

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java ist nicht zu bremsen

Mobile statt Cloud

Java Enterprise Edition 7, Seite 8

Morphia, Spring Data & Co.

Java-Persistenz-Frameworks

für MongoDB, Seite 20

Starke Performance

Java ohne Schwankungen, Seite 50

Java und Oracle

Dynamische Reports innerhalb

der Datenbank, Seite 53



iJUG
Verbund

Sonderdruck

20

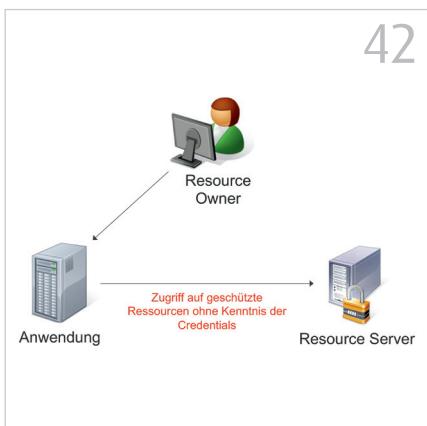


Java-Persistenz-Frameworks für MongoDB, Seite 20



Hibernate und Extra-Lazy Initialization, Seite 38

- | | | |
|--|---|--|
| <p>5 Das Java-Tagebuch
<i>Andreas Badelt,
Leiter der DOAG SIG Java</i></p> <p>8 Java Enterprise Edition 7:
Mobile statt Cloud
<i>Markus Eisele</i></p> <p>14 Source Talk Tage 2013
<i>Stefan Koopal</i></p> <p>15 WebLogic-Grundlagen:
Die feinen Unterschiede
<i>Sylvie Lübeck</i></p> <p>20 Morphia, Spring Data & Co. –
Java-Persistenz-Frameworks für
MongoDB
<i>Tobias Trelle</i></p> <p>26 Noch mehr Web-Apps mit „Play!“
entwickeln
<i>Andreas Koop</i></p> | <p>31 Flexible Suche mit Lucene
<i>Florian Hopf</i></p> <p>35 Automatisiertes Behavior Driven
Development mit JBehave
<i>Matthias Balke und Sebastian Laag</i></p> <p>38 Heute mal extra faul –
Hibernate und Extra-Lazy Initialization
<i>Martin Dilger</i></p> <p>41 Programmieren lernen mit Java
<i>Gelesen von Jürgen Thierack</i></p> <p>42 OAuth 2.0 – ein Standard wird
erwachsen
<i>Uwe Friedrichsen</i></p> <p>47 „Der JCP bestimmt die Evolution
der Programmiersprache ...“
<i>Interview mit Dr. Susanne Cech
Previtali</i></p> | <p>49 Java ohne Schwankungen
<i>Matthew Schuetze</i></p> <p>52 Dynamische Reports innerhalb von
Oracle-Datenbanken
<i>Michael Simons</i></p> <p>56 Universelles Ein-Klick-Log-in mit WebID
<i>Angelo Veltens</i></p> <p>61 Leben Sie schon oder programmieren
Sie noch UIs?
<i>Jonas Helmig</i></p> <p>55 Inserenten</p> <p>66 Impressum</p> |
|--|---|--|



OAuth 2.0 – ein Standard, Seite 42

The screenshot shows the 'MyProfile' website interface. At the top, it says 'Das von MyProfile ausgestellte Client-Zertifikat wurde erfolgreich gespeichert.' Below this is a 'Create / Edit Prof' button. The main form is titled 'New Profile' and contains fields for 'Local username' (maxmuster), 'Full name' (Max Mustermann), and 'Email'. To the right, a modal window displays certificate details: 'Seriennummer: 00:FF:D5:DE:C6:E3:E0:E1:1A', 'Gültig von 08.06.13 15:55:44 bis 06.06.23 15:55:44', and 'Ausgestellt von: CN = MyProfile, OU = MyProfile, O = MyProfile, ST = Essonne, C = FR'. It also mentions 'Gespeichert in: NSS Certificate DB'. Buttons for 'Zertifikatinformationen', 'Abbrechen', and 'OK' are at the bottom of the modal.

Universelles Ein-Klick-Log-in mit WebID, Seite 56

56



dungen auf ein immer niedrigeres Niveau. Leider sind herkömmlichen JVMs Grenzen gesetzt, die durch Unterbrechungen bei der Speicherbereinigung verursacht wer-

den; hierdurch erhöht sich die Latenz selbst dann, wenn man größere Speicherbereinigungsvorgänge hinauszögert, bis das System neu gebootet werden kann.

```
Java -Xmx12g -XX:MaxPermSize=64M -XX:PermSize=32M -XX:MaxNewSize=2g
-XX:NewSize=lg -XX:SurvivorRatio=l28 -XX:+UseParNewGC
-XX:+UseConcMarkSweepGC -XX:MaxTenuringThreshold=0
-XX:CMSInitiatingOccupancyFraction=60 -XX:+CMSParallelRemarkEnabled
-XX:+UseCMSInitiatingOccupancyOnly -XX:ParallelGCThreads=l2
-XX:LargePageSizeInBytes=256m ...

Java -Xms8g -Xmx8g -Xmn2g -XX:PermSize=64M -XX:MaxPermSize=256M
-XX:-OmitStackTraceInFastThrow -XX:SurvivorRatio=2 -XX:-UseAdaptiveSizePolicy
-XX:+UseConcMarkSweepGC -XX:+CMSConcurrentMTEnabled
-XX:+CMSParallelRemarkEnabled -XX:+CMSParallelSurvivorRemarkEnabled
-XX:CMSMaxAbortablePrecleanTime=10000 -XX:+UseCMSInitiatingOccupancyOnly
-XX:CMSInitiatingOccupancyFraction=63 -XX:+UseParNewGC -Xnoclassgc ...
```

