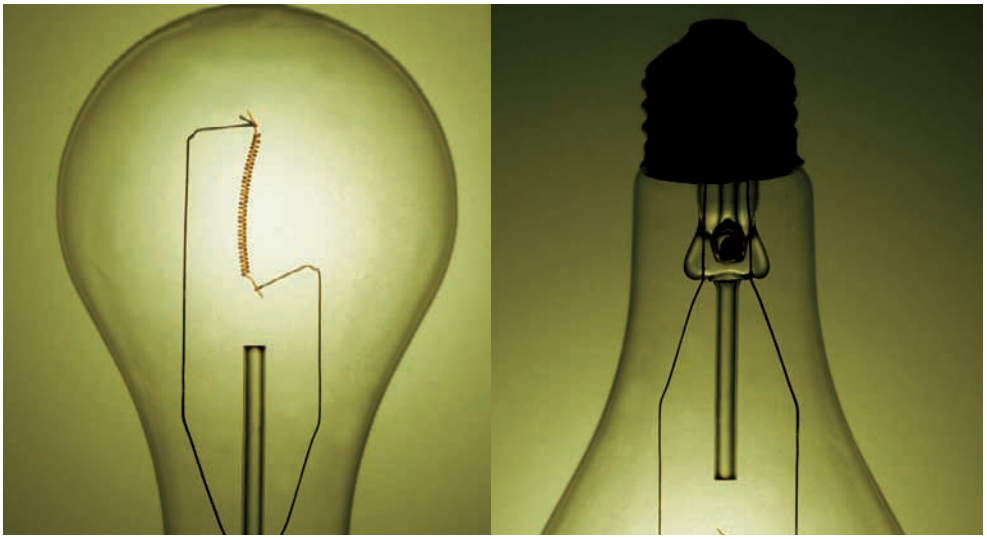
Von der initialen Projektstruktur bis zur Tool-Integration: ein Java-Starter-Cockpit mit Eclipse

Die IDE(e) macht's!

VON SABINE WINKLER

Spätestens mit dem Ziel, nach den ersten Java-Programmen in die Java-Webentwicklung einzusteigen, bemerkt man recht schnell, dass neben Java-Quelldateien weitere Ressourcen eingebunden werden müssen, die nach einer klaren, wieder verwendbaren Struktur verlangen. Ebenso wünscht man sich Entwicklungsumgebungen, die z.B. eine Integration mit dem Web-Container oder der Datenbank ermöglichen und das Deployment unterstützen. Hier erstellen wir ein „Java-Starter-Cockpit" mit Eclipse, welches diese Wünsche vereint.



Viele integrierte Entwicklungsumgebungen (IDEs) bieten für Java-Webanwendungen bereits vordefinierte Projekt-Templates, um im Rahmen der IDE automatisiert arbeiten zu können. Für den Einsteiger bleibt aus Gründen der Komplexität verborgen, was die IDE im Hintergrund anlegt, verarbeitet usw. Problematisch wird es, wenn man das Entwicklungswerkzeug wechselt und erst jetzt die Verzahnung zwischen Werkzeug und Projektaufbau erkennt. Eine Teilung der Verantwortungsbereiche zwischen Entwickler und Entwicklungswerkzeug, die mehr Unabhängigkeit bietet, ist jederzeit möglich, beispielsweise indem man komfortable Editoren, Generatoren sowie weitere Features einer IDE nutzt, sich jedoch seine grundlegende Projektstruktur unabhängig von dieser definiert.

Vom Aufbau einer unabhängigen Projektstruktur bis hin zum Arbeiten mit dieser innerhalb der IDE wird gezeigt, welche Schritte zum erfolgreichen Java-Webprojekt notwendig sind. Als Entwicklungswerkzeug kommt in diesem Artikel Eclipse [1] zum Einsatz.

„Was, wo, warum?" – Definition einer eigenen Projektstruktur

Webanwendungen mit Java zu realisieren stellt den beginnenden Java-Entwickler gleich vor mehrere Aufgaben. Neben Java-Quelldateien, die beispielsweise die Logik der Anwendung enthalten, werden Java-Webtechnologien in Form von Java-Servlets [3] und JavaServer Pages (JSP) [4], statischen HTML-Ressourcen und XML-Dateien benötigt. Kommt eine Datenbank zum Einsatz, finden sich weitere Dateien im Projekt, wie z.B. der Datenbank-Treiber, Klassen zum Zugriff auf die Datenbank und Dateien zur Konfiguration. Wie ist eine entsprechende Projektstruktur aufzubauen? Es gilt nicht nur, eine klare Organisation zu finden, die idealerweise für zukünftige Webanwendungen wieder einsetzbar ist. Ebenso sollte die Projektstruktur auf die Aktivitäten beim Entwickeln abgestimmt sein – z.B. Erzeugen der benötigten Archive oder ein automatisches Deployment in den Server.

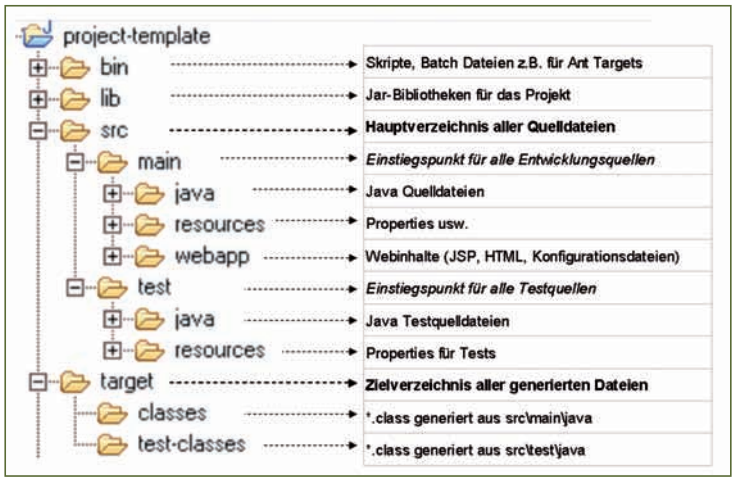


Abb. 1: Detaillierte Struktur des Projekt-Templates

Grundlegend sollten Quelldateien von generierten Inhalten getrennt werden. Ebenso ist es sinnvoll, eingesetzte Bibliotheken in Form von JAR-Dateien mit im Projekt zu verwalten, diese ebenfalls in einem eigenen Verzeichnis. Dies hat im Vergleich zu einer globalen Ablage von Bibliotheken den Vorteil, *alle* Bestandteile innerhalb des Projektes gelagert zu haben und somit eine Portierung einfach durchführen zu können. Nachteilig ist das eventuell redundante Vorhalten von Bibliotheken über mehrere Projekte. Fasst man an dieser Stelle zusammen, kann die Hauptstruktur des Projektes wie folgt definiert werden:

