
KAFFEEKLATSCH

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

12/2011



KAFFEEKLATSCH

— Das Magazin rund um Software-Entwicklung —

Sie können die elektronische Form des KAFFEEKLATSCHS
monatlich, kostenlos und unverbindlich
durch eine E-Mail an

abo@bookware.de

abonnieren.

Ihre E-Mail-Adresse wird ausschließlich für den Versand
des KAFFEEKLATSCHS verwendet.

Editorial

Germany's Top Programmer

Nachdem die Suche im Deutschen Fernsehen nach Models, Talenten und Stimmen mehr oder weniger erfolgreich abläuft,

drängt sich die Frage auf, ob es nicht Zeit wird, auch endlich einmal nach dem besten Software-Entwickler zu suchen. *Germany's Next Top Programmer* oder *Deutschland sucht den Programmier-Star*, was zeitgemäß und -geistig mit GNTP oder DSDP abgekürzt werden könnte.

So wäre es vorstellbar, dass die Teilnehmer klassische Programmierprobleme interpretieren und implementieren müssen: Suchen, Sortieren, Strukturieren und Berechnen. Den Programmierer bei der Arbeit zu beobachten, wäre dabei wohl nicht ganz so ergiebig, wenngleich man Noten für die Ordnung auf dem Schreibtisch und die Haltung vor dem Rechner vergeben könnte. Abzüge bei der B-Note gäbe es dann allerdings, wenn man mit seinem Rechner spricht; disqualifizierend wäre es, wenn man ihn anspricht oder gar schlägt.

Pluspunkte könnte es für Vorarbeiten und deren Qualität geben. Ein souveränes *Vom-Rechner-ins-Hirn*-Vorgehen könnte zwar beeindruckend wirken, aber leider auch als Nachlässigkeit bewertet werden. Da käme es natürlich darauf an, wie der Programmierer dieses Vorgehen verkauft: Auch die Persönlichkeit zählt.

Damit die Jury nicht von Äußerlichkeiten abgelenkt wird, könnte man sich alternativ ausschließlich mit dem Code beschäftigen. Die Jury säße vor Monitoren, an denen langsam Code-Fragmente vorbeiscrollen. Bewertet würde dann vorrangig die Funktionalität, aber auch das Format, die Wahl der Farben für Schlüsselwörter etc., die Quantität und Qualität der Kommentare und natürlich der Stil würden mit in die Benotung einfließen.

Und was winkt dem Gewinner? Ewiger Ruhm und ein exklusiver Arbeitsvertrag bei einem Unternehmen, das nur tolle Sachen entwickelt. Und am Arbeitsplatz steht dem Gewinner ein 64-Core-Rechner mit acht Monitoren zur Verfügung, der mindestens so cool ist wie der im Film *Password: Swordfish*.

Ganz unabhängig davon wünsche ich allen KAFFEEKLATSCH-Lesern ein frohes Weihnachtsfest und einen geruhsamen Übergang ins nächste Jahr, welches für jeden das mit sich bringen möge, was er am dringendsten braucht.

Ihr MICHAEL WIEDEKING
Herausgeber

Beitragsinformation

Der KAFFEEKLATSCH dient Entwicklern, Architekten, Projektleitern und Entscheidern als Kommunikationsplattform. Er soll neben dem Know-how-Transfer von Technologien (insbesondere Java und .NET) auch auf einfache Weise die Publikation von Projekt- und Erfahrungsberichten ermöglichen.

Beiträge

Um einen Beitrag im KAFFEEKLATSCH veröffentlichen zu können, müssen Sie prüfen, ob Ihr Beitrag den folgenden Mindestanforderungen genügt:

- Ist das Thema von Interesse für Entwickler, Architekten, Projektleiter oder Entscheider, speziell wenn sich diese mit der Java- oder .NET-Technologie beschäftigen?
- Ist der Artikel für diese Zielgruppe bei der Arbeit mit Java oder .NET relevant oder hilfreich?
- Genügt die Arbeit den üblichen professionellen Standards für Artikel in Bezug auf Sprache und Erscheinungsbild?

Wenn Sie uns einen solchen Artikel, um ihn in diesem Medium zu veröffentlichen, zukommen lassen, dann übertragen Sie Bookware unwiderruflich das nicht exklusive, weltweit geltende Recht

- diesen Artikel bei Annahme durch die Redaktion im KAFFEEKLATSCH zu veröffentlichen
- diesen Artikel nach Belieben in elektronischer oder gedruckter Form zu verbreiten
- diesen Artikel in der Bookware-Bibliothek zu veröffentlichen
- den Nutzern zu erlauben diesen Artikel für nicht-kommerzielle Zwecke, insbesondere für Weiterbildung und Forschung, zu kopieren und zu verteilen.

Wir möchten deshalb keine Artikel veröffentlichen, die bereits in anderen Print- oder Online-Medien veröffentlicht worden sind.

Selbstverständlich bleibt das Copyright auch bei Ihnen und Bookware wird jede Anfrage für eine kommerzielle Nutzung direkt an Sie weiterleiten.

Die Beiträge sollten in elektronischer Form via E-Mail an redaktion@bookware.de geschickt werden.

Auf Wunsch stellen wir dem Autor seinen Artikel als unveränderlichen PDF-Nachdruck in der kanonischen KAFFEEKLATSCH-Form zur Verfügung, für den er ein unwiderrufliches, nicht-exklusives Nutzungsrecht erhält.

Leserbriefe

Leserbriefe werden nur dann akzeptiert, wenn sie mit vollständigem Namen, Anschrift und E-Mail-Adresse versehen sind. Die Redaktion behält sich vor, Leserbriefe – auch gekürzt – zu veröffentlichen, wenn dem nicht explizit widersprochen wurde.

Sobald ein Leserbrief (oder auch Artikel) als direkte Kritik zu einem bereits veröffentlichten Beitrag aufgefasst werden kann, behält sich die Redaktion vor, die Veröffentlichung jener Beiträge zu verzögern, so dass der Kritisierte die Möglichkeit hat, auf die Kritik in der selben Ausgabe zu reagieren.

Leserbriefe schicken Sie bitte an leserbrief@bookware.de. Für Fragen und Wünsche zu Nachdrucken, Kopien von Berichten oder Referenzen wenden Sie sich bitte direkt an die Autoren.

Werbung ist Information

Firmen haben die Möglichkeit Werbung im KAFFEEKLATSCH unterzubringen. Der Werbeteil ist in drei Teile gegliedert:

- Stellenanzeigen
- Seminaranzeigen
- Produktinformation und -werbung

Die Werbeflächen werden als Vielfaches von Sechsteln und Vierteln einer DIN-A4-Seite zur Verfügung gestellt.

Der Werbeplatz kann bei Frau NATALIA WILHELM via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-16 gebucht werden.

Abonnement

Der KAFFEEKLATSCH erscheint zur Zeit monatlich. Die jeweils aktuelle Version wird nur via E-Mail als PDF-Dokument versandt. Sie können den KAFFEEKLATSCH via E-Mail an abo@bookware.de oder über das Internet unter www.bookware.de/abo bestellen. Selbstverständlich können Sie das Abo jederzeit und ohne Angabe von Gründen sowohl via E-Mail als auch übers Internet kündigen.

Ältere Versionen können einfach über das Internet als Download unter www.bookware.de/archiv bezogen werden.

Auf Wunsch schicken wir Ihnen auch ein gedrucktes Exemplar. Da es sich dabei um einzelne Exemplare handelt, erkundigen Sie sich bitte wegen der Preise und Versandkosten bei Alexandra Specht via E-Mail unter alexandra.specht@bookware.de oder telefonisch unter 09131/8903-14.

