Nr. 03 | Herbst 2012 | www.ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

Java in Höchstform

Java EE 6:
GlassFish, JBoss und Geronimo, Seite 11

Android:
Wissenschaftliche Applikationen der nächsten Generation entwickeln, Seite 38

Jnect:
Bewegungen in Java erkennen, Seite 59

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

iJUG Verbund

Sonderdruck

4 191978 304903 03

Java Forum Stuttgart

JUGS
java user group stuttgart

Die Java User Group Stuttgart e.V. veranstaltet am 5. Juli 2012 im Kultur- & Kongresszentrum Liederhalle (KKL) in Stuttgart wieder das Java Forum Stuttgart. Wie im Vorjahr werden rund 1.200 Teilnehmer erwartet. Geplant sind 42 Vorträge in sechs parallelen Tracks. Zudem werden bis zu 35 Aussteller vor Ort sein, darunter auch der Interessenverbund der Java User Groups e.V. (iJUG). An der Community-Wand stehen sowohl offene White-Boards als auch BoF-Boards (Bird-of-a-Feather). Abends gibt es die Gelegenheit, sich bei verschiedenen BoF-Sessions mit Gleichgesinnten zu treffen, um über ein bestimmtes Thema zu diskutieren und sich auszutauschen. Darüber hinaus wird es wieder eine Jobbörse/Karriereecke für die Besucher geben.

Workshop „Java für Entscheider"
Die eintägige Überblicksveranstaltung am Vortag (4. Juli 2012) zeigt Begrifflichkeiten und wichtige Technologien aus der seit Jahren in der Industrie etablierten Plattform Java. Ausgehend von strategischen Gesichtspunkten wie Bedeutung und Verbreitung reicht der Blick über das Client-seitige Java (Java SE) und die wesentlichen Entwicklungswerkzeuge bis zum Server-seitigen Java (Java EE). Dort stehen dann die Bedeutung von Java als Integrationsplattform und die verschiedenen Technologien im Mittelpunkt. Weiterhin wird noch der Einsatz von Java in den immer wichtiger werdenden mobilen Lösungen (Android, iOS) gestreift. Abschließend kommen noch das Ausrollen von Java-Lösungen und das sehr interessante Eclipse als Rich-Client zur Sprache, um dann den Bogen von der Software-Entwicklung hin zum Betrieb zu schlagen.

Experten-Forum Stuttgart
Am 6. Juli 2012 findet wieder im Anschluss an das Java Forum Stuttgart ein Experten-Forum Stuttgart statt. Auf dem Programm stehen zwölf halbtägige Workshops in sechs parallelen Tracks. Die Workshops in kleinen Gruppen mit maximal 25 Teilnehmern ermöglichen einen intensiven Austausch zwischen Trainer und Zuhörern.
Anmeldung und weitere Informationen zum Java Forum Stuttgart unter **www.java-forum-stuttgart.de**

10 Years PatternTesting – ein Rückblick

Oliver Böhm, T-Systems GmbH

Wir schreiben das Jahr 2002. Das Y2K-Problem ist bereits Geschichte, der Euro eingeführt und der Zusammenbruch des Neuen Marktes noch deutlich zu spüren. Man ist vorsichtiger geworden mit neuen Technologien, die die Welt verändern werden. In dieser Zeit machte sich eine kleine Gruppe um Vincent Massol (heute XWiki) und Matt Smith auf, eine der ersten AOP-Bibliotheken in die Welt zu setzen. Was aber verbirgt sich hinter Aspekt-orientierter Programmierung (AOP)?

Dieser Artikel gibt einen Einblick in die wunderbare Welt der Aspekt-Orientierung und die Arbeit an und mit PatternTesting. Was hat sich in den letzten zehn Jahren verändert, außer dass der AspectJ-Compiler inzwischen Eingang in das Eclipse- Öko-System gefunden hat? Sind Aspekte inzwischen in der Java-Welt angekommen? Oder ist AOP nur die Antwort auf die Frage, die keinen interessiert?

Es war einmal …

… eine Programmiersprache, die erfreute sich unter Kaffeetrinkern (und nicht nur dort) großer Beliebtheit. Aber es fehlte etwas. Man konnte zwar mit Objekten alle Dinge dieser Welt nachbilden, aber die Realisierung von Transaktionen, Autorisierung oder Logging ging durch alle Klassen und entzog sich meist erfolgreich der Kapselung. Wie schafft man es, diese „Querschnittsbelange" (auch als „crosscutting concerns" bezeichnet) herauszuziehen und in einer eigenen Klasse zu kapseln? Dies waren die Überlegungen, die bereits im letzten Jahrhundert zu einem neuen Paradigma führten – der Aspekt-orientierten Programmierung (AOP). AOP ist nichts komplett Neues, sondern baut auf der Objekt-Orientierung (oder auch der prozeduralen Programmierung) auf, bietet jedoch zusätzliche Sprachmittel an, um die bereits angesprochenen Querschnittsbelange zwar nicht in Klassen, aber in Aspekten zu kapseln.