- ein Quellverzeichnis *src*, in dem sich Java-, JSP-, HTML-, XML- und Properties-Dateien befinden
- ein Zielverzeichnis *target*, in das Class-Dateien und Archive generiert werden
- ein Bibliotheksverzeichnis *lib*, welches alle benötigten JAR-Dateien zum Projekt enthält
- *Optional* kann ein Ausführungsverzeichnis *bin* beim Einsatz von Build-Management-Werkzeugen wie Ant mitgeführt werden, um das Projekt über Kommandozeile zu steuern

Sowohl beim Quell- als auch beim Zielverzeichnis ist eine weitere Strukturierung sinnvoll. Abbildung 1 zeigt im Einzelnen, wie in der Projektstruktur grundlegend Quellen der Entwicklung von denen des Tests getrennt abgelegt werden und wie nach Typ der Quelle, z.B. Java-, XML-, JSP- oder Properties-Dateien getrennt wird. Ein initiales Projekt-Template befindet sich auf der beigelegten DVD.

Eclipse – *die* IDE!

Man darf immer wieder darüber diskutieren, inwieweit es „die" IDE gibt, die allen Wünschen und insbesondere allen Anwendern gerecht wird. In diesem Artikel kommt Eclipse zum Einsatz, welches sich nicht ohne Grund zu einer der beliebtesten Open-Source-IDEs in den letzten Jahren entwickelt hat. Eine schnelle Installation, die erweiterbare Plattform durch Plug-in-Mechanismen, Integration von Build-Management, Testwerkzeugen und CVS sowie zahlreiche komfortable Editoren und eine umfangreiche Hilfe sprechen für Eclipse. Mehr Fakten zum Projekt finden sich unter [3].

Eclipse als Open-Source-Entwicklungsumgebung für Java einzusetzen ist nur eine von vielen Möglichkeiten – weitere sind beispielsweise die NetBeans IDE [14] oder der Oracle JDeveloper [15]. Um Eclipse einzusetzen, genügt ein Download [6] des Eclipse SDK (Software Development Kit) sowie eine JRE (Java Runtime Environment) [7], beide abhängig vom eingesetzten Betriebssystem. Nach der Installation der entsprechenden JRE kann das Eclipse-Archiv extrahiert werden und die Entwicklungsumgebung ist startbereit. Mit Eclipse arbeitet man immer in einem Workspace, welcher physikalisch ein Arbeitsverzeichnis darstellt, in dem unterschiedliche Projekte, Bibliotheken und Konfigurationen gespeichert werden. Startet man Eclipse zum ersten Mal, erscheint ein Welcome-Dialog, welcher auf Tutorials, Code-Beispiele,

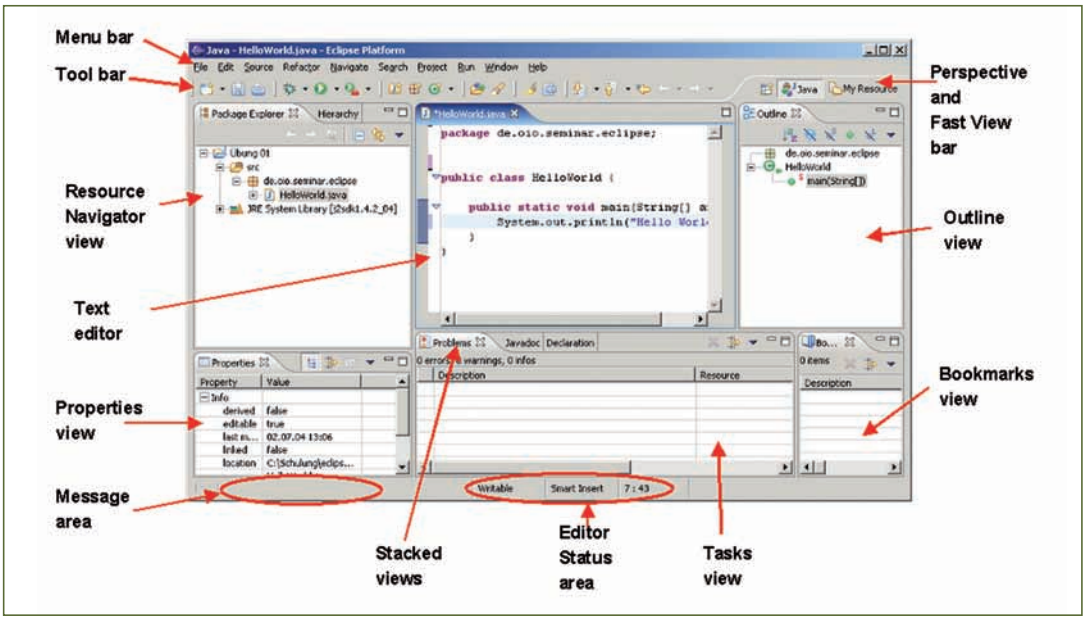


Abb. 2: Eclipse-Terminologie am Beispiel der Java-Perspektive (Quelle: eclipse.org)

Neuerungen und Eclipse im Allgemeinen verweist. Das Arbeiten mit Eclipse erfolgt immer in so genannten Perspektiven. In der Eclipse-Terminologie ist dies eine sinnvolle Zusammenstellung einzelner Fenster, der so genannten Views, die ein konkretes Vorhaben (z.B. Entwicklung, Debugging ...) unterstützen sollen.