Copyright

Das Copyright des KAFFEEKLATSCHS liegt vollständig bei der Bookware. Wir gestatten die Übernahme des KAFFEEKLATSCHS in Datenbestände, wenn sie ausschließlich privaten Zwecken dienen. Das auszugsweise Kopieren und Archivieren zu gewerblichen Zwecken ohne unsere schriftliche Genehmigung ist nicht gestattet.

Sie dürfen jedoch die unveränderte PDF-Datei gelegentlich und unentgeltlich zu Bildungs- und Forschungszwecken an Interessenten verschicken. Sollten diese allerdings ein dauerhaftes Interesse am KAFFEEKLATSCH haben, so möchten wir diese herzlich dazu einladen, das Magazin direkt von uns zu beziehen. Ein regelmäßiger Versand soll nur über uns erfolgen.

Bei entsprechenden Fragen wenden Sie sich bitte per E-Mail an copyright@bookware.de.

Impressum

KAFFEEKLATSCH Jahrgang 4, Nummer 12, Dezember 2011
 ISSN 1865-682X
 BOOKWARE – eine Initiative der MATHEMA Software GmbH
 Henkestraße 91, 91052 Erlangen
 Telefon: 0 91 31 / 89 03-0
 Telefax: 0 91 31 / 89 03-55
 E-Mail: redaktion@bookware.de
 Internet: www.bookware.de
 Herausgeber/Redakteur: MICHAEL WIEDEKING
 Anzeigen: NATALIA WILHELM
 Grafik: NICOLE DELONG-BUCHANAN

Inhalt

Editorial	3
Beitragsinfo	4
Inhalt	5
User Groups	16
Werbung	18
Das Allerletzte	20

Artikel

Das Schweizer Taschenmesser im Compiler Code erzeugen mit Annotation Processing	6
Lisa, die neue Version Linux Mint 12 mit Gnome 3 im Klassikmodus	12

Kolumnen

So herum oder anders? Des Programmierers kleine Vergnügen	13
Lobgesang Deutsch für Informatiker	14
Kleines Weihnachtswunder Kaffeesatz	15

Das Schweizer Taschenmesser im Compiler

Code erzeugen mit Annotation Processing 6
von ANDREAS HEIDUK

Durch Plugins kann jeder *Java-Compiler* den eigenen Wünschen und Anforderungen angepasst werden – unabhängig von anderen Werkzeugen zur Code-Generierung. Und das Beste: Auch die Integration in die IDEs kommt frei Haus.

Lisa, die neue Version

Linux Mint 12 mit Gnome 3 im Klassikmodus 12
von PHILIPP HELMERT

Lisa heißt die aktuelle Version von *Linux Mint*. Hier steht vor allem der Desktop im Vordergrund. Die Distribution kommt mit *Gnome3*, einer Menge eigener Erweiterungen und liefert auch noch den *Gnome-2-Fork Mate* mit.

So herum oder anders?

Des Programmierers kleine Vergnügen 13
von MICHAEL WIEDEKING

Liest man ein beliebiges Datum ein, so stellt sich die Frage, wie die einzelnen Bytes innerhalb dieses Datums zu interpretieren sind. Hierbei unterscheidet man zwischen einem *big-endian*- und *little-endian*-Zugriff, je nachdem, ob das höchstwertigste Byte direkt an der Adresse liegt oder das niederwertigste. Wie kann man aber eigentlich herausfinden, ob man es mit dem einen oder anderen Zugriff zu tun hat?

Das Schweizer Taschenmesser im Compiler

Code erzeugen mit Annotation Processing

VON ANDREAS HEIDUK

D

urch Plugins kann jeder *Java-Compiler* den eigenen Wünschen und Anforderungen angepasst werden – unabhängig von anderen Werkzeugen zur Code-Generierung. Und das Beste: Auch die Integration in die IDEs kommt frei Haus.

Annotationen gehören nun schon seit langer Zeit zum Sprachumfang von Java. Einige der Annotationen wie `@Override` oder `@SuppressWarnings` werden direkt vom Compiler ausgewertet. Andere wie `@EJB` und `@Stateless` wertet ein Container zur Laufzeit aus und verwendet dabei die *Reflection-API*. Ein dritter Weg, die Annotationen zur *Compile-Zeit* auszuwerten, ist dagegen sehr selten anzutreffen.

Warum sollte man sich auch die Mühe machen? Ein gutes Beispiel liefert *JPA2* [2]. Die dort definierte *Criteria-API* erlaubt typ- und refactoring-sichere Abfragen wie folgendes gekürztes Beispiel aus dem Standard zeigt:

```
CRITERIABUILDER cb = ...;
CRITERIAQUERY<STRING> q = ...;
q.select(customer.get(Customer_.name)).where(cb.equal(
    item.getItem().get(Product_.productType),
    "printer"
));
```

Das Metamodell der *Entities* und der Attribute, hier z. B. `Customer_` oder `Customer_.name`, muss in diesem Fall schon während des Kompilierens verfügbar sein oder zumindest zugleich erzeugt werden. Da die Entitäten ja mit Annotationen definiert wurden, bietet es sich gerade in diesem Fall an, diese zur *Compile-Zeit* auszuwerten, entsprechenden Source-Code zu generieren und diesen dann ebenfalls gleich zu kompilieren.

JPA2 zeigt also, dass man schon zur *Compile-Zeit* Metadaten über Klassen oder Felder verfügbar machen

kann. Aber kann man das zugrunde liegende Verfahren auch in eigenen Projekten sinnvoll einsetzen? Ein einfaches Beispiel soll helfen, diese Frage zu klären.

Eigene Metadaten

Als Beispiel dienen folgende Klassen, bei denen zur besseren Übersicht der übliche Wasserkopf wie *Getter-* und *Setter-*Methoden nicht gezeigt werden:

```
class BUCHUNG {
    private BUCHUNGSTYP typ;
    private LIST<BUCHUNGSTEIL> teile;
}

class BUCHUNGSTEIL {
    private STRING bemerkung;
    private BIGDECIMAL soll;
    private BIGDECIMAL haben;
}
```

Angenommen man möchte aus allen Buchungen des Typs *KFZ* die Summe aller *soll*-Werte berechnen.

In einer hybriden funktionalen Sprache wie *Xtend* würde man das natürlich sehr elegant und übersichtlich formulieren:

```
buchungen.select(b | b.typ == KFZ).teile.soll.sum()
```

In reinem Java ist das auf die übliche Weise mit einer doppelt verschachtelten Schleife und einer Bedingung

auch in zehn Zeilen erledigt – aber leider ohne irgendeine sinnvolle Möglichkeit der Wiederverwendung oder der Vereinheitlichung. Wenn man viele ähnliche Aufgaben hat, die sich hauptsächlich durch die Namen der Properties unterscheiden, geht die Übersichtlichkeit schnell verloren.

