

## IntelliJ IDEA 6.0: Rechtfertigen die neuen Features den Upgrade?

# Demetra

■ VON MICHAEL HÜTTERMANN

Die Softwareschmiede JetBrains veröffentlichte Anfang Oktober die Version 6.0 ihrer preisgekrönten kommerziellen Entwicklungsumgebung IntelliJ IDEA – Codename: Demetra. Zahlreiche neue Features sind gegenüber der Vorgängerversion hinzugekommen. Doch lohnt sich ein Upgrade auf die neue Version [1]?

Das im Jahre 2000 gegründete Unternehmen JetBrains hat seinen Hauptsitz in Prag und sein Research & Development Center in St. Petersburg. Durch seine Produkte, allen voran IntelliJ IDEA, gelang es JetBrains, sich einen guten Ruf in der Entwickler-szene zu erarbeiten. Trotz der namhaften Konkurrenz durch kommerzielle Markt-begleiter und kostenlose Alternativen wie Eclipse und NetBeans gewann IDEA zahlreiche Preise, darunter den zweiten Platz als beste Entwicklungsumgebung der Readers' Choice 2006 des *Java Magazins* [2] und den zweiten Platz als beste Java-Anwendung bzw. den ersten Platz als beste Java-IDE beim JDJ Readers' Choice Award 2005 [3]. Doch kann auch die neue Version 6.0 Preise gewinnen? JetBrains hat in vielen Bereichen neue Funktionalitäten ergänzt und alte ausgebaut. Schauen wir uns die Version einmal genauer an.

### Java EE und Web Development

Ein erster Eyecatcher ist die ausgebaut JavaScript-Unterstützung. Die bereits vor 6.0 angebotenen Dienste wie Code Completion, Error Highlighting, Refactorings, Code Formatting, Find und Highlight Usages, Code Folding, Go-to- und Label-Unterstützung sowie die hilfreiche Structure View wurden weiter ausgebaut. Nun existieren zahlreiche weitere Code Inspections: Mehr als 70 Checks, z.B. auf unerreichbaren Code, unterstützen den Entwickler bei der täglichen Arbeit. Ergänzt wurden weitere Refactorings, Quick Fixes und eine Smart Completion anhand von Detection Patterns. Diese basiert auf statischen Deklarationen, welche

Muster wiedererkennen und fremde Libraries testbar machen sollen.

Wer schon einmal umfangreiche Webanwendungen für unterschiedliche Zielplattformen (sprich: Browser) entwickelt hat, der wird die browserspezifische Code Completion zu schätzen wissen. Sie kann bei der Arbeit mit unterschiedlichen DOM-Derivaten wahre Wunder wirken: Neben dem einfügbaren Code erscheinen Icons, welche Aufschluss über die jeweilige Browser-Unterstützung von Internet Explorer, Mozilla, Opera und Safari dieser Properties oder Methoden geben (Abb. 1). Weiterhin existieren browserabhängige Warnings: Cross Browser Scripting leicht gemacht.

Der Web Developer wird nicht nur an der JavaScript-Unterstützung seine Freude haben: Neue Code Inspections sowie Search & Replace-Unterstützung existieren nun auch für HTML und XML. Auch CSS kommt bei Inspections nun nicht mehr zu kurz. Darüber hinaus wird jetzt auch das XML-DocBook-Format unterstützt: Wer also seine Dokumentation in diesem Format ablegt, kann dies nun auch via IDEA machen. Summa summarum bietet JetBrains eine breite Ajax-Unterstützung. Böse Zungen behaupten, Ajax sei lediglich „alter Wein in neuen Schläuchen“, zumindest kann sich der geneigte IDEA-Entwickler nun mit den unterstützten Frameworks Dojo, Bindows und Prototype sowie der neuen GWT-Unterstützung selbst ein Bild machen. Diese umfasst eine Lauf- und Debug-Umgebung sowie die obligatorische IDEA-Code-Unterstützung, welche auch für eingebettetes JavaScript gilt.

Wie sieht es serverseitig aus? Totgesagte leben länger: Ein klares Signal pro Struts ist die nun eingebrachte Struts-Unterstützung. Wird ein Webmodul via Kontextmenü für die Struts-Entwicklung vorbereitet, werden automatisch die notwendigen Artefakte von [ibiblio.org](http://ibiblio.org) heruntergeladen, dessen URL jeder Maven 2-Nutzer nur allzu gut kennt. Die Struts-Unterstützung umfasst Features wie Tiles, Validator und Struts-EL Taglibs, visuelle Editierung der Konfiguration, einen Properties Inspector und die für IDEA obligatorischen kontextsensitiven Refactorings, Quick Fixes und das Highlighting. Nicht zuletzt durch die Neuausrichtung an JSF (Stichwort: Shale) und die Verschmelzung mit WebWork hat Struts zur richtigen Zeit die notwendigen, richtungsweisenden Schritte vollzogen. Das häufig eingesetzte Struts Framework ist nun wieder zukunftssicher geworden, was sich auch in der Unterstützung in IDEA widerspiegelt.