Um im Weiteren nicht in einem Verzeichnis-„Wirrwarr" zu enden, wird folgende Verzeichnisstruktur initial definiert:

C:\starter	**im Weiteren STARTER_HOME**
C:\starter\server	als Verzeichnis für die Datenbank MySQL und den Web-Container Tomcat
C:\starter\workspace	als Verzeichnis für das Entwicklungsprojekt

Extrahieren Sie das Eclipse-Archiv in *STARTER_HOME*. Es wird ein neues Verzeichnis namens *eclipse* angelegt, in welchem sich die Startdatei *eclipse.exe* befindet. Führt man diese aus, wird man über einen Dialog gefragt, mit welchem *Workspace* gearbeitet werden soll, da hier Metadaten zu eigenen Projekten abgelegt werden. Bitte setzen Sie über den Browse-Button den Pfad auf *STARTER_HOME\workspace*. Mit dem Ziel, eine Java-Webanwendung (siehe Artikel „Java-Web-Anwendung mit Equinox" auf S. 57) zu erstellen, wechseln wir direkt über den Pfeil oben rechts zum Workspace. Es lohnt sich, mit ein wenig Ruhe z.B. die Tutorials auszuprobieren oder sich ein genaueres Bild über die IDE zu machen.

Was sich nun öffnet, ist die Java-Perspektive. Das Wechseln von einer Perspektive zu einer anderen erfolgt über das Menü Window | Open Perspective | [Name der Perspektive]. Die Java-Perspektive bietet neben dem Package Explorer (links), in dem später die Projekt-Ressourcen angezeigt werden, im unteren Bereich Fenster für Fehlermeldungen (Problems), Javadoc usw., welche über einzelne Reiter aktivierbar sind. Im mittleren Bereich werden sich später die Editoren befinden, rechts die Outline View als eine Art Überblick zur aktuell bearbeiteten Quelle.

Innerhalb von Eclipse arbeitet man immer auf der Basis von Projekten. Das eingangs zusammengestellte Projekt kommt nun zum Einsatz. Extrahieren Sie das *project-template.zip* von der DVD in das Verzeichnis *STARTER_HOME\workspace*. Wechseln Sie zurück zu Eclipse und rufen Sie im Bereich des Package Explorer über das Kontextmenü *IMPORT...* auf. Wählen Sie im folgenden Dialog Existing Projects into Workspace und drücken Sie Next. Navigieren Sie über den Browse-Button auf das Verzeichnis *STARTER_HOME\workspace*. Eclipse findet nun automatisch das Projekt und mit Finish ist das Projekt im Package Explorer auffindbar.

Im Projekt befinden sich eine kleine Java-Anwendung *StringUtil* und eine korrespondierende JUnit-Test-Klasse *StringUtilTest*. Möchte man eine von diesen ausführen, wählt man im Package Explorer die entsprechende Klasse aus und geht über das Kontextmenü auf Run as | Java Application bzw. Run As | JUnit Test. Beim Ausführen einer Java-Klasse erfolgt die Ausgabe in der Console View. Wählt man Run As | JUnit Test wird innerhalb von Eclipse über ein Plug-in die JUnit-Testumgebung gestartet und der Test in dieser durchgeführt.

Während die *src*-Verzeichnisse für die Java-Quelldateien und -Ressourcen direkt unter der Projektwurzel angezeigt werden, muss man für Webinhalte, also JSP, HTML, Bilder und Deskriptor, das Verzeichnis *src–main–webapp* expandieren. Ab diesem Punkt entspricht die relative Verzeichnisstruktur dem Aufbau eines Webarchivs. Ein Web Archive, auch WAR-Datei genannt, ist die deploybare Einheit, welche in einen Web-Container installiert werden kann. Die Struktur dieses Archives ist durch die Java-Servlet-Spezifikation vorgegeben, vgl. Version 2.4 Chapter 9. Bevor die Webinhalte genauer fokussiert werden, veranschaulicht der nächste Abschnitt an einer eigenen Java-Klasse ein paar hilfreiche Features von Eclipse.

Entwickeln mit Eclipse

Öffnen Sie die *StringUtil* aus dem Projekt. Fügen Sie der Klasse rein zur Demonstration folgendes Attribut hinzu: *private*

String text;. Um beispielsweise Zugriffsmethoden („Getter“ und „Setter“) für dieses Attribut bereitzustellen, öffnen Sie direkt im Editor das Kontextmenü über die rechte Maustaste und wählen Sie Source | Generate Getters And Setters. Ein Dialog bietet über Checkboxen die Auswahl der zu generierenden Methoden für die Attribute der Klasse, in diesem Fall *text*. Durch deren Auswahl und Bestätigung mittels OK wird die Klasse nun um eine Methode *getText()* und *setText(String)* erweitert. Im Allgemeinen ist Source der Einstiegspunkt zum Bearbeiten der Java-Quelldatei und Erweitern durch generierte Code-Fragmente. Es lassen sich automatisch Methoden überschreiben, Imports einfügen, try/catch-Blöcke für das Exception Handling einfügen usw. Möchte man mittels Refactoring Quellen überarbeiten, bietet Eclipse Unterstützung über Refactor im Kontextmenü. Hiermit können beispielsweise Konstanten oder Methoden ausgelagert oder Quellcode-Abschnitte verschoben werden, wobei alle Referenzen auf Objekte der verschobenen Quellcode-Inhalte automatisch ein Update erhalten.

Womit Eclipse dem Anwender wirklich Freude bereitet, ist die kontextbezogene Code Completion (Strg + Space) und der berühmte Quick Fix (Strg + 1). Letzterer macht Vorschläge zu sinnvollen Ergänzungen einer Stelle im Quellcode bzw. weist auf Änderungen oder Erweiterungen hin, die zur Behebung eines aktuellen Fehlers notwendig sein können.