Dabei wäre eine brauchbare Annäherung im Prinzip schnell und einfach möglich:

```

BigDECIMAL sum = AGGREGATES.sum(
    BUCHUNGSTEIL_.SOLL,
    QUASIFUNCTIONAL.collectFlatten(
        BUCHUNG_.TEILE,
        QUASIFUNCTIONAL.select(
            BUCHUNGSTYP_.EQ_KFZ, buchungen
        )
    )
);

```

Dieses Verfahren steht und fällt natürlich damit, dass man entsprechende Zugriffsmethoden auf die Felder als *First Class Object* [3] zur Hand hat. In diesem Beispiel benötigt man für jedes Feld eine Implementierung des Interface *Property*:

```

public interface PROPERTY<T, V> {
    V get(T item);
    void set(T item, V value);
}

```

Dabei entspricht *T* dem Typ der Klasse, die das Feld enthält, *V* entspricht dem deklarierten Typ des Feldes selbst. *SOLL* muss also so definiert werden:

```
PROPERTY<BUCHUNGSTEIL, BIGDECIMAL> SOLL = ...;
```

Die statischen Methoden *sum*, *collectFlatten* und *select* sind einfache, naive Implementierungen. Sie iterieren über die angegebenen Listen, wenden die Zugriffsfunktion auf jedes Element an und geben die jeweiligen Ergebnisse meist wieder als Liste zurück. Da die Implementierung dieser Methoden für das Beispiel nicht weiter relevant ist, wird nicht weiter darauf eingegangen.

Für die automatische Erzeugung der Zugriffsobjekte bietet es sich an, ein *Compiler-Plugin* zu schreiben. Das führt dann direkt zu einem *Pluggable Annotation Processor*.

Pluggable Annotaton Processing API ...

Seit Java 6 unterstützt der Java-Compiler den Standard *JSR-269* [1] – *Pluggable Annotation Processing API*. Dieser Standard erlaubt es, herstellerunabhängig in den *Compile-Prozess* einzugreifen, benötigte Annotationen und Typinformationen auszuwerten und neue Dateien

oder neuen Source-Code zu erstellen. Letzterer wird dann gleich kompiliert. Leider ist es nicht möglich, vorhandenen Source-Code zu ändern – die automatische Generierung der *Getter* und *Setter* ist also alleine mit diesem Standard nicht möglich.

Im Detail umfasst der Standard folgende Punkte:

- Erzeugen eines Annotation Processors (*AP*).
- Definition, wann der *AP* mit welchem Kontext aufgerufen wird.
- Schnittstellen, um die verschiedenen Elemente der kompilierten Quellen abzufragen (*Elements*).
- Schnittstellen, um auf alle verwendeten Typen zuzugreifen (*Types*).
- Schnittstellen, um Warnungen und Fehler an den Aufrufer zu kommunizieren (*Messenger* [sic!]).
- Schnittstellen, um neue Sourcen oder allgemeine Dateien anzulegen (*Filer*).

Diese Reihenfolge entspricht auch dem zeitlichen Ablauf im Code, man kann dem Beispiel-Code also entsprechend gut folgen.

Einbinden und Starten

Jedes Compiler-Plugin muss dem Compiler als *Jar-File* übergeben werden. In diesem *Jar-File* muss der Klassenname des Plugins als Service *javax.annotation.processing.Processor* eingetragen sein. Konkret muss also in der Datei *META-INF/services/javax.annotation.processing.Processor* der Wert *MyProcessor* vermerkt sein.

Ein solches *Plugin-Jar* wird dem Java-Compiler mittels der Option *-processorpath* übergeben. In Eclipse fügt man ein solches *Jar* über die Projekteigenschaften unter *Java Compiler – Annotation Processing – Factory Path* hinzu.

Der Annotation Processor

In beiden Fällen wird ein entsprechender Prozessor erzeugt und befragt, welches Sprachniveau er unterstützt, welche Optionen er versteht und vor allem welche Annotationen er behandeln möchte.

Im einfachsten Fall überlässt man der abstrakten Klasse *AbstractProcessor* den Großteil dieser Arbeit und annotiert seine konkrete Implementierung einfach mit *@SupportedAnnotationTypes* und übergibt dabei den qualifizierten Klassennamen seiner eigenen Annotation.

In diesem Beispiel werden einfach alle Felder, für die Zugriffsobjekte erzeugt werden sollen, mit der Annotation *@PropertyMarker* versehen. Die Annotation selbst ist folgendermaßen definiert:

```
@Target(ELEMENTTYPE.FIELD)
@Retention(RETENTIONPOLICY.SOURCE)
public @interface PROPERTYMARKER {
}
```

Sobald nun der Compiler eine Klasse übersetzt, in der diese Annotation vorkommt, wird die Methode *process* aufgerufen. Dabei bekommt man neben einem *Set* von Annotationstypen auch eine Referenz auf ein *RoundEnvironment* übergeben. Diese beiden Teile spielen wie folgt zusammen: Der Compiler arbeitet in einer oder mehreren Runden. In jeder Runde übersetzt er ein wenig und fragt dann der Reihe nach die registrierten Prozessoren, ob sie sich für gefundene Annotationen vom übergebenen Typ interessieren. Wenn *process* mit *true* zurückkehrt, so hat der aktuelle Prozessor die Annotationen endgültig verarbeitet und weitere Prozessoren bekommen diesen Typ in dieser Runde nicht mehr angeboten. Wenn man sich für die angebotenen Annotationen interessiert, so kann man mittels *RoundEnvironment* die eben übersetzten *Elemente* untersuchen. Wenn ein Prozessor in einer Runde neuen Source-Code oder neue *Class-Files* erzeugt, werden diese in der nächsten Runde berücksichtigt.

Elemente

Mit dem Interface *Element* und dessen Unterklassen werden die groben Sprachelemente von Java beschrieben. Es gibt daher je einen Elementtyp für:

- Methoden, Konstruktoren und Initialisierer,
- Packages,
- Klassen und Interfaces,
- generische Typparameter und
- Felder, Konstanten, Parameter von Methoden und Konstruktoren sowie lokale Variablen.

Die verschiedenen Elemente sind baumartig miteinander verbunden und können sowohl zu Fuß als auch per *Visitor* untersucht werden.

Im einfachsten Fall aber lässt man sich die passend annotierten Elemente geben und filtert sicherheitshalber nach Feldern, hier vom Typ *VariableElement*.

```
SET<? extends ELEMENT> annotatedElements =
    roundEnv.getElementsAnnotatedWith(markerAnnotation);
SET<VARIABLEELEMENT> annotatedFields =
    ELEMENTFILTER.fieldsIn(annotatedElements);
```

Wie der Name *ElementFilter* schon andeutet, finden sich hier einige Hilfsmethoden, um schnell nach bestimmten Elementtypen filtern zu können. Eine weitere Werk-

zeugkiste (eher ein Kistchen) findet man im Interface *Elements*. Eine Referenz darauf erhält man, wenn man wie in diesem Beispiel von *AbstractProcessor* erbt, über *processingEnv.getElementUtils()*.

Die Elemente selbst beinhalten keinerlei Informationen über die verwendeten Typen, diese sind in einer zweiten Hierarchie basierend auf *TypeMirror* abgebildet. Mit *Element.asType()* erhält man den entsprechenden *TypeMirror*. In der Rückrichtung funktioniert das nur bei bestimmten Typen wie z. B. bei Deklarationen. Auch die Typ-Hierarchie kann mit und ohne *Visitor* untersucht werden und über das *ProcessingEnvironment* erhält man analog zu *Elements* Zugriff auf einige Hilfsmethoden in der Klasse *Types*.

Für das Beispiel werden pro annotiertem Feld nur vier Informationen benötigt:

- Name des Feldes
- Typ des Feldes
- Name der Bean
- Name des Packages

Diese Informationen holt man sich am einfachsten direkt. Ausgehend von einem *VariableElement* erhält man den Feldnamen über *getSimpleName()*, den Typ in fertig formatierter Form über *asType().toString()*. Der Name der Bean ist über *getEnclosingElement().getSimpleName()* erreichbar und der Package-Name über einen weiteren derartigen Schritt, da ja annotierte Felder in einer Klasse und diese wieder in einem Package liegt. Nun hat man für die Code-Erzeugung alle Informationen beisammen.