Die IDEA-Unterstützung für Java EE 5 inkl. der Komponenten EJB, Servlet, JSP und JSF wurde auf den neuesten Standard gebracht. So wird nun auch EJB 3.0 unterstützt. Dies enthält Annotations-Un-

### IDEA 6.0: Scope der Neuerungen

- Java EE & Web Development
- JUnit 4 & Code Coverage/Testabdeckung
- Erweiterter Swing-GUI-Designer
- Produktivitäts- und Editier-Features
- Extended Type System: Qualitätserhöhung mit `@Nullable` und `@NotNull`
- Teamwork-Funktionalität – Collaboration

terstützung, auch im gemischten Modus: EJBs können sowohl via Annotations als auch mit Deskriptoren deklariert werden. Bei der Entwicklung von Entity Beans, Message Driven Beans und Session Beans sind umfangreiche Refactorings, Quick Fixes und Checkings verfügbar, die sowohl bei den Java-Artefakten als auch bei den Deployment-Deskriptoren allesamt „EJB-aware“ sind.

Eine auf einem Datenbank-Schema basierte Persistence Unit lässt sich recht einfach erstellen: Zunächst muss eine DATA SOURCE VIA TOOLS | DATA SOURCES konfiguriert werden, anschließend gilt es, auf dem EJB-Modul via GENERATE PERSISTENCE MAPPING | BY DATABASE SCHEMA in einem Dialog die Tabellen zu den Entities zu mappen. Ein Persistence-Diagramm kann dargestellt werden; F4 auf einer bestimmten Persistence Unit genügt. Es können ferner Entities aus Hibernate oder JDBC-Sourcen generiert werden.

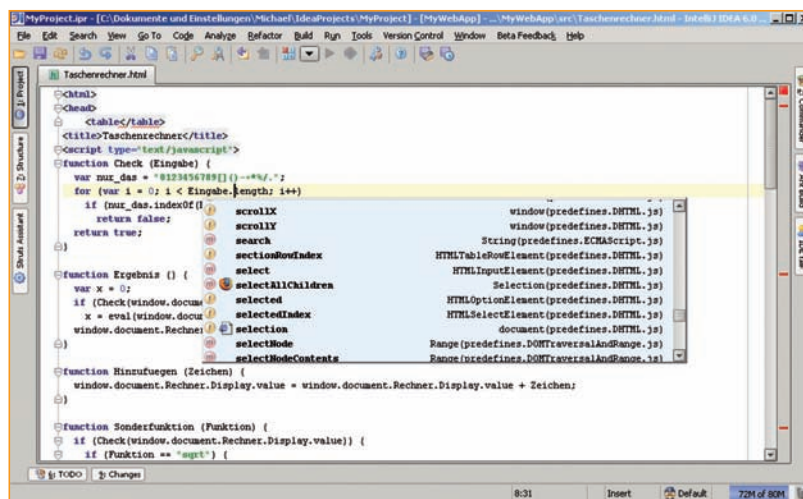
Interessant ist auch die Migrationsunterstützung: IDEA 6.0 verfügt über Funktionalität, um Legacy EJBs auf EJB 3 zu migrieren. Dazu gehört, dass Deskriptoren in Annotations umgewandelt sowie die Klassen beziehungsweise Interfaces transformiert und aufgeräumt werden. Auch lassen sich Entity Beans auf CMP überführen. Um den Assistenten aufzurufen, reicht ein Rechtsklick auf die Bean oder auf das ganze EJB-Modul und eine Selektion des Menüpunktes APPLY EJB 3.0 STYLE.

Die Unterstützung von Application-Servern wurde erweitert. Wurden früher Tomcat und Weblogic 8.x unterstützt, so existiert mit 6.0 Unterstützung für WebLogic 9.x, WebSphere, JBoss, Geronimo und Glassfish. Die integrierte Deployment- und Debugging-Unterstützung für diese Vielzahl von Servern zieht im Falle von JBoss und WebSphere lediglich mit dem Mainstream mit.

Beachtlich ist allerdings die Unterstützung von Geronimo und Glassfish, wobei gerade Letzterer in der nahen Vergangenheit von sich reden gemacht hat und mittlerweile durchaus als Production Stable angesehen wird.