Gehen Sie in die *main()*-Methode und schreiben Sie folgenden Code zum Erzeugen eines neuen Objektes der Klasse ein: *new StringUtil();*. Damit das Objekt im Weiteren referenziert werden kann, fehlt eine lokale Variable. Wenn der Cursor direkt hinter dem Semikolon der gerade eingefügten Codezeile steht, aktivieren Sie den Quick-Fix-Mechanismus über Strg + 1. Er bietet Ihnen an, das Objekt einer lokalen Variable oder einem Feld zuzuweisen. Wählen Sie die erste Variante. Der generierte Quellcode ist nun eckig umrahmt, hierdurch kann man Änderungen am Namen und Typ der Variablen vornehmen. Über Enter wird navigiert, über Esc das Ändern abgebrochen. Vervollständigen Sie weiter:

```
StringUtil util = new StringUtil();
util.setText("Test");
```

Bereits beim Setzen des Punktes hinter der Objektreferenz *util* wird Eclipse alle möglichen Aufrufe, die an dieser platziert werden können, über eine Liste anbieten. Um die bearbeitete Klasse zu übersetzen, speichern Sie einfach mittels Strg + S. Ein explizites Übersetzen der Java-Quellen muss nicht ausgeführt werden – Eclipse kommt mit einem eigenen inkrementellen Compiler daher, der grundlegend immer beim Speichern ein Compile durchführt.

Um den Inhalt des *text*-Attributs auf der Console View auszugeben, schreiben Sie in die folgende Zeile *sysout* und betätigen Sie über Strg + Space die Code Completion. Hierdurch wird ein kompletter *System.out.println()*-Aufruf erzeugt, den Sie mit *text* innerhalb der Klammern vervollständigen. Um die Anwendung zu starten, wählen Sie über das Kontextmenü auf der Klasse im Package Explorer Run as | Java Application.

Es bleibt bei dieser „Minivorstellung“ von Eclipse, die einen Vorgeschmack auf das Arbeiten mit der IDE geben möchte. Was diverse „hilfreiche“ Shortcuts betrifft – über das Menü Help | Key Assist können Sie sich diese einblenden lassen. Für einen umfangreichen Einstieg in die IDE sei auf die Hilfe sowie auf zahlreiche Artikel, z.B. im *Eclipse Magazin*, verwiesen.

Arbeiten mit und ohne IDE – Java-Build-Management mit Apache Ant

Mit Eclipse sind nun Editoren, Compiler und Intepreter komfortabel zur Java-Entwicklung vorhanden. Wie aber kann man aus dem Projekt ein Web Archive erstellen, die „Einheit“, die ein Web-Container entgegennimmt, um sie als Webanwendung bereitzustellen? Und wie funktioniert das Deployment, d.h. die Installation in den Container? Eine mögliche Variante, diesen Anforderungen zu begegnen, bietet das Java-Build-Management-Werkzeug Apache Ant. Ant ist auf jeder Java-fähigen Plattform einsetzbar und durch die Verwendung von XML mit einer portablen Syntax ausgestattet. Als De-facto-Standard im Java-Umfeld findet es in der Java-Welt flächendeckend Einsatz für das Build-Management und wird neben

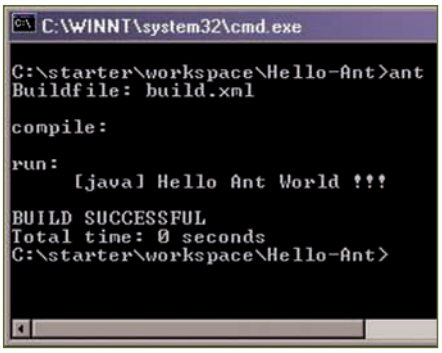


Abb. 3: Ant über Kommandozeile ausgeführt

Eclipse von vielen IDEs unterstützt. Mit Ant lassen sich eine Menge auszuführender Anweisungen zusammengefasst aufrufen, die in XML-basierten Dateien definiert werden. Typische Vorgänge, die über Ant gesteuert werden, sind das Packaging, das Deployment oder das Ausführen von Tests. Ein Blick in das Ant Manual [8] verrät, wie multifunktional Ant ausgestattet und somit einsetzbar ist.

Zur Installation von Ant sind wenige Schritte notwendig. Unter [5] ist die aktuelle Binärdistribution herunterzuladen. Extrahieren Sie das Ant-Archiv nach *STARTER_HOME*. Setzen Sie auf dieses Verzeichnis eine Umgebungsvariable *ANT_HOME*. Um Ant über die Kommandozeile aufrufen zu können, erweitern Sie Ihre Path-Variable um *%ANT_HOME%\bin*. Voraussetzung ist natürlich ein installiertes JDK (Java Development Kit). Dies sollte über die Variable *JAVA_HOME* zur Verfügung gestellt werden.

Um Ant über Kommandozeile auszuführen, benutzt man den Aufruf *ant*. Im entsprechenden Verzeichnis wird nach einem so genannten Build File gesucht, welches die für ein Projekt definierten Targets, also ausführbaren Anweisungen, enthält. Per Default heißt diese Datei *build.xml*. In Listing 1 ist eine einfache „Hello Ant“ Build XML abgebildet. Sie enthält eine *project*-Definition. Die drei Targets *compile*, *run* und *clean* stellen die ausführbaren Anweisungen dar. Zur Demonstration befindet sich ein Beispielprojekt auf der CD. Extrahieren Sie das Archiv *Hello-Ant.zip* in *STARTER_HOME\workspace* und navigieren Sie über Kommandozeile auf das neu angelegte Verzeichnis. Geben Sie nun *ant* ein. Das, was erscheinen sollte, sehen Sie in Abbildung 3.