Code-Erzeugung

Für den Zugriff auf Dateien gibt es in der allgemeinen Werkzeugkiste *ProcessingEnvironment* den *Filer*. Mit seiner Hilfe kann man

- Sourcen schreiben,
- Class-Files schreiben,
- allgemeine Dateien schreiben
- und alle drei Arten von Dateien lesen.

Die Unterscheidung bei den schreibenden Zugriffen liegt in der Art und Weise, wie die geschriebenen Dateien in den weiteren Compile-Prozess eingebunden werden.

Zu beachten ist, dass geschriebener Source nicht in das Verzeichnis geschrieben wird, in dem die normalen Sourcen liegen, sondern in ein eigenes Verzeichnis. Das hat auch den Vorteil, dass man alle erzeugten Artefakte schnell wieder wegräumen kann.

Bei den schreibenden Zugriffen kann man optional die Elemente angeben, die Einfluss auf den Inhalt der erzeugten Datei haben. Für den Standard-Compiler ist diese Information nicht relevant, aber der inkrementelle Compiler von Eclipse sowie der Fehlermarker kann sie verwenden. Ansonsten bietet der *Filer* keine weiteren wichtigen Überraschungen.

Fehlermeldungen

Apropos Überraschungen: Wie werden Fehlermeldungen sinnvoll ausgegeben?

„Sinnvoll“ bedeutet in diesem Zusammenhang nicht unstrukturierte Ausgaben auf der Konsole, sondern etwas, womit auch IDEs Fehler exakt einem Code-Element zuordnen können. Über *ProcessingEnvironment* bekommt man einen *Messenger*. Dieser bietet die Möglichkeit, Nachrichten mit verschiedenen Dringlichkeitsstufen auszugeben. Bei den Nachrichten kann man auch angeben, auf welches Element sie sich beziehen. Eclipse behandelt so erzeugte Fehler und Warnungen wie eigene Meldungen – sie werden also im *Editor* angezeigt und in der *Problems-View* aufgelistet und sind mit dem fehlerhaften Code-Element verbunden. Alle anderen Meldungen landen in dem *Error-View* von Eclipse.

Fazit

Damit ist ein erster Rundgang durch die notwendigen Schritte abgeschlossen. Wie das Beispiel zeigt, kann man die nötigen Daten in unter 60 Zeilen Code extrahieren und mit rund 40 weiteren Zeilen, passenden Code erzeugen. Der Aufwand für eine maßgeschneiderte und individuelle Lösung ist also gar nicht so groß, sobald die ersten Hürden überwunden sind. Vielleicht probieren Sie es auch einmal?

Der vollständige Prozessor und die Beispiele stehen für Sie hier zum Download bereit:

http://www.bookware.de/kaffeeklatsch/AptProcessor_2011-12.zip

Referenzen

- [1] JSR-269 *Pluggable Annotation Processing API*, 2006
- [2] JSR-317 *Java Persistence 2.0*, Dez 2009
- [3] WIKIPEDIA *First-Class-Objekt*,
<http://de.wikipedia.org/wiki/First-Class-Objekt>

Kurzbiographie



ANDREAS HEIDUK (andreas.heiduk@mathema.de) ist als Senior Consultant für die MATHEMA Software GmbH tätig. Seine Themenschwerpunkte umfassen die Java Standard Edition (JSE) und die Java Enterprise Edition (JEE). Daneben findet er alle Themen von hardware-naher Programmierung bis hin zu verteilten Anwendungen interessant.

Source (Code)

MyProcessor.java

```
package pap;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.annotation.processing.AbstractProcessor;
import javax.annotation.processing.RoundEnvironment;
import javax.annotation.processing.SupportedAnnotationTypes;
import javax.annotation.processing.SupportedSourceVersion;
import javax.lang.model.SourceVersion;
import javax.lang.model.element.Element;
import javax.lang.model.element.TypeElement;
import javax.lang.model.element.VariableElement;
import javax.lang.model.type.PrimitiveType;
import javax.lang.model.type.TypeMirror;
import javax.lang.model.util.ElementFilter;
import javax.tools.Diagnostic.Kind;

@SupportedSourceVersion(
    SourceVersion.RELEASE_6
)
@SupportedAnnotationTypes(
    MyProcessor.API_MARKER_PROPERTY
)
public class MyProcessor extends AbstractProcessor {
    public static final String API_MARKER_PROPERTY =
        "api.PROPERTYMARKER";

    private void note(String msg){
        processingEnv.getMessenger().printMessage(
            Kind.NOTE, msg
        );
    }

    @Override
    public boolean process(
        Set<? extends TypeElement> annotations,
        RoundEnvironment roundEnv
    ){
        note("Hello World! Got "+annotations);

        if( annotations == null || annotations.isEmpty() )
            return false;

        final TypeElement markerAnnotation =
            processingEnv.getElementUtils().getTypeElement(
                API_MARKER_PROPERTY
            );

        Map<TypeElement,
            List<PropertyDescription>> todoItems =
            new HashMap<TypeElement,
                List<PropertyDescription>>();
```

```

SET<? extends ELEMENT> annotatedElements =
    roundEnv.getElementsAnnotatedWith(
        markerAnnotation
    );
SET<VARIABLEELEMENT> annotatedFields =
    ELEMENTFILTER.fieldsIn(annotatedElements);

note("found annotated fields: "+annotatedFields);

for (VARIABLEELEMENT varElem: annotatedFields){
    TYPEELEMENT enclosingElement =
        (TYPEELEMENT)varElem.getEnclosingElement();

    LIST<PROPERTYDESCRIPTION> list =
        todoItems.get(enclosingElement);
    if(list==null){
        list = new ARRAYLIST<PROPERTYDESCRIPTION>();
        todoItems.put(enclosingElement, list);
    }

    STRING propertyName =
        varElem.getSimpleName().toString();
    TYPEMIRROR propertyType = varElem.asType();
    if( propertyType.getKind().isPrimitive() )
        propertyType =
            processingEnv.getTypeUtils().boxedClass(
                (PRIMITIVE TYPE) propertyType
            ).asType();
    STRING propertyTypeName =
        propertyType.toString();
    list.add(
        new PROPERTYDESCRIPTION(
            propertyName, propertyTypeName
        )
    );
}

PROPERTYFILEWRITER pfw =
    new PROPERTYFILEWRITER(processingEnv);
for(
    MAP.ENTRY<TYPEELEMENT,
    LIST<PROPERTYDESCRIPTION>>
    entry: todoItems.entrySet()
){

    TYPEELEMENT typeElement = entry.getKey();
    STRING packageName =
        typeElement.getEnclosingElement().
        getSimpleName().toString();

    STRING beanName =
        typeElement.getSimpleName().toString();
    LIST<PROPERTYDESCRIPTION> propertyDescriptions =
        entry.getValue();

    pfw.writePropertyFile(
        packageName, beanName, propertyDescriptions,
        typeElement
    );
}

return true;
}
}

```

PropertyDescription.java

```

package pap;

public class PROPERTYDESCRIPTION {
    private STRING propertyName;
    private STRING propertyType;

    public PROPERTYDESCRIPTION(){
    }

    public PROPERTYDESCRIPTION(
        STRING propertyName, STRING propertyType
    ){
        this.propertyName = propertyName;
        this.propertyType = propertyType;
    }

    @Override
    public STRING toString() {
        return "("+propertyType+" "+propertyName+")";
    }

    public STRING getPropertyName() {
        return propertyName;
    }

    public void setPropertyName(STRING propertyName) {
        this.propertyName = propertyName;
    }

    public STRING getPropertyType() {
        return propertyType;
    }

    public void setPropertyType(STRING propertyType) {
        this.propertyType = propertyType;
    }
}

