstellen.

9.3.5 Dateien und Verzeichnisse selektieren

Bei der Ausführung vieler Tasks ist die Angabe von Verzeichnissen oder Dateilisten notwendig. Dazu stehen verschiedene Wildcardzeichen zur Verfügung.

?	ein beliebiges Zeichen
*	0 bis n Zeichen
**	0 bis n Verzeichnisse oder Dateien, z.B. \src** - alle Dateien in allen Unterverzeichnissen

Die Festlegung der auszuwählenden Dateien erfolgt über das Element `<fileset>`. Über die Unterelemente `<include>` und `<exclude>` können die aufzunehmenden und die nicht zu verwendeten Dateien angegeben werden.

```
<fileset dir="${src}">
  <include name="**\*.java" />
  <exclude name="**\Test*.java />
</fileset>
```

Weiterhin gibt es noch `<filelist>`-, `<dirset>`- und `<pathelement>`-Elemente zur Festlegung von Verzeichnispfaden und Dateinamen.

9.4 Eigene Tasks entwickeln

- Erstellen Sie eine Klasse, die von `org.apache.tools.ant.Task` erweitert wird.
- Für eigene Attribute fügen Sie get- und set-Methoden ein (wie bei einer JavaBean).
- Stellen Sie eine Methode mit der folgenden Signatur bereit. Diese wird beim Aufruf der Task ausgeführt.
  `public void execute() throws BuildException`
- Über die Methoden der Klasse `Task` haben Sie u.a. Zugriff auf das Projekt und darüber auf alle anderen Informationen.

Für spezielle Tasks, die z.B. auch verschachtelte Elemente besitzen können, sind weitere Schritte notwendig. Diese entnehmen Sie der Dokumentation zu Ant.

```
package de.delnet.anttask;
import org.apache.tools.ant.*;
public class MyAntTask extends Task
{
  String attribut1 = "";
  public void setAttribut1(String value)
  {
    attribut1 = value;
  }
  public String getAttribut1()
```

```
  {
    return attribut1;
  }
  public void execute() throws BuildException
  {
    System.out.println(attribut1);
    System.out.println(getProject().getDefaultTarget());
  }
}
```

Übersetzen und Archiv erzeugen:

```
> javac -classpath .;d:\apache-ant-1.6.5\lib\ant.jar de\delnet\anttask\*.java
> jar cvf MyAntTask.jar de\delnet\anttask\*.class
```

- Neues Archiv nach *..\lib* im Ant-Installationsverzeichnis kopieren.
- Neuen Task bekannt machen, dazu folgenden Eintrag in die *build.xml* aufnehmen:
  ```
  <taskdef name="MyAntTask" classname="de.delnet.anttask.MyAntTask" />
  ```
- Und die neue Task verwenden
  ```
  <target name="BuildAll" depends="compile">
    <MyAntTask attribut1="Hallo" />
  </target>
  ```

9.5 Verwendung in Eclipse

- Klicken Sie auf den Menüpunkt RUN - EXTERNAL TOOLS - EXTERNAL TOOLS.
- Markieren Sie ANT BUILD und klicken Sie auf NEW.
- Wählen Sie die Build-Datei und das Basisverzeichnis aus.

ODER

- Rufen Sie in einem Projekt den Menüpunkt PROJECT - PROPERTIES auf.
- Markieren Sie die Kategorie BUILDERS.
- Wählen Sie Ant und bestätigen Sie mit OK.
- Legen Sie die *Build.xml*-Datei fest.
- Legen Sie in den Projekteigenschaften, Katagorie JAVA BUILD PATH als Ziel das korrekte Verzeichnis, z.B. *../classes* fest. Die Zielverzeichnisse müssen existieren.

INF: Für die Verwendung in Eclipse muss die Variable JAVA_HOME auf die gleiche Java-Version zeigen, die auch für Eclipse verwendet wird. Ansonsten kann es Probleme mit Ant geben.

INF: Gegebenenfalls wird beim Build keine Aktion durchgeführt, wenn sich die Dateien nicht geändert haben (auch wenn der Build vorher nicht erfolgreich war).

Index