Im Land der Aspekte

Um uns einen ersten Eindruck von der Aspekt-orientierten Vorgehensweise zu verschaffen, betrachten wir Listing 1 aus dem Bank-Bereich. Für eine reale Konto-Verwaltung sollte man jetzt zwar nicht unbedingt ein „double" verwenden (siehe http://haupz.blogspot.de/2009/01/ich-bin-reich.html), aber ansonsten kann man hier die eigentliche Business-Logik einer Konto-Verwaltung noch sehr gut nachvollziehen. Im Gegensatz zu diesen fachlichen Anforderungen (im AOP-Jargon auch als „Concern" bezeichnet) gibt es jede Menge nicht-fachlicher „Concerns" wie folgende Punkte, die sich mit OO-Mitteln schwer kapseln lassen:

- Protokollierung (Logging)
- Autorisierung
- Sicherheit
- Transaktionen

Betrachten wir dazu die Anforderung „Alle Kontobewegungen müssen protokolliert werden" (siehe Listing1).

```
public class Konto {

private double kontostand = 0.0;

public double abfragen() {
return kontostand;

}

public void einzahlen(double betrag) {
kontostand = kontostand + betrag;

}

public void abheben(double betrag) {
kontostand = kontostand - betrag;

}

public void ueberweisen(double betrag, Konto anderesKonto) {
this.abheben(betrag);
anderesKonto.einzahlen(betrag);
}

}
```

Listing 1

```
public class Konto {

private static Logger log = Logger.getLogger(Konto.class)
private double kontostand = 0.0;
...

public void einzahlen(double betrag) { kontostand =
kontostand + betrag;
log.info(„neuer Kontostand: „ + kontostand);
}

public void abheben(double betrag) { kontostand =
kontostand - betrag; log.info(„neuer Kontostand: „ +
kontostand);
}
...

}
```

Listing 2

```
public aspect LogAspect {

private static Logger log = Logger.getLogger(LogAspect.
class);

after(double neu) : set(double Konto.kontostand) &&
args(neu) {
log.info("neuer Kontostand:" + neu);
}

}
```

Listing 3

```
public void ueberweisen(double betrag, Konto anderes-
Konto) {
assert anderesKonto != null; this.abheben(betrag); ande-
resKonto.einzahlen(betrag);
}
```

Listing 4

```
public static User login(String name) throws LoginEx-
ception {
assert name != null;
…
assert user != null;
return new user;
}
```

Listing 5

```
public aspect NullPointerTrap {
after() returning(Object ret) : execution(public Ob-
ject+ *..*(..)) {
assert ret != null;
}
}
```

Listing 6

```
@MayReturnNull
public static User login(String name) {
…

}
```

Listing 7

```
after() returning(Object returned) :
execution(public Object+ *..*(..))
&& !execution(@MayReturnNull public Object+
*..*(..)) {
...
}
```

Listing 8

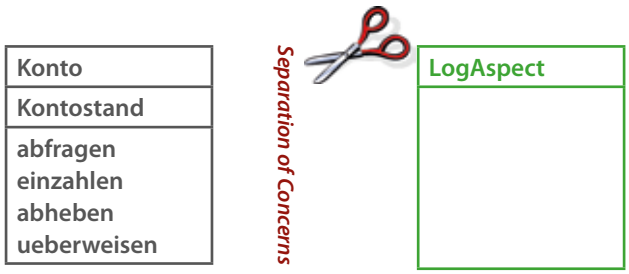

Abbildung 1: Separation of Concerns

Achtung: Code-Verschmutzung

In Java würde man dieses Problem vielleicht wie in Listing 2 lösen (die neu hinzugekommenen Anweisungen sind kursiv gekennzeichnet).

Die Business-Logik ist schon etwas schwieriger zu erkennen. Wenn weitere Anforderungen wie Transaktionssicherheit oder Autorisierung hinzukommen, wird es sehr schnell unübersichtlich.

Rettung naht

Wie würde man die obigen Anforderungen in der Aspekt-Orientierung ausdrücken? In etwa so: Füge an allen Stellen, an denen das Attribut „kontostand" geändert wird, eine Log-Anweisung hinzu. Mit AspectJ (einer Aspekt-orientierten Erweiterung von Java) sähe die Realisierung dieser Anforderung ungefähr so aus (siehe Listing 3).