```

PropertyFileWriter.java

```

package pap;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import
    javax.annotation.processing.ProcessingEnvironment;
import javax.lang.model.element.Element;
import javax.tools.Diagnostic.Kind;
import javax.tools.JavaFileObject;

public class PROPERTYFILEWRITER {
    private final PROCESSINGENVIRONMENT env;

```

```

public PROPERTYFILEWRITER(
    PROCESSINGENVIRONMENT env
){
    this.env = env;
}

public void writePropertyFile(
    STRING packageName, STRING beanName,
    LIST<PROPERTYDESCRIPTION> propertyDescriptions,
    ELEMENT... originatingElements
){
    try {
        writePropertyFileIntern(
            packageName, beanName, propertyDescriptions,
            originatingElements
        );
    }
    catch (IOEXCEPTION e){
        throw new RUNTIMEEXCEPTION(e);
    }
}

private void writePropertyFileIntern(
    STRING packageName, STRING beanName,
    LIST<PROPERTYDESCRIPTION> propertyDescriptions,
    ELEMENT... originatingElements
) throws IOEXCEPTION {
    STRING generatedClassName = beanName+'_'+;

    STRING qualifiedName =
        packageName == null || packageName.isEmpty() ?
        generatedClassName
        : packageName+'.'+generatedClassName;

    env.getMessenger().printMessage(
        KIND.OTHER, "writing "+qualifiedName
    );
    JAVAFILEOBJECT jfo =
        env.getFile().createSourceFile(
            qualifiedName, originatingElements
        );
    PRINTWRITER pw = new PrintWriter(
        jfo.openWriter()
    );
    try {
        if( packageName !=
            null && !packageName.isEmpty()
        )
            pw.println("package "+packageName+";");

```

```

pw.println("import api.PROPERTY;");

pw.println("public class "+generatedClassName+" {");
for(PROPERTYDESCRIPTION propertyDescription:
    propertyDescriptions
){
    STRING propertyName =
        propertyDescription.getPropertyName();
    STRING propertyType =
        propertyDescription.getPropertyType();

    STRING methodSuffix =
        propertyName.substring(0, 1).toUpperCase()+
        propertyName.substring(1);
    STRING getterName = "get"+methodSuffix;
    String setterName = "set"+methodSuffix;

    pw.println("
        public final static PROPERTY<"+beanName+",
            "+propertyType+"> "+propertyName+" =
        new PROPERTY<"+beanName+",
            "+propertyType+">() {");
    );
    pw.println("    @Override");
    pw.println("    public void set(
        "+beanName+" item, "+propertyType+" value
    ) {");
    );
    pw.println("        item."+setterName+"(value);");
    pw.println("    }");
    pw.println("    @Override");
    pw.println("
        public "+propertyType+" get("+beanName+" item) {");
    );
    pw.println("        return item."+getterName+"();");
    );
    pw.println("    }");
    pw.println("};");
    pw.println();
}
pw.println("}");
}
finally {
    pw.close();
}
}
}

```

Lisa, die neue Version

Linux Mint 12 mit Gnome 3 im Klassikmodus

VON PHILIPP HELMERT

Lisa heißt die aktuelle Version von *Linux Mint*. Hier steht vor allem der Desktop im Vordergrund. Die Distribution kommt mit *Gnome 3*, einer Menge eigener Erweiterungen und liefert auch noch den *Gnome-2-Fork Mate* mit.

Allgemeines

Nachdem vor ungefähr einem Monat *Ubuntu 11.10* veröffentlicht worden ist, ziehen nun die Mint-Entwickler nach. Linux Mint, Codename Lisa, verwendet Ubuntu 11.10 als Untergerüst und hat somit die aktuelle Software-Ausstattung.

Neuerungen

Die Entwickler von Linux Mint haben das Artwork komplett überarbeitet, zu dem gehört nun auch ein schwarzer *Boot*-Bildschirm. Allerdings hat dieser keine Fortschrittsanzeige mehr, da die grafischen Elemente von der verwendeten Hardware abhängig waren und somit gab es auf einigen Systemen Darstellungsschwierigkeiten. Auch die *Icons* und das *Mint-Gtk-Theme* wurden überarbeitet.

Die größte Neuerung ist allerdings wie bei Ubuntu 11.10 der Umstieg auf Gnome 3. Allerdings merkt der Nutzer kaum etwas davon. Das liegt daran, dass bei Linux Mint standardmäßig die *Mint*-Erweiterungen aktiviert sind (*Mint Gnome Shell Extensions* MGSE). Nachgerüstet wurde somit ein *Panel* am unteren Bildschirmrand, dort findet man den Desktop-Umschalter sowie auch ein klassisches Startmenü und eine Fensterleiste.

Sicherlich gibt es auch den Einen oder Anderen, der Gnome 3 in voller Pracht genießen möchte. Dies ist auch kein Problem. Deaktiviert man alle Mint-Erweiterungen, so läuft Gnome 3 mit den Standardeinstellungen. Wenn man wissen möchte, welche Erweiterungen im Moment aktiv sind, so kann man diese im *gnome-tweak-tool* einsehen. In dem Tuning-Werkzeug für Gnome stehen im Bereich *Shell-Erweiterungen* alle Mint-Erweiterungen

zur Auswahl, die sich einzeln an- und abschalten lassen. So kann man beispielsweise auf die Anzeige des *Panels* am unteren Bildschirmrand verzichten, aber trotzdem Menü und Fensterleiste nutzen.

Die Gnome-Shell-Erweiterungen lassen sich normalerweise im laufenden Betrieb an- und abschalten. Allerdings treten hier zur Zeit noch einige Fehler auf. So kann es passieren, dass sich die Fensterliste nicht mehr ausblenden oder sich das gerade aktivierte Menü nicht mehr anklicken lässt. Das Problem behebt man am schnellsten indem man die Gnome-Shell mit folgendem Befehl neu startet: *gnome-shell --replace*.

Der erste Eindruck

Linux Mint wirkt sehr gelungen. Die solide Ubuntu-Basis in Kombination mit neuen Ideen beim Desktopbereich begeistern immer mehr User. Mit den Gnome-Shell-Erweiterungen haben die Entwickler auf die Wünsche vieler User reagiert. Das haben sie auch sehr gut umgesetzt, ohne dabei die Wahlfreiheit der Anwender einzuschränken.

Lisa ist somit eine sehr moderne und stilsichere Distribution, die viele Funktionen bietet aber auch mit dem Design überzeugen kann.

Referenzen

[1] Linux Mint *home*, <http://linuxmint.com>

Kurzbiographie



PHILIPP HELMERT (philipp.helmert@mathema.de) ist Auszubildender bei der MATHEMA Software GmbH. Sein Interesse liegt neben der Administration bei der Programmiersprache Ruby. Außerdem beschäftigt er sich mit dem Web-Framework Ruby on Rails und zahlreichen Linux-Distributionen.

Des Programmierers kleine Vergnügen So herum oder anders?

VON MICHAEL WIEDEKING

Liest man ein beliebiges Datum ein, so stellt sich die Frage, wie die einzelnen Bytes innerhalb dieses Datums zu interpretieren sind. Hierbei unterscheidet man zwischen einem *big-endian*- und *little-endian*-Zugriff, je nachdem, ob das höchstwertigste Byte direkt an der Adresse liegt oder das niederwertigste. Wie kann man aber eigentlich herausfinden, ob man es mit dem einen oder anderen Zugriff zu tun hat?