Bei JSP hat sich auch einiges getan: Nun wird die Unified Expression Language unterstützt. Damit einher geht die

Abb. 1: Java-Script-Unterstützung: Lookup auf DOM, browser-abhängig



Unterstützung der EL Resolvers. Sämtliche Java-Features der IDE werden auch innerhalb von JSP-Seiten angeboten. Das Gleiche gilt übrigens auch für die HTML-, CSS- und JavaScript-Unterstützung, die dem Entwickler innerhalb der JSP weiter zur Verfügung steht. Es lassen sich Custom Tag Libraries erstellen, welche auf unterschiedlichen, verteilten JSP- und JSPX-

Seiten genutzt werden können, vorausgesetzt, die Tag Library wird installiert.

JSF wird nun in der Version 1.1 unterstützt. Es existiert kein eigenes Modul für die JSF-Entwicklung, die Nutzung wird vielmehr automatisch innerhalb des Webmoduls erkannt. Ein Modul kann „JSF-enabled“ werden, sodass IDEA dann automatisch die notwendigen Artefakte

## Anzeige

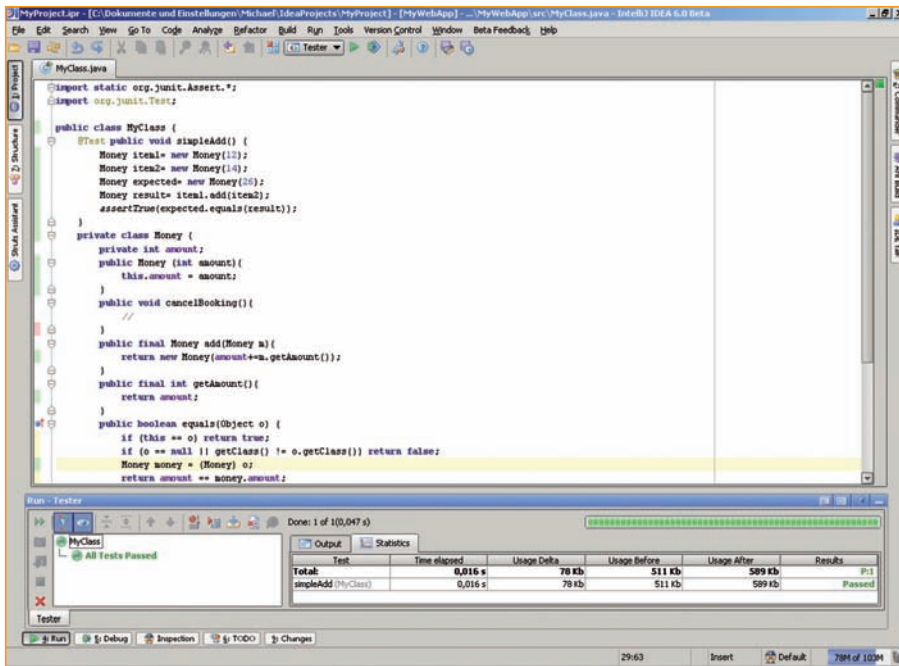


Abb. 2: JUnit 4-Unterstützung mit Testabdeckung

von dezidierten JetBrains-Seiten downloaden kann. Abhängig von den Vorlieben des Nutzers wird das Modul für eine Entwicklung nach JSF 1.0 oder 1.1 sowie der JSF-Referenzimplementierung oder Apache MyFaces hergerichtet. Die eigentliche Entwicklung mit einem formularbasierten Konfigurationseditor für die *faces-config.xml* inklusive Error Highlighting und Code Completion ist sehr gut möglich. Auch JSF-Navigationsdiagramme sind verfügbar: Der Page Flow kann grafisch modelliert werden und wird von IDEA automatisch in Code transformiert. Invalide Navigationsregeln werden angezeigt, können via Doppelklick behoben und via Strukturtabulator optimiert werden.

Eine Kombination respektive Integration ist bequem möglich: JSF innerhalb von JSP-Seiten mit JSF EL Support, JSF-spezifischen Checks und nun mit JBoss Seam-Unterstützung. Bei der JSF-Unterstützung ist sicherlich noch Luft nach oben und für die Zukunft einiges zu erwarten.

### JUnit 4 + Testabdeckung

Ein Höhepunkt des Release ist zweifellos die JUnit 4-Unterstützung verbunden mit der Code-Coverage-Funktionalität. IDEA 6.0 unterstützt JUnit 4 voll und ganz, sodass Testklassen nun komfortabel mithilfe von Annotations erstellt werden können.