Das Vorgehen von Ant lässt sich wie folgt beschreiben: Im *project*-Element ist ein *default* Target names *run* definiert. Dieses Target beschreibt über ein *depends*, dass vor dessen Ausführung erst das Target *compile* ausgeführt werden muss. In *compile* wiederum wird der Java Compiler aufgerufen. Die komplette Anweisungsfolge lautet: *«Kompiliere alle Dateien aus dem Projektverzeichnis und führe danach eine Klasse namens de.oio.starter.Hello aus.»* Möchte man einzelne Targets ausführen, schreibt man in der Kommandozeile den Befehl „ant“ und fügt Leerzeichen getrennt zum Namen des gewünschten Targets hinzu, z.B. *ant clean*.

Da Ant in Entwicklungsumgebungen integriert werden kann, ist es möglich, direkt aus Eclipse heraus eine *build.xml* auszuführen. Wechseln Sie hierzu zurück zu Eclipse und importieren Sie im Package Explorer über das Kontextmenü – Import – das *HelloAnt*-Beispiel in den Workspace. Gehen Sie nun in das Menü Window | Show View | Ant. Für Ant Build Files stellt Eclipse eine View bereit, die im rechten Bereich Ihrer IDE erscheint. Ziehen Sie per Drag & Drop aus dem *HelloAnt*-Projekt die *build.xml* in die Ant View. Nun sehen Sie die einzelnen Targets aufgelistet, die per Doppelklick ausgeführt werden können. Das Default Target ist blau hervorgehoben. Beim Ausführen schreibt Eclipse den Ant-Output auf die Console View im unteren Bereich.

Im Projekt *project-template* befindet sich auch eine vordefinierte *build.xml*. Ziehen Sie diese in die Ant View. Hier ist das Default Target *package*. Es erstellt eine WAR-Datei. Führen Sie das Target aus und sehen Sie sich das Ergebnis im Verzeichnis *target* an. Neben dem Erzeugen eines Web Archives bringt die Projekt-*build.xml* Möglichkeiten zum Deployment und Undeployment mit. Diese beziehen sich auf einen Apache Tomcat. Tomcat unterstützt Remote Deployment und bietet hierzu Ant Targets an. Das Ausführen des Remote Deployment bei Tomcat ist nur Benutzern in der Rolle „manager“ erlaubt. Per Default muss ein solcher Benutzer in der *tomcat-users.xml* im *conf*-Verzeichnis des Tomcat angelegt sein. Für das definierte Projekt wird ein Benutzer *manager* mit Passwort *manager* in der Rolle *manager* benötigt. In Listing 2 ist der notwendige Eintrag in der *tomcat-users.xml* abgebildet. Mit diesem Benutzer kann die *Starter.war* in einen Tomcat deployt werden. Wie Sie den Tomcat installieren und in Betrieb nehmen, finden Sie ausführlich auf Seite 28.

Starten Sie Ihren Tomcat über die *startup.bat* aus dem *bin*-Verzeichnis und führen Sie in der Eclipse Ant View das Target *deploy* mittels Doppelklick aus. Über eine interne Web Browser View kann man sich das Ergebnis direkt aus Eclipse heraus ansehen. Gehen Sie hierzu auf Window | Show View | Other. Im Dialogfenster wählen Sie aus dem Verzeichnis *Basic* den

Listing 1

„Hello Ant“ Build XML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<project default="run" name="HelloAnt">

 <target name="compile">
  <javac srcdir="."/>
 </target>

 <target name="run" depends="compile">
  <java fork="yes" classname=
                "de.oio.starter.Hello" classpath="."/>
 </target>

 <target name="clean">
  <delete file="de/oio/starter/ant/Hello.class"/>
 </target>

</project>
```

Listing 2

tomcat-users.xml mit einem Manager

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
 <role rolename="tomcat"/>
 <role rolename="role1"/>
 <role rolename="manager"/>
 <user username="tomcat" password="tomcat" roles=
                                        "tomcat"/>
 <user username="role1" password="tomcat" roles=
                                        "role1"/>
 <user username="both" password="tomcat" roles=
                                   "tomcat,role1"/>
 <user username="manager" password=
                          "manager" roles="manager"/>
</tomcat-users>
```

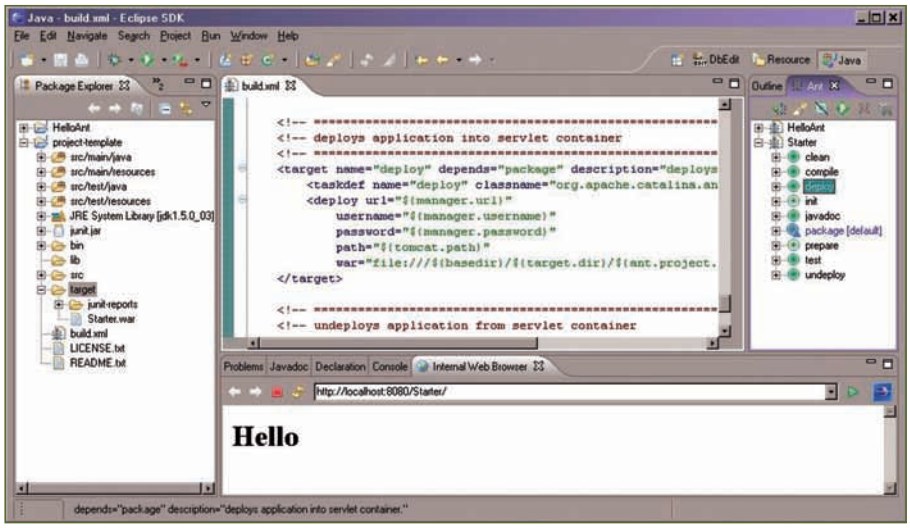


Abb. 4: Interner Web-Browser von Eclipse

Internal Web Browser. Er wird initial im unteren Bereich zu den bereits vorhandenen Reitern gelegt. Über den URL *http://localhost:8080/Starter/* kann das Ergebnis angesehen werden.