„aspect" ist der Rahmen um das Ganze, in dem die eigentliche Anforderung implementiert wird. Die tatsächliche Anweisung ist kursiv hervorgehoben und entspricht in etwa dem, was im obigen Kasten als Pseudo-Anweisung geschrieben steht. Damit hat man es geschafft, diese neue Anforderung der Protokollierung in einem eigenen Aspekt zu kapseln (siehe Abbildung 1).

Außer beim Zugriff auf Klassen-Attribute kann man noch bei folgenden Punkten im Programm eingreifen:

- Aufruf einer Methode
- Ausführen einer Methode
- Behandeln einer Exception
- Initialisierung einer Klasse
- Initialisierung eines Objekts

Diese Punkte heißen im AOP-Jargon auch „Joinpoints". Über „Advices" können diese Stellen dann um weitere Funktionalitäten ergänzt werden. Advices sind so etwas wie die Methoden der Aspekt-Orientierung – im obigen Beispiel war es die (einzelne) Log-Anweisung.

Damit haben wir alle Zutaten beisammen, um die Anforderungen auf Klassen (wie bisher) oder Aspekte aufzuteilen und dem Ziel, der „Separation of Concerns", ein

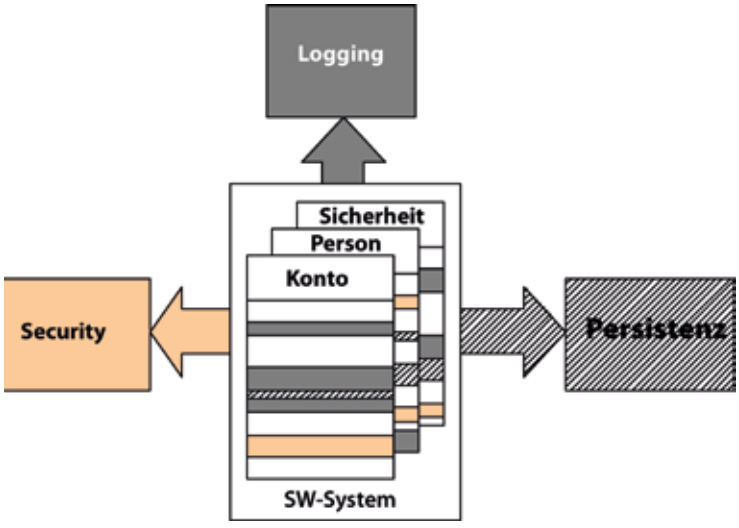

Abbildung 2: Das System als Menge von „Concerns"

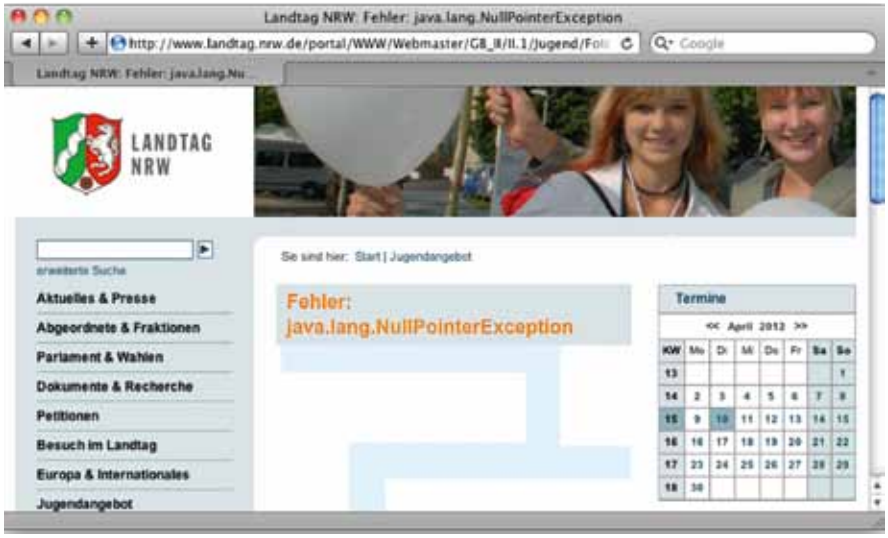

Abbildung 3: Das Jugendangebot des Landtags von NRW (Stand: 10. April 2012)

gutes Stück näher zu kommen. Und das Schönste daran ist, dass wir bei der Entwicklung nicht immer an alles auf einmal denken müssen, sondern uns jetzt immer nur auf einen „Concern" konzentrieren können (siehe Abbildung 2).

Neues Altes aus der Java-Welt

Verlassen wir kurz die Welt der Aspekte und wenden wir uns wieder der Java-Welt zu. 2002 erschien Java 1.4, das als Spracherweiterung „asserts" mitbrachte. Damit kann man Sicherungen im Code einbauen, um die Fehlersuche zu beschleunigen (siehe Listing 4).