Das angefügte Beispiel in Abbildung 2 zeigt, wie dem Framework mithilfe der Annotation *@Test* signalisiert wird, die Methode *simpleAdd()* zu testen. Zugrundeliegt eine einfache, innere Klasse, die neben einem Konstruktor, einem Getter und exemplarischen fachlichen Methoden auch *equals* und *hashCode* überschreibt. In dieser Form kann, wie im Snippet dargestellt, mit der Methode *assertTrue* auf Gleichheit überprüft werden. Nachdem JUnit im Klassenpfad ergänzt wurde, kann über *RUN | EDIT CONFIGURATIONS* eine Laufkonfiguration erstellt werden. Hier wird die zu testende Klasse angegeben und ob eine Überprüfung der Testabdeckung durchgeführt werden soll. Im vorliegenden Fall ist der Unit-Test erfolgreich, was wir an dem JUnit-Statusbereich unten in Abbildung 2 sehen. Es empfiehlt sich, bei den Projekt-Settings das eine oder andere Checking für JUnit einzuschalten.

Die Code-Coverage-Unterstützung basiert auf dem freien Framework EMMA [4] und die Integration ist geglückt: Sowohl Unit-Tests als auch Tests, die mit einfachen Applikationen und einer *main*-Methode durchgeführt werden, können überprüft werden. Die Code-Coverage-Untersuchung wird on the fly durchgeführt, kommt also ohne ein Prescanning der Klassen aus. Die Statisti-

ken können über *VIEW | SHOW CODE COVERAGE INFORMATION* für Klassen oder ganze Packages dargestellt werden. Dabei wird im linken Bereich des Editors ein farbiger Balken dargestellt: roter Balken: keine Abdeckung; gelber Balken: teilweise Abdeckung (z.B. durch if-Zweige); grüner Balken: komplette Abdeckung. Eine schöne Erweiterung für Freunde der testgetriebenen Entwicklung, die nicht gerade das mächtige TestNG nutzen.

### Erweiterter Swing-GUI-Designer

IDEA 6.0 hat seinen GUI-Designer weiter ausgebaut. Bei der Erstellung eines UIs kann aus einer breiten Palette von Layout-Managern gewählt werden. Neben den standardisierten Swing-LayoutManagern zählen auch JGoodies (wer möchte FormLayout wirklich missen?) und ein JetBrains-eigener GridLayoutManager dazu. IDEA 6.0 baut auf Java 5 auf, Java 5 ist also für den Betrieb Voraussetzung, und so ist es nicht verwunderlich, aber dennoch schade, dass das mit Java 6 verfügbare GroupLayout nicht enthalten ist.

Durch Ziehen von grafischen Swing-Komponenten auf die Form-Arbeitsfläche werden automatisch so genannte Grids erstellt. Grid Rows und Columns können per Drag & Drop verschoben oder auch gelöscht werden. Morphing erlaubt die Konvertierung von Komponenten in neue Klassen, während die Properties unverändert bleiben. Einzelne Komponenten können schnell in andere eingebettet werden (beispielsweise in JPanel oder JScrollPane). Erwähnenswert ist weiterhin die bekannte Trennung zwischen dem UI-Layout und der eigentlichen Funktionalität sowie die Unterstützung für verschachtelte Forms (UI Forms können in andere gepackt werden). All dies kann schnell und intuitiv via Keyboard durchgeführt werden.

Wer Internationalisierung (I18N) bzw. Lokalisierung (L10N) bereits einmal „from scratch“ und umfassend selbst realisiert hat, weiß die IDEA-Unterstützung zu schätzen: Vom GUI-Designer aus können direkt Property Keys erstellt und verwaltet und Locales zur Design-Zeit umgeschaltet werden.

Der UI-Designer unterstützt eine Vielzahl neuer Komponenten wie JToolBar oder JProgressBar sowie weitere Funktio-