Eclipse zur Java-Webentwicklung

Eine eigene Projektstruktur für Java-Webanwendungen ist angelegt. Diese ist in Eclipse integriert und kann über Ant gesteuert werden. Eigentlich kann nun die Webentwicklung direkt beginnen. Andererseits wäre es z.B. praktischer, Tomcat direkt aus Eclipse zu steuern oder eine verwendete Datenbank direkt beim Entwickeln in der IDE einsehen zu können. Beides ist möglich, da Eclipse mit einem so genannten Plug-in-Konzept arbeitet. Die IDE selbst ist eine Zusammensetzung aus einem Eclipse-Kernsystem und vielen Plug-ins. Über wohldefinierte Schnittstellen können Plug-ins mit dem Kernsystem und untereinander kommunizieren und die gesamte Eclipse-Infrastruktur benutzen. Dies bedeutet beispielsweise, dass ein Eclipse-Plug-in eigene Views anbieten kann, welche sich in Standard-Perspektiven von Eclipse einbinden lassen. Diese Views können auf Ereignisse aus anderen Views reagieren oder umgekehrt selbst Ereignisse auslösen, die über die Infrastruktur delegiert werden.

Für die Verwendung von Tomcat innerhalb von Eclipse gibt es von Sysdeo ein frei verfügbares Tomcat-Plug-in [9]. Mit ihm kann man den Tomcat aus Eclipse heraus starten und stoppen, Projekte direkt mit Tomcat erstellen und innerhalb der Webanwendung debuggen. Die Installation ist schnell vollzogen. Unter [9] wird das Tomcat-Plug-in zum Download bereitgestellt. Das heruntergeladene *tomcatPluginV31.zip* muss in *STARTER_HOME\eclipse\plugins* extrahiert werden, um von Eclipse bereitgestellt zu werden. Starten Sie nach dem Extrahieren des Archivs Eclipse neu. In der oberen Menüzeile sind drei neue Schaltflächen zum Starten, Stoppen und Restart des Tomcats zu sehen. Damit diese verwendet werden können, erfordert das Plug-in die Einstellung, wo sich ein zu verwendender Tomcat befindet.

Wählen Sie das Menü WINDOW | PREFERENCES. Im linken Baum findet sich ein Knoten *Tomcat*. Gehen Sie auf diesen und geben Sie im rechten Dialog Ihre Tomcat-Version an sowie den Pfad, wo sich Tomcat befindet. Bestätigen Sie Ihre Eingabe mit APPLY und schließen Sie den Dialog (Abb. 5). Nun können Sie die Schaltfläche TOMCAT STARTEN betätigen. Wichtig ist hierbei, dass außerhalb der IDE nicht noch eine Tomcat-Instanz gestartet ist. Dies könnte zu einem Port-Konflikt führen. In der Console View von Eclipse sehen Sie nach dem Betätigen des Tomcat-Start-Buttons die Tomcat-Ausgabe. Über den Web-Browser können Sie erneut versuchen, die Starter-Anwendung mit dem URL *http://localhost:8080/Starter/* aufzurufen.

Das Tomcat-Plug-in startet den Web-Container automatisch im Debug Mode. Diese Funktionalität kann durch Setzen

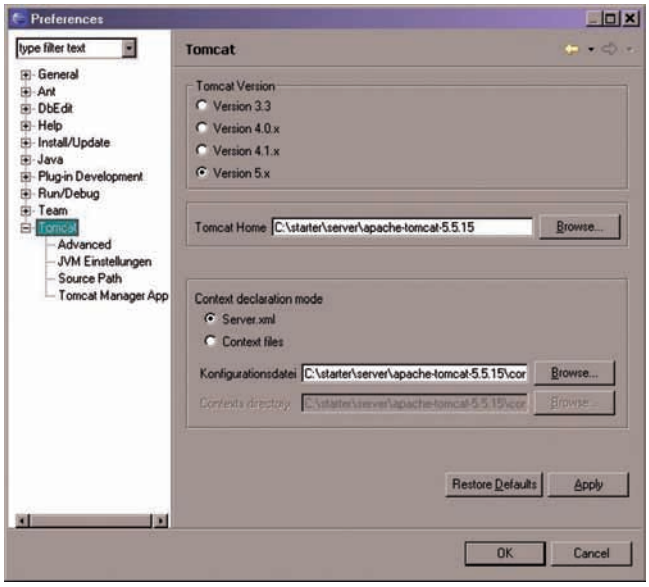


Abb. 5: Konfiguration des Tomcat-Plug-ins in Eclipse

eines Breakpoints (über das Kontextmenü im rechten Rand eines Editors | Toggle Breakpoint) an der entsprechenden Java-Quelle genutzt werden. Sobald über den Browser ein Request gesendet wird, welcher diese Java-Klasse ausführt, wechselt Eclipse in eine Debug-Perspektive. Hier kann der Quellcode zeilenweise durchschritten und mit seinen Laufzeitwerten eingesehen werden.

Weiter ist es zu empfehlen, Eclipse mit dem DBEdit-Plug-in [10] aufzustocken, um einen Blick auf die verwendete Datenbank zu haben und ggf. direkt Änderungen vornehmen zu können. DBEdit arbeitet mit dem JDBC API [11] und erhält hierdurch ebenso seine Einschränkungen. Das Plug-in erhebt nicht den Anspruch, ein vollwertiges Administrationswerkzeug zu ersetzen. Vielmehr ist es eine Anreicherung, um aus Eclipse schnell und einfach Zugriff auf eine Datenbank zu erhalten. Zum Einbinden in Eclipse laden Sie das DBEdit-Archiv *dbedit_1.0.3_1.bin.dist_3.X.zip* unter [10] herunter und extrahieren es nach *STARTER_HOME\eclipse*. Starten Sie Eclipse neu und gehen Sie über Window | Open Perspective | Other auf die DBEdit-Perspektive. Im linken Bereich können verschiedene Verbindungen hinterlegt werden, deren Präsentation in Form von Tabelleninhalten im mittleren Bereich erfolgt.