Aktiviert werden „asserts" über den Schalter „-ea" (enable assertions) beim Start der Java-VM. Übergibt man dann während der Test-Phase (in der man üblicherweise diesen Schalter aktiviert) einen Null-Parameter an diese Methode, gibt die assert-Anweisung eine Warnmeldung aus, dass die zugesicherte Bedingung verletzt wurde, und das Programm wird beendet. Dadurch kommt man der Fehlerursache sehr viel früher und schneller auf die Spur. Da sich dieses Test-Hilfsmittel noch nicht bei allen Entwicklern herumgesprochen hat, nachfolgend ein Vorschlag, wie man damit NullPointerExceptions fangen kann.

Wie man NullPointerExceptions fängt

NullPointerExceptions sind mit die ärgerlichsten Fehler, die anzutreffen sind (siehe Abbildung 3). Bei der Eingabe von „NullPointerException filetype:jsp" als Suchbegriff in Google erhält man alle JSP-Seiten, in denen „NullPointerException" auftaucht (darunter auch einige Foren – die man getrost übergehen kann). Dabei ist es eigentlich relativ einfach, NullPointerExceptions zu vermeiden – man lässt einfach keine Null-Werte zu, weder als Argument, noch als Rückgabewert (siehe Listing 5).

Wenn jetzt während der Testphase (und umgelegtem „-ea"-Schalter) doch ein null-Argument übergeben wird, ist der Schuldige schnell gefunden. Auch wenn man damit NullPointerExceptions drastisch reduzieren kann, ist es ziemlich öde, diese assert-Anweisungen manuell am Anfang und Ende jeder Methode einzufügen. Hier kommt wieder die Aspekt-Orientierung ins Spiel: Sie erlaubt uns, diese „Querschnitts"-Anforderung zu formulieren (siehe Listing 6). Damit wird der zweite Teil der obigen Anforderung realisiert: Am Ende jeder Methode mit Rückgabewert eine assert-Anweisung einfügen.

Eine der Stärken von AOP ist, dass man über Wildcards mehrere Stellen (im AOP-Jargon auch als Joinpoint bezeichnet) im Programm ansprechen kann, um wie hier zusätzliche Überprüfungen einzufügen. Um auf die Eingangsfrage zurückzukommen („Wie fängt man NullPointerExceptions?"): Gar nicht! Man vermeidet einfach „null" als gültigen Wert, indem man sich an folgende Dinge hält:

- Mit Exceptions wie mit einer FinderException arbeiten, um „kein Ergebnis" oder andere Ausnahme-Situationen zu signalisieren
- Objekte mit einer Null-Semantik (wie „Collections.EMPTY_LIST") einführen, die als Ersatz für „null" als Parameter oder Rückgabewert dienen können

Die Anfänge von PatternTesting

Die gedankenlose Verwendung von null-Werten ist nur eine von vielen Ursachen für Fehler im ausgelieferten Programm. Weitere, immer wieder gern gemachte Fehler sind fehlende Freigaben von Ressourcen, Deadlock-Situationen, vergessene Exception-Handler (mit generierter „e.printStackTrace()"-Anweisung) und viele andere mehr. Einige dieser Kandidaten lassen sich durch statische Code-Analyse und Tools wie FindBugs oder PMD aufspüren; für andere Fehlerquellen wie den sorglosen Umgang von null als Parameter oder Rückgabewert müsste man in den Code eingreifen, um sie aufzudecken.

Wie wir aber bereits gesehen haben, ist dies mit der Aspekt-Orientierung möglich und führte dazu, dass Vincent Massol im März 2002 mit PatternTesting eine der ersten AOP-Bibliotheken aus der Taufe hob. Anfangs war PatternTesting mehr ein „Proof of Concept", um Architektur- und Implementierungs-Entscheidungen automatisch überprüfen zu können. Es war als Sammlung von Aspekten angelegt, die man erweitern und spezifizieren konnte, wie weit diese Prüfungen reichen sollte (sofern man sich mit AspectJ auskannte). Es gab einen AbstractNullTest, der auf „null" als Argument oder Rückgabewert hinwies, einen AbstractSop-Aspekt, der bei „System.out.println(..)"-Anweisungen Compiler-Warnungen ausgab oder einen AbstractDatabaseTest, mit dem man beim Einsatz von Hibernate oder anderer Persistenz-Frameworks Aufrufe aus java.sql unterbinden konnte.