Zunächst einmal sei erwähnt, dass dieses Problem eigentlich nichts für Weicheier ist. Java- und .Net-Entwickler mit ihrem verwalteten Speicher und den Referenzen können da eigentlich gar nicht mitreden, denn die wissen ja meist noch nicht einmal, was eine Adresse überhaupt ist.

Natürlich ist das nicht ganz ernst gemeint, denn das Problem tritt immer noch auf, wenn man beispielsweise ein *int* Byte für Byte über ein Netzwerk überträgt. Da muss ja auch entschieden werden, ob das erste Byte, das übertragen wird, das höchstwertigste oder niederwertigste Byte darstellen soll.

Das Problem ist übrigens keines, das man nur in der Informatik findet. Beispielsweise übertragen wir im Deutschen die zweistelligen Zahlen zwischen 21 und 99, die nicht glatt durch zehn teilbar sind, verbal in einem *little-endian*-Format. Denn die Zahl 23 sprechen wir als *drei-und-zwanzig* aus, der niederwertige Teil wird also zuerst über das Medium Luft verschickt.

Im Englischen ist es genau umgekehrt, denn dort bedient man sich der *big-endian*-Variante. Hier wird die Zahl 23 mathematischer als *twenty-three* ausgesprochen. Das war aber nicht schon immer so. In älterer Literatur findet sich nämlich auch die *little-endian*-Variante *three-and-twenty*.

Das macht man sich bei Unicode im Zusammenhang mit einem *Zwei-Byte*-Transportformat ganz geschickt zu Nutze, denn dort ist das Zeichen U+FFFE mit dem Wert 0xFFFF nicht definiert. Das „umgekehrte“ Zeichen U+FEFF mit dem Wert 0xFEFF ist aber sehr wohl definiert und wird auch als *Byte Order Mark* (BOM) bezeichnet. Trifft man nämlich auf die Kombination 0xFFFF, so weiß man, dass die Datei in umgekehrter Byte-Reihenfolge interpretiert werden muss. Bei dem Wert 0xFEFF hat man dann entsprechend die Gewissheit, dass die Datei in der korrekten Byte-Folge gelesen wird. Das BOM

kann in diesem Fall getrost ignoriert werden, weil es als nicht trennendes Leerzeichen ohne Ausdehnung definiert ist.

Legt man beispielsweise das *int* 0x12345678, das aus den Bytes 0x12, 0x34, 0x56 und 0x78 besteht, im Speicher an der Adresse *a* ab, so befindet sich auf einer *big-endian*-Maschine das höchstwertigste Byte 0x12 an der Adresse *a* und das niederwertigste Byte 0x78 an der Stelle *a + 3*. Auf einer *little-endian*-Maschine ist es genau umgekehrt, da befindet sich das niederwertigste Byte 0x78 an der Stelle *a* und das höchstwertigste Byte 0x12 an der Stelle *a + 3*.

Um herauszufinden, ob eine Maschine, auf der ein Programm läuft, eine *big-endian*- oder *little-endian*-Architektur hat, braucht man also nur herauszufinden, ob bei einem geeigneten *int* an dessen Anfangsadresse das höchstwertigste oder niederwertigste Byte steht. In C lässt sich das ganz einfach mit der Funktion

```
int isLittleEndian() {
    int i = 1;
    return *((char *) &i);
}
```

herausfinden. Dazu speichert man einfach die Zahl 1 (hexadezimal 0x00000001) in der *int*-Variablen *i* ab. Dann lässt man sich mit *&i* die Adresse dieser Variablen geben. Die Adresse *castet* man auf eine Byte-Adresse, die man dann mit ***(...) dereferenziert. Damit steht einem genau das Byte zur Verfügung, das an der Anfangsadresse der *int*-Variablen liegt.

Das niederwertigste Byte 0x01 bleibt dann und nur dann an der Anfangsadresse enthalten, wenn es sich um eine *little-endian*-Architektur handelt; im Falle der *big-endian*-Architektur findet sich dort das höchstwertigste Byte 0x00. Das ist deswegen so praktisch, weil in C der Wert 1 als *true* gilt und der Wert 0 als *false*. Voilà!

Wer dann auch noch das Problem hat, dass er wegen der falschen Architektur die Byte-Folge umdrehen muss, dem sei die Oktoberausgabe des *Kleinen Vergnügens* aus 2009 empfohlen [1], die sich schon ausgiebig mit diesem Thema beschäftigt hat. Und wer erfahren möchte, wie das mit den Begriffen *big-endian* und *little-endian* seinen Anfang genommen hat, der sollte den legendären Aprilscherz von 1980 [2] lesen.

Referenzen

- [1] WIEDEKING, MICHAEL *Des Programmierers kleine Vergnügen – Teile und ... tausche*, KAFFEEKLATSCH, Jahrgang 2, Nr. 10, Bookware, Oktober 2009
<http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2009-10.pdf>
- [2] COHEN, DANNY *On Holy Wars And A Plea For Peace*, Internet Engineering Task Force (IETF), 1. April 1980
<http://www.ietf.org/rfc/ien/ien137.txt>

Lobgesang

VON ALEXANDRA SPECHT

In dieser weihnachtlichen Kolumne geht es um die Hymne.

Die Hymne (aus dem Griechischen: *hymnos*=Tongefüge) ist eine Gedichtform. Die beste Übersetzung ist wohl *Lobgesang*!

Die Hymne hat keine feste Form, was es nicht viel einfacher macht, eine zu schreiben, weil manchmal so ein bisschen Struktur und gewisse Vorgaben das Leben vereinfachen können, aber das muss ich Ihnen ja nicht sagen. Ganz ohne Vorgaben mag wohl niemand arbeiten.

Inhaltlich befasst sich die Hymne oft mit der Preisung (eines) Gottes. Sie kann aber auch die Natur, eine Person oder aber ein Gefühl besingen. Mehr gibt es auch schon nicht wirklich zu der Hymne zu schreiben, außer vielleicht, dass es sie schon in der ägyptischen Antike gab; zu erwähnen sind da besonders die sogenannten Pyramidentexte aus dem 24. Jahrhundert vor Christus.

In der Bibel gibt es auch Lobeshymnen, z. B. das Buch der Psalmen. Natürlich gab es auch in der griechisch-römischen Antike Hymnen, im Barock und besonders auch in der Empfindsamkeit; wer kennt nicht SCHILLERS Ode *An die Freude*.

Was hat das aber nun mit *Deutsch für Informatiker* zu tun, fragen Sie sich, weil ich gerade nicht da bin, um mich zu fragen. Nun, ich dachte, wir haben dieses Jahr schon Grammatik gehabt, ein bisschen Soziolinguistik, Programmpoesie und noch einiges mehr. Darum ein bisschen was zur Entspannung: Literatur.

Literatur lässt sich historisch betrachtet in die drei Gattungen Epik, Lyrik und Dramatik gliedern. Epik klammern wir aus, Dramatik haben Sie sicher in Ihrem Job genug und darum hier noch ein bisschen mehr Lyrik.

Eigentlich wollte ich Ihnen eine schöne Hymne schreiben. Aber ich habe alle Versuche verwerfen müssen und bin froh, dass ich meinen Lebensunterhalt nicht als Lyrikerin bestreiten muss. Stattdessen möchte Sie dazu ermutigen, selbst Hymnen zu schreiben. Auf Ihre Leidenschaft, sei es die Natur, die Profession, die Liebste, der Liebste. Und wenn sie möchten, schicken Sie sie uns! Das wäre doch mal eine schöne Ausgabe *Deutsch für Informatiker*, lauter Hymnen von Informatikern.