Anzeige

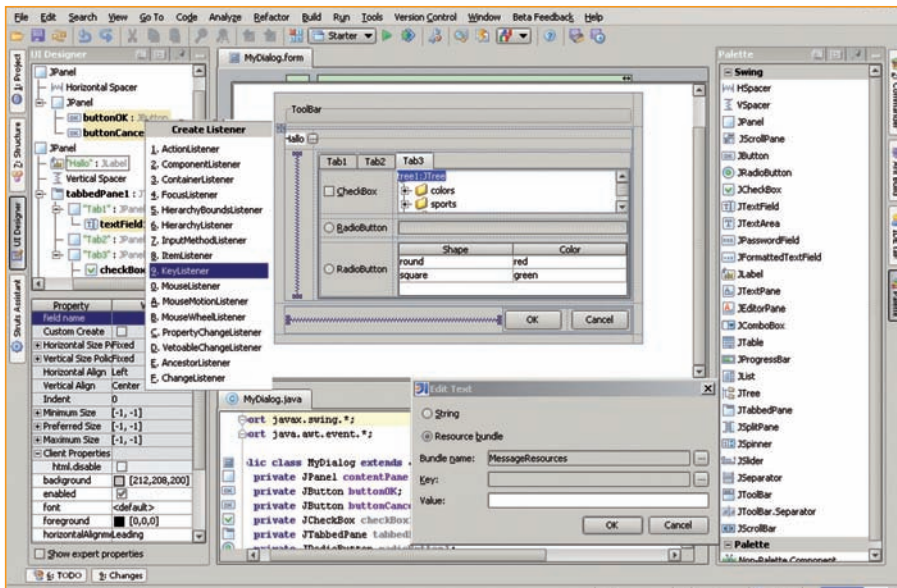


Abb. 3: UI-Designer Form und Class: Auflösung eines hart verdrahteten Strings in einen Resource Bundle; Auswahl und Zuweisung eines Listeners

nalitäten wie Button Groups und mehrere Border-Typen. Erwähnenswert ist außerdem die Migrationsunterstützung. Sowohl per Hand kodierte UIs als auch mit anderen UI-Designern erstellte Forms können importiert und zu JetBrains Forms konvertiert werden. Dazu werden Layout- und Komponenteneigenschaften ganzer Forms oder Teile davon „erbetet“.

Auch beim Design von UIs wird die Qualität des Codes nicht dem Zufall überlassen: So gibt es Inspections und Quick Fixes für vielerlei Szenarien: zum Beispiel fehlende Mnemonics, doppelte Mnemonics (sie können aber auch selbst eine intelligente Button-Traversierung implementieren), ein Radio-Button, welcher nicht in einer Group ist, eine Button Group mit nur

einem Button, eine Komponente ohne Label und eine scrollbare Komponente, die nicht in einer ScrollPane positioniert ist.

Weitere Unterstützung umfasst das Erstellen von Listeners (ein CTRL+O auf der selektierten Komponente genügt) und ein schnelles Navigieren zu diesen zwischen Form und Klasse sowie solche Gimmicks wie die Erstellung einer *main()*-Methode, um eine Form ausführbar zu machen.

Abbildung 3 zeigt einen komplett visuell erstellten OK-Cancel-Dialog. Im oberen Fenster ist die Form, im unteren die gebundene, generierte Java-Klasse zu sehen, in welcher anhand von Icons die an die Form gebundenen Komponenten und deren Typen erkennbar sind. Auch der Name orientiert sich an den Einstellungen

innerhalb der Form. Es wird gerade ein hart verdrahteter String in ein Resource Bundle aufgelöst. Ferner wurde in die Abbildung ein separater Arbeitsschritt, die Auswahl eines hinzuzufügenden Listeners, einmontiert.

Aber über UI-Designer lässt sich sicherlich streiten: Die einen mögen sie, die anderen nutzen sie generell nicht. Zumindest für die Erstellung von Prototypen können erste Entwürfe mit dem Designer nützlich sein. Die Unzahl von Refactorings und Hilfen sowie die umfassende Unterstützung von Komponenten und Layout-Managern mit Migrationsunterstützung sind gut gelungen – vorausgesetzt, Ihre UI besteht aus standardisierten, JavaBeans-konformen Komponenten. Und die Skeptiker, die lieber genau sehen möchte, was an Code automatisch generiert wird, können bei IDEA 6.0 einzelne Komponenten markieren und diese von der automatischen Generierung ausschließen bzw. manuell begleiten.

### Produktivitäts- und Editing-Features

Es wurden viele weitere Features ergänzt, die der Steigerung der Produktivität dienen sollen. Dazu gehören neue Intention Actions (Assistenten, die auf der gerade editierten Zeile dargestellt werden), eine Menge Refactorings und Code-Analyse-Features inklusive JavaDocs Checks mit Editor Highlighting. Dann projektweites Error Highlighting: Dateien mit Compilation Errors werden in dem Projekt und dem Package-Fenster gekennzeichnet, das heißt, Packages und das ganze Projekt werden als fehlerhaft markiert (Wellenlinie unter dem Namen), wenn eine Klasse innerhalb des Packages/des Projektes einen Fehler hat. Der Settings-Dialog ist durchsuchbar: Geben Sie einfach ein Suchwort in das Suchfeld ein. IDEA 6.0 unterstützt darauf hin weitere Compiler wie zum Beispiel auch den Eclipse Compiler.