AspectJ 1.0 und 1.1

In der Anfangsphase arbeitete der AspectJ-Compiler noch als eine Art Präprozessor, der die Aspekte in Java-Code umwandelte, bevor er sie an den Java-Compiler weiterreichte. Entsprechend war die erste Version (v0.2) von PatternTesting auch ein Proof of Concept für den Einsatz des AspectJ-Compilers, zumal auch die Tool- Un-

GLOSSAR

Concern
Spezifische Anforderung oder Gesichtspunkt, welche(r) in einem Software-System behandelt werden muss, um die übergreifenden Systemziele zu erreichen (nach Ramnivas Laddad, 2003)

Querschnittsbelang (Crosscutting Concern)
Dies sind Anforderungen und Dinge wie Transaktionen, Autorisierung oder Security, die sich durch alle Klassen ziehen und sich mit Mitteln der Objekt-Orientierung nur schwer kapseln lassen.

Joinpoint
Ein Punkt im Programm, an dem man einen –> Advice zur Ausführung bringen kann. Joinpoints sind zum Beispiel der Aufruf oder die Ausführung von Methoden, der Zugriff auf Attribute oder die Initialisierung von Objekten.

Pointcut
Pointcuts sind ein AOP-Sprachmittel und dienen dazu, einzelne oder mehrere Joinpoints zusammenzufassen.

Advice
Der Code, der an den –> Pointcuts ausgeführt wird.

Compile-Time-Weaving (CTW)
Beim Compile-Time-Weaving wird vom Compiler aus den Java-Klassen und Aspekten direkt während des Compile-Vorgangs Code erzeugt.

Load-Time-Weaving (LTW)
Beim Load-Time-Weaving wird über einen Weaving-Agent während des Lade-Vorgangs der Byte-Code der geladenen Klassen erweitert.

Avalon-Framework
Ein ehemaliges Komponenten-Framework für serverseitige Container, das inzwischen eingestellt wurde.

terstützung für die Entwicklung noch in den Kinderschuhen steckte.

Dies änderte sich mit AspectJ 1.1, das 2003 auf dem Markt erschien. Damit wandelte sich die Architektur grundlegend. Der AspectJ-Compiler setzte jetzt auf dem Java-Compiler aus Eclipse auf und konnte so direkt Bytecode erzeugen. Auch die inkrementelle Kompilierung hielt damit Einzug in den AspectJ-Compiler und sorgte zusammen mit AJDT, dem AspectJ-Development-Tool-Plug-in für Eclipse, dafür, dass die IDE- Unterstützung immer besser wurde. Mit PatternTesting 0.3 wurde 2004 auf AspectJ 1.1 umgestellt und Unterstützung für das Avalon-Framework hinzugefügt. Gleichzeitig übernahm Matt Smith die Projekt- Leitung.

Java 5 und AspectJ 5

Mit JDK 1.5 (das von Sun als Java 5 verkauft wurde) hielten 2005 einige Neuerungen Einzug in die Sprache: Generics, Autoboxing oder Annotations. Auch bei der AOP- Unterstützung für Java tat sich einiges: AspectJ und AspectWerkz fusionierten. AspectWerkz war neben AspectJ ebenfalls eine Aspekt-orientierte Erweiterung für Java, webte aber die Aspekte beim Laden einer Klasse ein (LTW: Load Time Weaving). Dies ist der Grund dafür, dass AspectJ heute neben dem Compile Time Weaving (CTW) auch diesen Weg unterstützt, um die Aspekte in den Code zu bekommen. Beide Verfahren haben Vor- und Nachteile, sodass es durchaus sinnvoll ist, sie zu unterstützen:

- Beim Compile Time Weaving (CTW) werden Laufzeit-Fehler vermieden, da die Aspekte bereits durch den Compiler überprüft und übersetzt wurden.
- Mit Load Time Weaving (LTW) können auch Bibliotheken von Dritt-Anbietern instrumentiert („kompiliert“) werden. Prinzipiell ist dies auch mit CTW möglich, allerdings verbieten manche Lizenz-Bestimmungen die Veränderung von Jar- Dateien.

Gleichzeitig passte man sich mit der Nummerierung der Java-Versionierung an. Nach AspectJ 1.2 (das sich von AspectJ 1.1 nur durch eine Verbesserung der Tool-Kette unterschied) kam AspectJ 5, das sich (analog zu Java) intern als Version 1.5 zu erkennen gab. Wichtigste Neuerung war neben der Fusion mit AspectWerkz und der Unterstützung von Load Time Weaving die Verwendung und Unterstützung von Annotations – sie können jetzt auch zur Auswahl von Joinpoints herangezogen werden.

Nehmen wir an, wir wollen für die Login-Methode „null“ als gültigen Rückgabewert zulassen, dann könnten wir dies durch eine @MayReturnNull-Annotation anzeigen (siehe Listing 7). Listing 8 zeigt eine Annotation, die von PatternTesting verwendet wird, um diese Methoden von der Überprüfung auszuschließen.