Ich wünsche Ihnen schöne entspannte Weihnachten mit oder ohne Literatur, aber mit viel Leidenschaft bei allem, was Sie machen!

Kleines Weihnachtswunder

VON MICHAEL WIEDEKING

Das Schöne an der Adventszeit ist, dass man sich trotz des obligatorischen Stresses zum Jahresende, doch noch relativ oft die Zeit nimmt, Plätzchen zu backen, gemütlich beisammen zu sitzen, Glühwein- oder Punsch-Varianten zu trinken und jene frisch gebackenen zu verspeisen. Und bei Allem was es zu erzählen gibt, ist eines ein immer wiederkehrendes Thema: die vergangenen, die gegenwärtigen und zukünftigen Weihnachtsfeiern und deren Gestaltung.

Was man da alles zu hören bekommt hat unbeschreiblichen Unterhaltungswert: Angefangen bei Abenteuern mit dem brennenden Adventskranz, über Dramen wegen falscher Geschenke oder Unmut wegen schlechten Timings, bis hin zu den abstrusesten Gewohnheiten und absonderlichsten Ritualen, und nicht zuletzt die kleinen Wunder.

Als wir neulich mit einem befreundeten Paar zusammen saßen, erzählten sie uns von einem Problem, das sie während einer der vergangenen Weihnachten hatten: Bei den diversen Geschenken, die für die Verwandtschaft gedacht waren, war wohl eines vergessen worden und musste sich noch im Buchladen befinden, in dem zwar es bestellt, aber offensichtlich nicht abgeholt worden war.

Das wäre kein Problem gewesen, wenn diese kleine Unzulänglichkeit nicht erst am Abend des ersten Weihnachtstages aufgefallen wäre. So überlegte man, wie man sich aus diesem Dilemma heraus manövrieren könnte, denn es kommt nicht wirklich gut, wenn alle ein Geschenk bekommen und nur einer nicht. So ergab das eine das andere und schließlich tat sich eine unwahrscheinliche, aber nicht unmögliche Lösung auf.

Allerdings glaube ich, dass nur Männer auf eine so blödsinnige Ideen kommen können. Denn er schlug vor, ob sie nicht einfach versuchen sollten in dem kleinen Buchladen anzurufen. Sie meinte allerdings, dass könne man unabhängig von den Erfolgsaussichten nicht ma-

chen, denn schließlich wäre es Weihnachtsabend. Nach einigem Hin und Her rief er aber trotzdem an. Obwohl er vielleicht insgeheim nur damit gerechnet haben mochte, den Anrufbeantworter zu erreichen, so meldete sich zu aller Überraschung tatsächlich die Buchhändlerin.

Das Problem war schnell beschrieben, der Fehler genau so schnell gefunden und tatsächlich befand sich das Buch noch im Laden. Man könne es noch abholen, ließ die Buchhändlerin verlauten, und so machte sich das Paar am Weihnachtsabend spät noch auf, um das Buch aus dem Laden abzuholen. Ich glaube, nur Frauen kann so etwas peinlich sein, und so ging sie nicht mit in den Laden, um das Buch abzuholen. Stattdessen versteckte sie sich und ließ ihn allein in den Laden gehen. Und so machte er das Unvorstellbare: er ging in den Laden und holte das Buch ab.

So konnte wider Erwarten das Geschenk am nächsten Tag doch noch ausgehändigt werden – Weihnachten war sozusagen gerettet.

Das müssen Amazon & Co. erst einmal nachmachen.

Kurzbiographie



MICHAEL WIEDEKING (michael.wiedeking@mathema.de) ist Gründer und Geschäftsführer der MATHEMA Software GmbH, die sich von Anfang an mit Objekttechnologien und dem professionellen Einsatz von Java einen Namen gemacht hat. Er ist Java-Programmierer der ersten Stunde, „sammelt“ Programmiersprachen und beschäftigt sich mit deren Design und Implementierung.

User Groups

Fehlt eine User Group? Sind Kontaktdaten falsch?
Dann geben Sie uns doch bitte Bescheid.

BOOKWARE

Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de

Java User Groups

DEUTSCHLAND

JUG Berlin Brandenburg

<http://www.jug-bb.de>
Kontakt: Herr Ralph Bergmann (orga@jug-bb.de)

Java User Group Saxony

Java User Group Dresden
<http://www.jugsaxony.de>
Kontakt: Herr Torsten Rentsch (torsten@jugsaxony.de)
Herr Falk Hartmann (falk@jugsaxony.de)
Herr Kristian Rink (kristian@jugsaxony.de)

rheinjug e.V.

Java User Group Düsseldorf
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf
<http://www.rheinjug.de>
Kontakt: Herr Heiko Sippel (info@rheinjug.de)

ruhrjug

Java User Group Essen
Glaspavillon Uni-Campus
Universitätsstr. 12, 45127 Essen
<http://www.ruhrjug.de>
Kontakt: Herr Heiko Sippel (heiko.sippel@ruhrjug.de)

JUGF

Java User Group Frankfurt
<http://www.jugf.de>
Kontakt: Herr Alexander Culum
(jvausergroupfrankfurt@googlemail.com)

JUG Deutschland e.V.

Java User Group Deutschland e.V.
c/o asc-Dienstleistungs GmbH
Ehrengard-Schramm-Weg 11, 37085 Göttingen
<http://www.java.de> (office@java.de)

JUG Hamburg

Java User Group Hamburg
<http://www.jughh.org>

JUG Karlsruhe

Java User Group Karlsruhe
Universität Karlsruhe, Gebäude 50.34
Am Fasanengarten 4, 76131 Karlsruhe
<http://jug-karlsruhe.de>
jug-karlsruhe@gmail.com

JUGC

Java User Group Köln
<http://www.jugcologne.org>
Kontakt: Herr Michael Hüttermann
(michael@huettermann.net)

jugm

Java User Group München
Jupiterweg 8, 85586 Poing
<http://www.jugm.de>
Kontakt: Herr Andreas Haug (ah@jugm.de)

JUG Münster

Java User Group für Münster und das Münsterland
<http://www.jug-muenster.de>
Kontakt: Herr Thomas Kruse (tkjugi@sforce.org)

JUG MeNue

Java User Group der Metropolregion Nürnberg
c/o MATHEMA Software GmbH
Henkestraße 91, 91052 Erlangen
<http://www.jug-n.de>
Kontakt: Frau Alexandra Specht
(alexandra.specht@jug-n.de)

JUG Ostfalen

Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
Siekstraße 4, 38444 Wolfsburg
<http://www.jug-ostfalen.de>
Kontakt: Uwe Sauerbrei (info@jug-ostfalen.de)

JUGS e.V.