Die bereits große Anzahl von über 600 Inspections wurde weiter erhöht und kann nun vom Entwickler mithilfe des Structural Search & Replace Template individuell erweitert werden. Inspections können auf einzelne Projekte oder Teile davon eingeschränkt werden. Mit der neuen Navigations-Bar steht nun eine Alternative zu der traditionellen Project

### JSR 305: „Annotations for Software Defect Detection“ und die JetBrains-Umsetzung

JSR 305: “This JSR will work to develop standard annotations (such as `@NonNull`) that can be applied to Java programs to assist tools that detect software defects.”

Die JetBrains-Umsetzung im Produktiveinsatz:

- `@Nls`: ein für den User sichtbarer, zu lokalisierender String, auswertbar durch L10N-Tools; anwendbar auf: Methoden, Felder, Parameter, Lokale Variablen, Typen, Packages
- `@Nonls`: ein nicht zu lokalisierender String, auswertbar durch L10N-Tools; anwendbar auf:

Methoden, Felder, Parameter, Lokale Variablen, Typen, Packages

- `@NotNull`: Element darf nicht null zurückliefern (bei Methoden), bekommen (bei Parametern) oder halten (bei lokalen Variablen und Feldern), auswertbar durch statische Analyse-Tools; anwendbar auf: Methoden, Felder, Parameter, lokale Variablen;
- `@Nullable`: Element darf null behandeln, auswertbar durch statischen Analyse Tools; anwendbar auf: Methoden, Felder, Parameter, lokale Variablen

View zur Verfügung. Zur Steigerung der Produktivität sollen auch die JetBrains-Erweiterungen des Type-Systems dienen.

### Qualitätserhöhung mit @Nullable und @NotNull: Extended Type System

Das Java-Type-System konnte in gewisser Weise schon immer durch Erstellung neuer Klassen erweitert werden. Allerdings ließen sich dadurch keine Compiler-Regeln verändern und keine weiteren Kompilierzeitchecks implementieren. Mit Java 5 gibt es nun die Annotations, die gewöhnlich für Code-Generierung und Spezifizierung von Runtime-Verhalten eingesetzt werden.

Ein Szenario, um durch den Einsatz von Annotations die Code-Qualität zu erhöhen, dient der Früherkennung von NullPointerExceptions mithilfe der JetBrains Annotations @NotNull und @Nullable [5].

Es ist Best Practice, immer auf null zu prüfen. Aber was passiert, wenn null von einer Methode geliefert wird, die es nicht liefern darf? Wo wir es nicht erwarten! Ein Assert ist keine optimale Lösung, denn ein Assertion Error ist nur wenig besser als eine NPE. Die Annotations können überprüfen, ob ein Methodenrumpf null zurückgibt, auch wenn jemand später am Code etwas ändert. Es ist auf einem Blick ersichtlich, ob die Methode null zurück-

geben darf oder nicht. Die Annotation ist hier eine praktische Metainformation, die dokumentiert, zur Laufzeit prüft und für Mehrwert sorgt.

In Abb. 4 sehen wir die *usingNullableMethod*, welche die mit der Annotation @Nullable markierte Methode *nullableMethod* nutzt. Der Compiler weiß nun, dass es hier eine NPE geben kann. Entsprechend wird ein Hinweis (Toolbar, farbliche Unterlegung) angeboten, damit der Methodenaufruf eine NPE produzieren kann. Das zweite, durch die Abbildung dokumentierte Szenario ist die Nutzung der @NotNull Annotation. Wenn zur Laufzeit der mit @NotNull annotierte Ausdruck zu null evaluiert wird, wird eine Exception geworfen. Dies trifft beim Starten der Klasse zu: Die Methode *splitName* wird mit dem Parameter null aufgerufen, was in dessen Signatur verboten ist. Ein Aufruf liefert folgende Exception:

```
com.intellij.rt.execution.application.AppMain com
    .huettermann.TestingNullable
Exception in thread "main" java.lang
    .IllegalArgumentException:
Argument 0 for @NotNull parameter of com/
    huettermann/TestingNullable.splitName must not be null
    at com.huettermann.TestingNullable.splitName
        (TestingNullable.java)
    at com.huettermann.TestingNullable
        .main(TestingNullable.java:17)
```