Während man in PatternTesting 0.3 noch AspectJ-Kenntnisse benötigte, um die bereitgestellten (abstrakten) Aspekte in ein Projekt einzubinden, kann ab PatternTesting 0.5 (das 2008 nach vierjähriger Ruhezeit erschien) diese Bibliothek auch ohne solche Kenntnisse eingesetzt werden, da alle Aspekte über Annotationen gesteuert werden können. Nach wie vor ist aber auch die Erweiterung bestehender Aspekte vorgesehen und möglich. Als weiterer Vorteil haben die verwendeten Annotationen auch Dokumentations-Charakter. So zeigt die „@MayReturnNull“-Annotation dem Entwickler an, dass die Login-Methode auch „null“ als Rückgabe-Wert liefern kann.

Von Maven 1 nach Maven 2

2008 war ein sehr bewegtes Jahr in der Geschichte von PatternTesting. Der Autor hatte ein Jahr zuvor Kontakt zu Vincent Massol und Matt Smith und ihnen unter anderem seine Hilfe bei PatternTesting angeboten, da seit ein paar Jahren keine Weiterentwicklung mehr zu erkennen war. Ehe er sich versah, hatte er die Projekt-Leitung inne, da beide mit anderen Projekten beschäftigt waren.

Anfangs befasste der Autor sich noch hauptsächlich damit, den Build und die Projekt-Struktur von Maven 1 auf Maven 2 umzustellen. Dabei leistete ihm das „aspectj-maven“-Plug-in wertvolle Dienste bei der Kompilierung. Als Infrastruktur für den Build wurde anfangs Continuum von Apache eingesetzt, inzwischen läuft der Build mit Jenkins auf einem ausgedienten Mac-Mini (PowerPC, Ubuntu). Während für die sonstige Infrastruktur (CVS, Wiki) Sourceforge.net sehr nützlich ist, sieht es im Bereich der öffentlichen Build-Server noch mau aus. Einzig CloudBees.com scheint

hier Angebote für Open-Source-Projekte zu unterbreiten, deren kostenlose Ressourcen sich aber als nicht ausreichend für PatternTesting erwiesen haben. Die nächsten Schritte waren dann die Umstellung auf AspectJ 5, die Verwendung von Annotationen sowie die Weiterentwicklung der vorhandenen Aspekte.

Performance-Probleme bis AJDT 1.6.1

Mit dem Anwachsen der Code-Basis und der Entwicklung neuer Aspekte litten vor allem größere Projekte unter der fehlende Performance des AJDT-Plug-ins in Eclipse 3.3. Dies änderte sich erst mit Eclipse 3.4 und AJDT 1.6.2. Vor allem die Möglichkeit, eigene Warnungen und Fehler schon während der Compile-Phase ausgeben zu lassen, erhöhten die Wartezeiten nach dem Speichern der Sourcen, wenn die automatische Kompilierung angestoßen wurde. Dies führte schließlich dazu, dass PatternTesting aufgeteilt wurde in:

- PatternTesting Runtime (die Basis für alle anderen Unter-Projekte)
- PatternTesting Check-CT (Compile-Time-Checks)
- PatternTesting Check-RT (Runtime-Checks)

Dadurch war man jetzt in der Lage, während der Arbeit mit Eclipse die Compile-Time- Checks wegzulassen und sie nur während des Daily Builds zu aktivieren. Weiterer Vorteil dieser Trennung war, dass man auch die Runtime-Checks weglassen konnte (z.B. nach der Test-Phase, wenn die Anwendung ausgeliefert werden soll), ohne dass der Source-Code geändert werden muss. Lediglich PatternTesting Runtime muss mit ausgeliefert werden, da dort die ganzen Annotationen enthalten sind.

Ein Jahr später kam dann mit PatternTesting Exception noch ein weiteres Projekt für die Unterstützung aussagekräftigerer Exceptions hinzu. Wenn etwa ein Socket mit „Socket socket = new Socket(„10.11.12.13“, 80);“ geöffnet wird und der angegebene Rechner (10.11.12.13) nicht erreichbar ist, bekommt man eine ConnectException mit der lapidaren Meldung „Connection refused“. Wäre es nicht sinnvoller, wenn stattdessen „Connection to 10.11.12.13 refused“ ausgegeben werden würde? Dies ist genau das Einsatz-