Ein weiterer Artikel dieser Ausgabe stellt die Datenbank MySQL vor (siehe Seite 34). Über DBEdit wird diese als Beispiel in Eclipse eingebunden. Um auf die MySQL-Datenbank zugreifen zu können, wird ein MySQL JDBC Connector benötigt, welchen Sie unter [12] zum Download finden. Extrahieren Sie den Connector in Form der Datei *mysql-connector-java-3.1.12-bin.jar* nach *STARTER_HOME\server\mysql\connector*.

Wechseln Sie zurück zu Eclipse und gehen Sie in der linken View der DBEdit-Perspektive über das Kontextmenü auf New | Connection. Ein Dialog öffnet sich. Geben Sie als Namen für diese Connection *MySQL* an. Bevor Sie Einstellungen zum Treiber usw. vornehmen, wechseln Sie auf den Reiter *Classpath*. Fügen Sie über den Button Add Archiv den MySQL Connector (*mysql-connector-java-3.1.12-bin.jar*) aus zuvor angelegtem Verzeichnis hinzu. Wenn Sie wieder zurück auf den Reiter *Common* wechseln und die Auswahlbox öffnen, finden Sie die Klassennamen aller verfügbaren Treiber von MySQL. Wählen Sie *org.gjt.mm.mysql.Driver* aus und gehen Sie weiter zum Eingabefeld *Server URL*. Nach folgendem Pattern wird der URL erstellt: *jdbc:mysql://host:port/dbname*. Verwenden wir beispielsweise auf *localhost* mit dem Default Port 3306 eine Datenbank namens *StarterDB*, sieht der Server-URL-Eintrag folgendermaßen aus: *jdbc:mysql://localhost:3306/starterdb*. Geben Sie, sofern definiert, noch den Benutzer an, mit dem Sie sich auf die Datenbank verbinden wollen. Nun kann über den Button Connect eine Verbindung erstellt werden. Im linken Bereich werden die Daten angezeigt, die Tabellen sind in einer Baumstruktur navigierbar. Durch Doppelklick auf einer Tabelle, wird diese im mittleren Bereich geöffnet (Abb. 6).

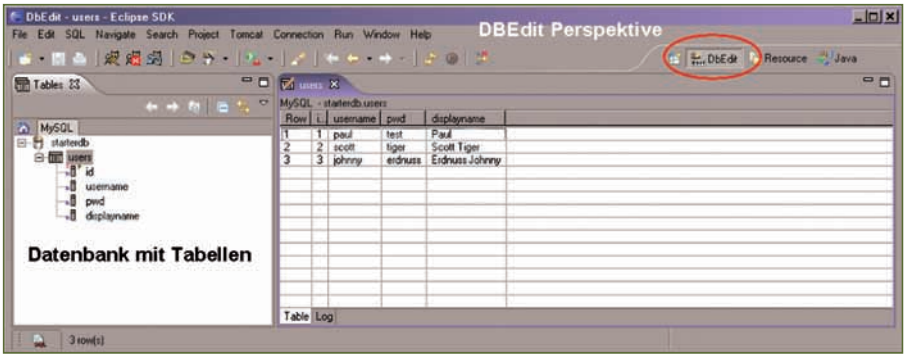


Abb. 6: DBEdit-Perspektive in Eclipse

Die Anbindung zu Web-Container und Datenbank ist konfiguriert. Für die für Webkomponenten spezifischen Inhalte fehlen noch Editoren. Beispielsweise wäre ein JSP-Editor hilfreich, welcher automatisch Tag Libraries [4] (kurz: Taglib), dies ist ausgelagerter Java-Quellcode, den man innerhalb einer JSP über Tags aufrufen kann, anhand des Deskriptors auslesen kann und zur Vervollständigung anbietet. Ebenso wären HTML- und XML-Editoren beim Einsatz dieser Markup-Sprachen sinnvoll. Gerade beim Einsatz von XML ist es hilfreich, mit DTD und XML-Schema-Validierern arbeiten zu können, damit beim Erstellen von XML-Dokumenten Elemente konform zu diesen erstellt werden können.

Ein eigenes Eclipse widmet sich genau diesen Anforderungen und bietet hierzu eine umfangreiche Palette an Funktionalität an – die Web Tools Platform [5]. Es wäre ein Fehler, diese Plattform als reine Editoren-Sammlung für Webinhalte anzusehen. Vielmehr ist es eine voll

automatisierbare Umgebung, die den kompletten Entwicklungsweg von Java EE-Komponenten durch definierte Projekt-Templates, Dialoge zum Erstellen der einzelnen Komponenten bis hin zum Deployment und Testen unterstützt.

Es ist erlaubt zu fragen, warum nicht von Beginn an die Web Tools Platform eingesetzt wurde. Während bisher in vielen kleinen Schritten eine Umgebung zusammengestellt wurde, wurden typische Punkte angesprochen, die auf diesem Weg notwendig sind. Bei Web Tools wäre sicher ein einfacherer Weg denkbar gewesen. Die bisher gewonnene Flexibilität und bereits eingangs Angesprochenes ließen von dieser Variante absehen.

Die Installation der Web Tools Platform (WTP) bedarf einiger Schritte – es muss nicht nur das 1.0.1-Release heruntergeladen werden [5]. WTP benötigt auch den EMF-(Eclipse Modeling Framework-) Treiber, den Graphical Editing Framework-(GEF-)Treiber sowie die Java EMF Model Runtime. Auf all diese Archive wird direkt auf der WTP-Download-Seite verwiesen. Nachdem Sie die Archive heruntergeladen haben, extrahieren Sie diese in *STARTER_HOME* und starten Sie erneut Eclipse aus *STARTER_HOME\eclipse*.