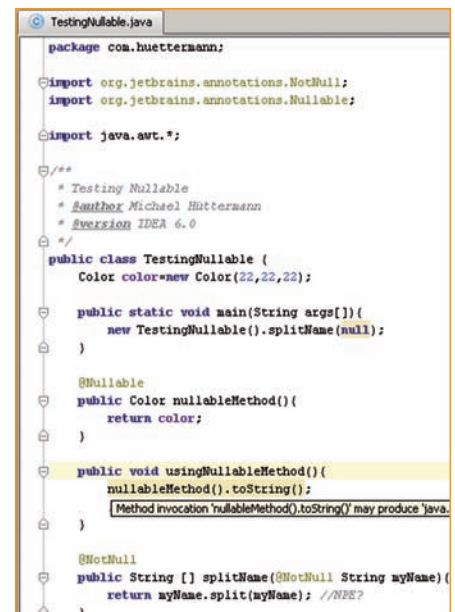


Abb. 4: Einsatz und Unterstützung der @Nullable- und @NotNull Annotations

Was ist zu machen werden, um von den Annotations zu profitieren? Dieses Feature ist kein Java-Standard, es kommt mit der im *redis*-Verzeichnis des JetBrains-Installationsordners liegenden *annotations.jar*, welche dem Klassenpfad hinzugefügt werden muss. Ferner muss dem Compiler gesagt werden, er möge diese Annotations auswerten. JetBrains arbeitet übrigens an dem JSR 305, der Entwicklung von Standard-Annotations [6], aktiv mit.

## Anzeige

Ein weiteres Szenario für den Einsatz von Annotations ist die Lokalisierung von Strings: Manche hart kodierten Strings sollen lokalisiert werden, andere nicht. IDEA bietet hier die Annotations `@Nls` und `@NonNls` (auf Nachfrage wurde mir bestätigt, dass NLS aus der Microsoft-Welt stammt und für „National Language Support“ steht). Einstellbar ist, dass für IDEA alle Strings lokalisiert werden müssen, bis auf die, die mit `@NonNls` annotiert sind [7]. `@Nls` ist die Default-Zuweisung, deren Nutzung lediglich der Dokumentation dient. Bei der Nutzung enthält der Code dann situativ Warnungen und Intention Actions. Nachteil kann sein, dass die Anwendung nach Java 5 entwickelt werden muss.

Auch in der Eclipse-Entwicklermannschaft hat dieses für Aufmerksamkeit gesorgt, es ist wohl normal – und das ist auch gut so –, dass man sich gegenseitig über die Schulter schaut. Dort gibt es Meinungen, dass das Feature allerdings nur aufgenommen werden sollte, wenn es zum Standard wird.

### Teamwork-Funktionalität – Collaboration

IDEA 6.0 umfasst eine TeamCity-Integration. TeamCity ist ein neues, Anfang Oktober mit IDEA 6.0 bereitgestelltes JetBrains-Produkt, welches via Plug-in in die IDE eingebunden wird und eigenständig verfügbar ist. Es unterstützt Continuous Integration, Build Management, serverseitige Code Coverage und Code-Analyse sowie ein Delayed Commit, um dem gefürchteten „5 o'clock check-in“ zu begegnen. Es lässt sich via Web-Interface konfigurieren und bietet Integration zu gängigen Build-Systemen, Unit-Testing-Frameworks und Versionskontrollsystemen.

Ferner existiert mit IDETalk eine Jabber-Integration. Der Instant Messenger unterstützt den Austausch von Nachrichten innerhalb der IDE mit Blick auf den Aktivitätsstatus anderer Entwickler, das User Group Management und eine durchsuchbare Message History. Mit einem einfachen Klick kann ein Code Pointer, also eine Referenz auf ein zu diskutierendes Code Snippet, ein Stacktrace oder ein Diff von gerade von Entwicklern

geöffneten Files an einen Peer übertragen werden. Dies ist nicht zu unterschätzen, muss zur Synchronisierung mit einem Kollegen doch nicht mehr der Umweg über das zentrale Versionierungssystem genommen werden (update).

Bei Versionskontrollsystemen gibt es ebenfalls zahlreiche Verbesserungen, wie zum Beispiel bessere Unterstützung von Visual SourceSafe, StarTeam und Subversion. Weiterer Benefit für die Teamarbeit: Run-Konfigurationen können nun in dem Projekt-File gespeichert und verteilt werden.

### Fazit

JetBrains legt mit IDEA 6.0 ein Release vor, welches vollgepackt ist mit neuen, nützlichen Funktionalitäten. Die Erweiterungen bauen Kernkompetenzen aus, wie beim UI-Designer und den Annotations. Bestehende Funktionalitäten wurden auf den neuesten Stand gebracht, wie bei der Application-Server-Unterstützung und den Java EE-Features.