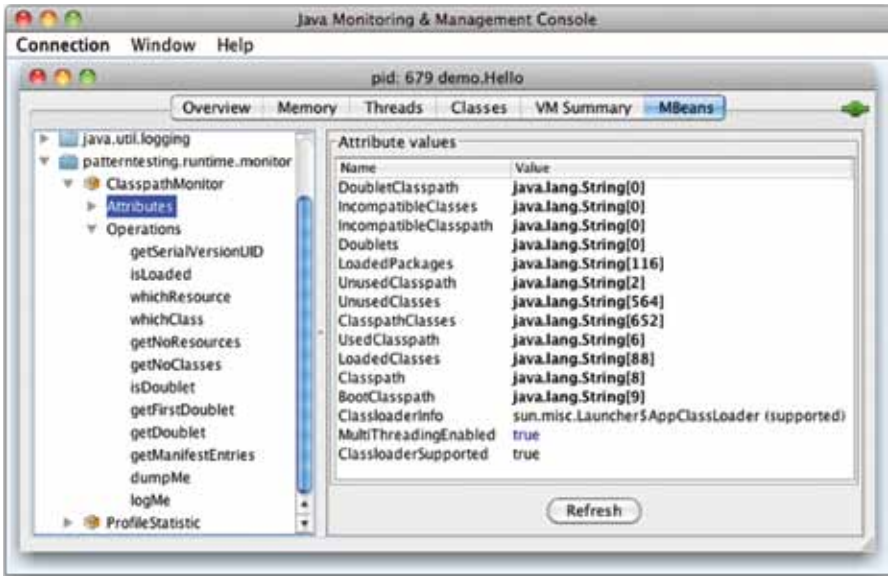

Abbildung 4: Der ClasspathMonitor innerhalb der JConsole

Szenario von PatternTesting Exception. Einige Exceptions werden um zusätzliche Informationen angereichert, um die Fehlersuche zu erleichtern. Und man erhält darüber hinaus Unterstützung für das Testen der Exception-Handler.

PatternTesting 2010

2010 war es dann endlich soweit: Die Version 1.0 wurde am 20. Juni 2010 ausgeliefert. Gleichzeitig gab es mit http://patterntesting.org eine neue Heimat, die von der Agentes GmbH gesponsert wird. Damit änderten sich auch die Maven- Koordinaten für die aktuelle Version von PatternTesting Runtime (siehe Listing 9). Seit Version 1.0.3 ist PatternTesting auch wieder im zentralen Maven-Repository zu finden, nachdem es zwischenzeitlich durch diverse Umstellungen dort längere Zeit nie angekommen war.

JUnit-Unterstützung

Mit 1.0 gab es auch einen eigenen JUnit-Runner, um bestehende Tests bei Bedarf ausblenden oder als „kaputt“ kennzeichnen zu können (siehe Listing 10).

Mit dem SmokeRunner können Tests als „Broken“ (mit Fix-Datum) oder als Integrations-Tests (die während dem Entwickler-Test ausgeblendet werden) gekennzeichnet werden. Auch können Tests an bestimmte Plattformen oder Bedingungen verknüpft werden, um ausgeführt zu werden.

Der kleine Bruder des SmokeRunners ist der ParallelRunner (aus PatternTesting Concurrent), der die gleiche Funktionalität besitzt, aber zur Beschleunigung der Tests alle Test-Methoden parallel ausführt. Eine weitere Beschleunigung ermöglicht die ParallelSuite (ab 1.2), die alle in einer Suite zusammengefassten Klassen parallel startet.

Durch die Abhängigkeit von PatternTesting Runtime zu JUnit 4.8 laufen diese Runner leider nicht in älteren Eclipse-Versionen. Dies wird sich aber mit der nächsten Version ändern – hier werden auch ältere Versionen bis Eclipse 3.4 (das z.B. Basis für IBMs RAD 7.5 ist) unterstützt werden.

```
<groupId>org.patterntesting</groupId>
<artifactId>patterntesting-rt</artifactId>
<version>1.2.10-YEARS</version>
```

Listing 9

```
@RunWith(SmokeRunner.class)
public class Rot13Test {
@Broken(till = "01-Jun-2012")
@Test
public final void testCryptBobsFile() throws
IOException { ...
}
@IntegrationTest("online access needed")
@Test
public final void testCryptURI() throws IOException {
...
}
}
```

Listing 10

Rückblick und Ausblick

Was haben uns die letzten zehn Jahre im Java-Umfeld gebracht? Der größte Sprung war sicherlich der von Java 1.4 auf Java 5, der uns neue Sprach-Features bescherte. Bei Java 6 lag der Fokus auf dem Desktop, Java 7 ließ fast 5 Jahre auf sich warten, nachdem viele geplante Features auf Java 8 verschoben wurden.

Bei AspectJ sind vor allem AspectJ 5 und die Vereinigung mit AspectWerkz hervorzuheben. Danach hat sich am Sprach-Kern nichts geändert, lediglich die Performance und die IDE-Unterstützung wurden stetig verbessert. Die Versionsnummer wurde dabei an Java angepasst – aktuell ist AspectJ 1.7 (oder kurz: AspectJ 7).