Gehen Sie im „Starter"-Projekt in das Verzeichnis *src\main\webapp*. Hier befinden sich die Webinhalte der Anwendung. Markieren Sie das Verzeichnis und öffnen Sie das Kontextmenü. Über New | Other öffnet sich ein Fenster mit einer Vielzahl von Typen.

Gehen Sie in den Ordner *Web* und wählen Sie *JSP* aus. Drücken Sie Next und geben Sie als Namen *test.jsp* an. Nun können Sie mit Finish bestätigen. Gehen Sie in den JSP Editor und positionieren Sie beliebig den Cursor. Wenn Sie nun Strg + Space drücken, kommt die Vervollständigung für HTML- und JSP-Elemente. Ebenso werden innerhalb der Elemente alle Attribute angeboten (Abb. 7) zeigt.

Richtig komfortabel ist das Arbeiten mit Tag Libraries. Über einen so genannten Taglib-Deskriptor (TLD) wird ausgelagerte Funktionalität mit Elementen gemappt, die innerhalb der JSP ausgeführt werden können. Im Projekt befindet sich unter *src\main\webapp\WEB-INF* ein solcher Deskriptor aus den JavaServer

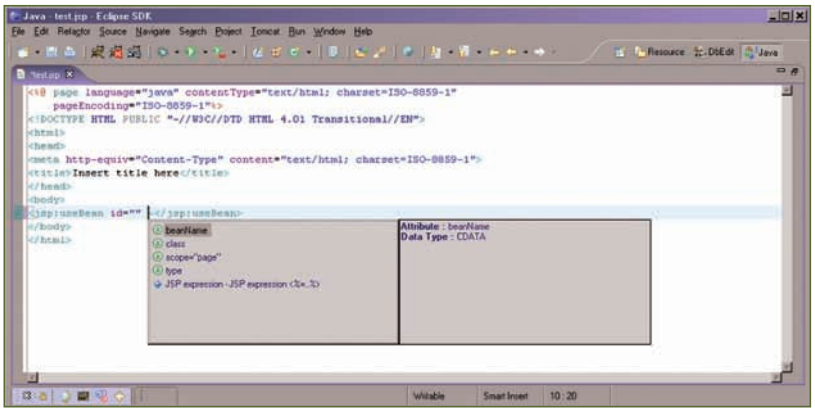


Abb. 7: JSP Code Completion mit der Eclipse WTP

Pages Standard Tag Libraries (JSTL). Die benötigten Bibliotheken sind bereits im *lib*-Verzeichnis abgelegt. Eine Bibliothek der JSTL ist die Core Taglib. Sie bietet Standard-Programmierfunktionalität, z.B. Bedingungen, Schleifen, Imports usw. an. Um die Core Taglib (Abkürzung: c) in der *test.jsp* zu verwenden, muss zunächst über eine Taglib-Direktive die Verwendung auf der Seite bekannt gegeben werden. Fügen Sie unterhalb der Page-Direktive (Element <%@ *page*...) die folgende Zeile ein:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/
core"%>
```

Gehen Sie zwischen das geöffnete und geschlossene *body*-Element und beginnen Sie *<c:* zu schreiben. Über Strg + Space können Sie sich alle verfügbaren Tags der Core Taglib einblenden lassen. Fügen Sie folgende Zeile ein, die eine einfache Ausgabe macht: *<c:out value="Test"/>*.

Um das Ergebnis zu sehen, reicht ein einfaches Deployment in den Tomcat. Bitte starten Sie Tomcat, sofern dies nicht bereits der Fall ist, und führen Sie aus der Eclipse Ant View das Target *deploy* aus. Ist die Anwendung noch im Tomcat, meldet Ant einen Fehler.

Führen Sie dann zunächst das Target *undeploy* aus, danach erst *deploy*. Über den internen Web-Browser lässt sich nun unsere JSP unter dem URL *http://localhost:8080/Starter/test.jsp* ansehen. Man kann nun leicht Veränderungen an der JSP durchführen, das Deployment über Ant starten und innerhalb von wenigen Sekunden das neue Ergebnis im Browser sehen. Beim Arbeiten mit einer Datenbank kann bis in dieses Medium eine Veränderung nachvollzogen werden.

Ein erstes Java-Starter-Cockpit ist zusammengestellt – für folgende Artikel zum Erstellen einer Webanwendung ist die Umgebung so eingerichtet, dass Sie nun Ihre Aufmerksamkeit ganz dem Technologischen widmen können. Viel Spaß dabei und natürlich bei weiteren Erkundungen in die vielen Eclipse-Details, die in diesem Artikel keinen Platz gefunden haben.

Sabine Winkler (winkler@oio.de) ist bei der Firma Orientation in Objects in Mannheim als Berater, Trainer und Entwickler im Bereich Java und XML beschäftigt. Im Bereich Java EE ist ihr besonderes Interesse der Webentwicklung und EJB im Zusammenhang mit Clustering gewidmet.

Links & Literatur

[1] Eclipse: www.eclipse.org
[2] Ant: ant.apache.org
[3] Java Servlet: java.sun.com/products/servlet/
[4] JavaServer Pages: java.sun.com/products/jsp/
[5] Eclipse Web Tools Platform (WTP): www.eclipse.org/webtools/
[6] Eclipse-Downloads: www.eclipse.org/downloads/
[7] Java Standard Edition 5: java.sun.com/j2se/1.5.0/download.jsp
[8] Apache Ant 1.6.5 Manual: ant.apache.org/manual/
[9] Tomcat-Eclipse-Plug-in: www.sysdeo.com/eclipse/tomcatplugin
[10] DBEdit-Eclipse-Plug-in: www.geocities.com/uwe_ewald/dbedit.html
[11] JDBC API: java.sun.com/products/jdbc/
[12] MySQL JDBC Connector: www.mysql.com/products/connector/j/
[13] JavaServer Pages Standard Tag Libraries: java.sun.com/products/jsp/jstl/
[14] NetBeans: www.netbeans.org
[15] Oracle JDeveloper 10g: www.oracle.com/technology/products/jdev/