JetBrains hat im kompletten Software Lifecycle aufgerüstet, angefangen beim UI-Design über die Lokalisierung der Anwendung hin zu einer breiten Unterstützung agiler Entwicklung durch Refactorings und der Dokumentation im DocBook-Format. Mit JUnit 4 und der Überprüfung auf Testabdeckung besteht eine charmante Unterstützung testgetriebener Entwicklung.

Auch wenn für die meisten Projekte IDEA alles mitbringt, was benötigt wird, baut JetBrains verstärkt auf ein Open API. Plug-in-Entwickler können nun auch die statische Code-Analyse erweitern, in dem sie selbst globale Inspections entwickeln [8]. Auch Unterstützung für injizierte Sprachen ist möglich (beispielsweise die Einbettung von SQL in Java-Sourcen). Die Arbeit an DSL (Domain Specific Language) ist bei JetBrains ein Dauerthema.

Über 150 Plug-ins sind verfügbar, einige davon haben eine doch etwas bescheidene Qualität, andere werden als Companion-Plug-ins von JetBrains explizit empfohlen und beworben [9] [10]. Über den Plug-in-Manager können Plug-ins deinstalliert oder neue hinzugefügt werden. JetBrains versucht, sehr nah an den Nutzern zu sein, um Wünsche und Opti-

mierungsbedarf frühzeitig identifizieren zu können [11].

Eine Neuanschaffung von IDEA 6.0 schlägt mit knapp 500 US-Dollar zu Buche. Es ist eine „Per Concurrent User“-Lizenz, kann also auf mehreren Maschinen gleichzeitig installiert sein. Ein Upgrade auf 6.0 kostet knapp 300 US-Dollar. Wer es neu erwirbt oder sein System aufrüstet, bekommt TeamCity 1.0 kostenlos dazu. Manche Personen halten IDEAs Start-up-Zeit für optimierbar (trotz Erhöhung des Heaps in der Datei `bin/idea.exe.vmoptions`).

Zurzeit gibt es bereits die Version 6.0.2, die diverse Speicherlöcher und Performance-Probleme beseitigt. Der Autor hat durch seine intensive Nutzung (Fehler werden direkt online zu JetBrains geschickt) und aktive Mitarbeit in der Tracking-Datenbank an der Optimierung mitgeholfen. Insgesamt macht das Release einen runden sowie stabilen Eindruck und der Support ist außerordentlich gut.

Manche Leute mögen vergeblich eine Unterstützung für Groovy, AOP, Velocity und Spring suchen. Es wird immer persönliche Präferenzen geben, JetBrains macht mit IDEA 6.0 seinen Anhängern sicher Freude und wird es den Kritikern weiter schwer machen.



**Michael Hüttermann** ist Senior Software Engineer und Enterprise Software Architect mit besonderem Interesse an agiler Softwareentwicklung. Er ist Autor von Fachartikeln, Sprecher auf Konferenzen, Aktivist der „Agile Cologne“ und „Agile Alliance“ und Organisator der Java User Group Köln.

### Links & Literatur

- [1] JetBrains IDEA: [www.jetbrains.com/idea/](http://www.jetbrains.com/idea/)
- [2] Readers' Choice 2006 des *Java Magazins*: [javamagazin.de/itr/service/psecom,id,300,nodeid,36.html](http://javamagazin.de/itr/service/psecom,id,300,nodeid,36.html)
- [3] JDJ Readers' Choice Awards: [jbj.sys-con.com/read/171303\\_3.htm](http://jbj.sys-con.com/read/171303_3.htm)
- [4] EMMA Code Coverage: [emma.sourceforge.net](http://emma.sourceforge.net)
- [5] @NotNull und @Nullable: [www.jetbrains.com/idea/documentation/howto.html](http://www.jetbrains.com/idea/documentation/howto.html)
- [6] JSR 305: [jcp.org/en/jsr/detail?id=305](http://jcp.org/en/jsr/detail?id=305)
- [7] @NonNls: [www.jetbrains.com/idea/features/i18n\\_support.html](http://www.jetbrains.com/idea/features/i18n_support.html)
- [8] Plug-in-Entwicklung: [www.jetbrains.com/idea/plugins/plugin\\_developers.html](http://www.jetbrains.com/idea/plugins/plugin_developers.html)
- [9] IDEA Plug-in Repository: [plugins.intellij.net](http://plugins.intellij.net)
- [10] IDEA Companions: [companions.jetbrains.com](http://companions.jetbrains.com)
- [11] IDEA Dashboard: [www.jetbrains.net](http://www.jetbrains.net)