Der große Durchbruch für AspectJ lässt noch auf sich warten. Aber AOP-Techniken haben sich in der Zwischenzeit ihren Platz in der Java-Welt erobert, um zusätzliche Anforderungen im Code unterzubringen, ohne bestehende Klassen zu verändern. Manche Frameworks agieren dazu als Java-Agent oder als Proxy (zum Beispiel einige Mock-Frameworks), andere setzen dazu einen eigenen Classloader ein (etwa JBoss-AOP).

PatternTesting hat sich von einer reinen AspectJ-Bibliothek zu einem Hybrid entwickelt. Die Runtime-Komponente kann auch als reine Java-Bibliothek eingesetzt werden und man erhält neben dem SmokeRunner noch einige Tester, um beispielsweise „equals(..)"- und „hashCode()"-Implementierungen zu überprüfen, die Serializable-Eigenschaft zu testen oder Dateien auf Gleichheit zu überprüfen. Ein ClasspathMonitor gibt Aufschluss über den Classpath und findet unter anderem doppelte und inkompatible Klassen (siehe Abbildung 4).

Leider funktioniert der ClasspathMonitor nur mit der Sun-VM korrekt. Für die Java-VM von IBM ist jedoch Besserung in Sicht: Hier wird es einen PatternTesting Agent geben, der die notwendigen Informationen für den ClasspathMonitor sammelt. Was bleiben wird, ist die Unterstützung für Java 5, da es noch einige Projekte gibt, die nicht auf ein aktuelleres JDK umschwenken können oder dürfen. Dazu passt auch die Unterstützung älterer JUnit-Versionen, um PatternTesting auch mit älteren Eclipse-Versionen einsetzen zu können. Ferner ist die Unterstützung weiterer Exceptions (wie der SQLException) geplant, um im Falle eines Falles aussagekräftigere Fehlermeldungen zu bekommen. Wer jetzt Lust auf PatternTesting bekommen hat, findet im PatternTesting-Wiki unter „Getting Started" (siehe http://sourceforge.net/apps/mediawiki/patterntesting/index.php? title=Getting_Started) ein einfaches Hello-World-Beispiel sowie mit „Testing with PatternTesting" eine Anleitung, wie man den SmokeRunner und die verschiedenen Tester einsetzen kann.

Links

1. PatternTesting: http://patterntesting.org/
2. PatternTesting Wiki: http://sourceforge.net/apps/mediawiki/patterntesting/ AspectJ-Homepage: http://eclipse.org/aspectj/
3. Performance Boost with Eclipse 3.4: http://oli.blogger.de/20090323/

Oliver Böhm

oliver.boehm@t-systems.com

Oliver Böhm beschäftigt sich mit Java-Entwicklung unter Linux und Aspekt-Orientierter SW-Entwicklung. Neben seiner hauptberuflichen Tätigkeit als JEE-Architekt bei T-Systems ist er Buchautor, Projektleiter bei PatternTesting und Board-Mitglied der Java User Group Stuttgart.

„Von den Erfahrungen der anderen zu profitieren, ist essentiell …"

Usergroups bieten vielfältige Möglichkeiten zum Erfahrungsaustausch und zur Wissensvermittlung unter den Java-Entwicklern. Sie sind aber auch ein wichtiges Sprachrohr in der Community und gegenüber Oracle. Wolfgang Taschner, Chefredakteur der Java aktuell, sprach darüber mit Tony Fräfel, dem Vorsitzenden der Swiss Oracle User Group (SOUG).

Wie bist du zur SOUG gekommen?

Fräfel: 1987 hat sich mein damaliger Arbeitgeber für das rDBMS Oracle entschieden. Die Oracle-Kundschaft in der Schweiz war damals noch überschaubar und ein Erfahrungsaustausch sehr willkommen. In der SOUG fanden wir erfahrene Oracle-Spezialisten und ich durfte meine Firma dort vertreten.

Wie ist die SOUG organisiert?

Fräfel: Die SOUG ist ein unabhängiger Verein. Ein siebenköpfiger Vorstand ist verantwortlich für die Aktivitäten. Er wird unterstützt von einem Komitee, das für die Aktivitäten in der französischsprechenden Schweiz verantwortlich ist, sowie von unserem Sekretariat.

Was zeichnet die SOUG aus?

Fräfel: Im Fokus steht klar der Erfahrungsaustausch. Unsere Mitglieder profitieren von den praktischen Erfahrungen, die andere Mitglieder gemacht haben. Wir ermöglichen es, Kontakte auf nationaler und internationaler Ebene zu knüpfen und zu pflegen. Wichtig ist uns auch unsere Unabhängigkeit – wir vertreten die Interessen unserer Mitglieder, nicht diejenigen von Herstellern.

Wie viele Veranstaltungen gibt es pro Jahr?

Fräfel: Zurzeit sind es circa vier Special-Interest-Group-Meetings pro Jahr in der

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG *News* und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.