Java User Group Stuttgart e.V.
c/o Dr. Michael Paus
Schönaicherstraße 3, 70597 Stuttgart
<http://www.jugs.org>
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)

SCHWEIZ

JUGS

Java User Group Switzerland
Postfach 2322, 8033 Zürich
<http://www.jugs.ch> (info@jugs.ch)
Kontakt: Frau Ursula Burri

.Net User Groups

DEUTSCHLAND

.NET User Group OWL

http://www.gedoplan.de/cms/gedoplan/ak/ms_net
% GEDOPLAN GmbH
Stieghorster Str. 60, 33605 Bielefeld

.Net User Group Bonn

.NET User Group "Bonn-to-Code.Net"
Langwartweg 101, 53129 Bonn
<http://www.bonn-to-code.net> (mail@bonn-to-code.net)
Kontakt: Herr Roland Weigelt

Dodned

.NET User Group Franken
<http://www.dodned.de>
 Kontakt: Herr Bernd Hengelein
 Herr Thomas Müller (consulting@tom-mue.de)

.net Usergroup Frankfurt

c/o Thomas Sohnrey, Agile IService
 Mendelssohnstrasse 80, 60325 Frankfurt
<http://www.dotnet-ug-frankfurt.de>
 Kontakt: Herr Thomas 'Teddy' Sohnrey
 (thomas.sohnrey@gmx.de)

.NET DGH

.NET Developers Group Hannover
 Landwehrstraße 85, 30519 Hannover
<http://www.dotnet-hannover.de>
 Kontakt: Herr Friedhelm Drecktrah
 (friedhelm@drecktrah.de)

INdotNET

Ingolstädter .NET Developers Group
<http://www.indot.net>
 Kontakt: Herr Gregor Biswanger
 (gregor.biswanger@web-enliven.de)

DNUG-Köln

DotNetUserGroup Köln
 Goldammerweg 325, 50829 Köln
<http://www.dnug-koeln.de>
 Kontakt: Herr Albert Weinert (info@der-albert.com)

.Net User Group Leipzig

Brockhausstraße 26, 04229 Leipzig
<http://www.dotnet-leipzig.de>
 Kontakt: Herr Alexander Groß (agross@dotnet-leipzig.de)
 Herr Torsten Weber (tweber@dotnet-leipzig.de)

.NET Developers Group München

<http://www.munichdot.net>
 Kontakt: Hardy Erlinger (hardy.erlinger@hotmail.com)

.NET User Group Oldenburg

c/o Hilmar Bunjes und Yvette Teiken
 Sachsenstr. 24, 26121 Oldenburg
<http://www.dotnet-oldenburg.de>
 Kontakt: Herr Hilmar Bunjes
 (hilmar.bunjes@dotnet-oldenburg.de)
 Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)

.NET User Group Paderborn

c/o Net at Work Netzwerksysteme GmbH,
 Am Hoppenhof 32, 33104 Paderborn
<http://www.dotnet-paderborn.de>
 (raacke@dotnet-paderborn.de)
 Kontakt: Herr Mathias Raacke

.Net Developers Group Stuttgart

Tieto Deutschland GmbH
 Mittlerer Pfad 2, 70499 Stuttgart
<http://www.devgroup-stuttgart.de>
 (GroupLeader@devgroup-stuttgart.de)
 Kontakt: Frau Catrin Busley

.net Developer-Group Ulm

c/o artiso solutions GmbH
 Oberer Wiesenweg 25, 89134 Blaustein
<http://www.dotnet-ulm.de>
 Kontakt: Herr Thomas Schissler (tschissler@artiso.com)

ÖSTERREICH**.NET Usergroup Rheintal**

c/o Computer Studio Kogoj
 Josefgasse 11, 6800 Feldkirch
<http://usergroups.at/blogs/dotnetusergrouprheintal/default.aspx>
 Kontakt: Herr Thomas Kogoj (thomas@kogoj.com)

.NET User Group Austria

c/o Global Knowledge Network GmbH,
 Gutheil Schoder Gasse 7a, 1101 Wien
<http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx>
 Kontakt: Herr Christian Nagel (ug@christiannagel.com)



Die Java User Group
 Metropolregion Nürnberg
 trifft sich regelmäßig
 einmal im Monat.

Thema und Ort werden über
www.jug-n.de
 bekannt gegeben.

Weitere Informationen
 finden Sie unter:
www.jug-n.de

► Objektorientierte Analyse und Design mit UML und Design Patterns

Methoden und Prinzipien für die Entwicklung von OO-Modellen und die Dokumentation mit der UML
30. Januar – 1. Februar 2012, 11. – 13. April 2012,
1.180,- € (zzgl. 19 % MwSt.)

► Testkonzepte

Software-Tests zur kontinuierlichen Sicherung der Qualität,
13. – 15. Februar 2012, 14. – 16. Mai 2012,
1.315,- € (zzgl. 19 % MwSt.)

► Refactoring

Verbesserung von Design und Quellcode bestehender Software,
27. – 28. Februar 2012, 18. – 19. Juni 2012,
925,- € (zzgl. 19 % MwSt.)

► Web-Anwendungen mit Apache Wicket

Umsetzung von Web-Projekten mit Apache Wicket
29. Feb. – 2. März 2012, 17. – 19. September 2012
1.180,- € (zzgl. 19 % MwSt.)

► JEE Anwendungsentwicklung

Entwicklung moderner, skalierbarer Anwendungen mit der Java Enterprise Edition (JEE)
19. – 23. März 2012,
7. – 11. Mai 2012,
2.095,- € (zzgl. 19 % MwSt.)



Lesen bildet. Training macht fit.

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de



join the
experts
of enterprise infrastructure

Software-Entwickler (m/w) Software-Architekt (m/w)

Arbeiten Sie gerne selbstständig, motiviert und im Team?
Haben Sie gesunden Ehrgeiz und Lust, Verantwortung zu übernehmen?

Wir bieten Ihnen erstklassigen Wissensaustausch, ein tolles Umfeld, spannende Projekte in den unterschiedlichsten Branchen und Bereichen sowie herausfordernde Produktentwicklung.

Wenn Sie ihr Know-how gerne auch als Trainer oder Coach weitergeben möchten, Sie über Berufserfahrung mit verteilten Systemen verfügen und Ihnen Komponenten- und Objektorientierung im .Net- oder JEE-Umfeld vertraut sind, dann lernen Sie uns doch kennen.

Wir freuen uns auf Ihre Bewerbung!

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

Enterprise Open Source Day 2012

IT 2.0 - agil, mobil und unkompliziert

Vorträge und Workshops über neueste Open Source Businesslösungen zu den Themen:

- Portale, Wissensmanagement, Collaboration
- Business Intelligence
- Enterprise Service Bus und Service Oriented Architecture
- Monitoring, Identity Management
- Mobile Applikationen
- Enterprise Search
- E-Commerce

08. Februar 2012

Hotel Hilton
Valznerweiherstraße 200
90480 Nürnberg

Anmeldung und Information unter
www.eosd-2012.de

Aktuelle Partner



Medienpartner



Kooperationspartner



Herbstcampus

Wissenstransfer par excellence

Der **Herbstcampus** möchte sich ein bisschen von den üblichen Konferenzen abheben und deshalb konkrete Hilfe für Software-Entwickler, Architekten und Projektleiter bieten.

Dazu sollen die in Frage kommenden Themen möglichst in verschiedenen Vorträgen besetzt werden: als Einführung, Erfahrungsbericht

oder problemlösender Vortrag. Darüber hinaus können Tutorien die Einführung oder die Vertiefung in ein Thema ermöglichen.

Haben Sie ein passendes Thema oder Interesse, einen Vortrag zu halten? Dann fragen Sie einfach bei info@bookware.de nach den Beitragsinformationen oder lesen Sie diese unter www.herbstcampus.de nach.

3.– 6. September 2012
in Nürnberg

Das Allerletzte

	engl. Personalpronomen	S	H	E
tech. Kraftarm				
Programmierersprache	H			L
B	E	I		
			Im Jargon: Gruppe von	
				italienischer Frauennamen

Dies ist kein Scherz!
 Dieses Kreuzworträtsel wurde tatsächlich in der freien
 Wildbahn angetroffen.

Ist Ihnen auch schon einmal ein Exemplar dieser
 Gattung über den Weg gelaufen?
 Dann scheuen Sie sich bitte nicht, uns das mitzuteilen.

Der nächste KAFFEEKLATSCH erscheint Mitte Januar.



Herbstcampus

Wissenstransfer par excellence

3.– 6. September 2012
in Nürnberg