Listing 1

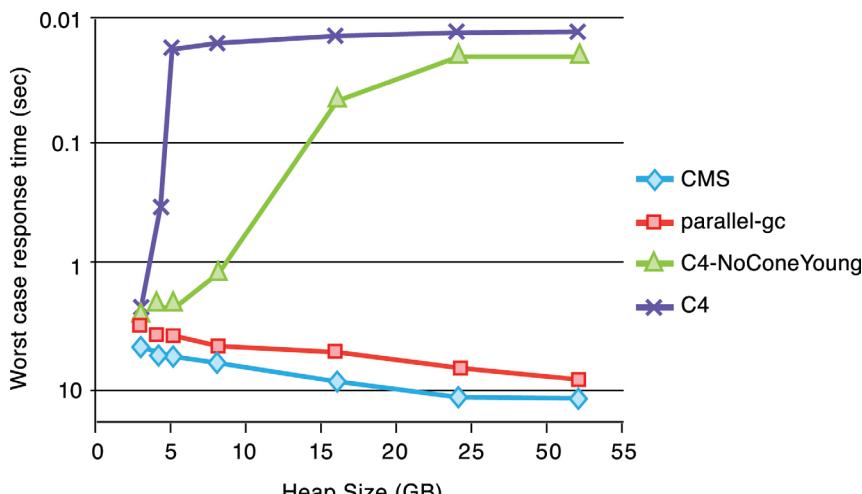


Abbildung 2: Reaktionszeiten im schlimmsten Fall

Die Zing JVM ermöglicht als ungetunes „Out of the box“-System unübertroffene Reaktionszeiten von höchstens zehn Millisekunden. Als einzige handelsübliche JVM koppelt sie Aspekte der Skalenmetrik, wie größere Heaps, mehr Transaktionen oder mehr Anwender, vollständig von der Reaktionszeit ab.

Matthew Schuetze
uroesch@azulsystems.com



Matthew Schuetze ist Produkt-Manager für die Zing-Prouktfamilie bei Azul Systems. Er arbeitet gemeinsam mit Kunden und Partnern an Ideen für neue Features und verfügt über mehr als fünfzehn Jahre Erfahrung bei der Erstellung von Software-Anwendungen und mit Entwicklungs-tools, davon knapp zehn Jahre mit statischen und dynamischen Code-Analyzern für Java und JVMs. Matthew Schuetze hat Ingenieur-Abschlüsse am Massachusetts Institute of Technology (MIT) und an der University of Michigan.

Wenn die „ADF Task Flows“ in JavaServer Faces einfließen, heißen sie „Faces Flows“ – Grundlagen eines neuen Features von JSF 2.2

Die Version 2.2 von JavaServer Faces (JSF) ist mit etlichen neuen Features herausgekommen. Für Entwickler sind im Standard-Framework unter den siebzig neuen Funktionen und Verbesserungen drei besonders interessant: „HTML5 Friendly Markup“, „Resource Library Contracts“ und „Faces Flows“. DOAG Online hat die „Faces Flows“ herausgepickt

und Ed Burns darum gebeten, die Grundlagen dieser Funktionalität zu erläutern.

Lesen Sie den Artikel auf
www.doag.org/go/java-serverfaces



Ed Burns
Consulting Member
of the Technical Staff
Oracle America, Inc.



Dynamische Reports innerhalb von Oracle-Datenbanken

Michael Simons, ENERKO Informatik GmbH

Etliche Java-Entwickler kennen ihre Datenbank nur über eine objektrelationale Abbildung. Umgekehrt ist für viele datenbanknahe Entwickler Java nur die Sprache der Anwendung, die ständig umständliche Statements an die Datenbank übergibt. Allerdings gibt es sehr wohl elegante Zugriffsmöglichkeiten aus Java auf Datenbanken und man kann ebenso objektorientiert in einer Datenbank arbeiten.

Oracle stellt bereits seit 1999 eine native JVM [1] innerhalb der Datenbank zur Verfügung, in der Geschäftslogik in nahezu jeder Form entwickelt und über eine PL/SQL-Schnittstelle anwendungsweit zur Verfügung gestellt werden kann. Anhand der ENERKO-Report-Engine wird gezeigt, wie man SQL, PL/SQL und Java-Datenbank einfach und gewinnbringend verbinden kann.

Im Rahmen eines Oracle-Forms-6i-Ablöseprojekts stellte sich die Frage nach dem Reporting. Oracle Reports ist Teil der Developer Suite und wurde entsprechend mit abgelöst. Da die vorhandene Applikation nicht mit einer Big-Bang-Ablösung umzustellen war, mussten vorhandene und neue Reports gleichermaßen im alten Forms und im neuen Java-SE-Client zur Verfügung stehen. Weiterhin sollte es möglich sein, Reports mit dem vorhandenen Wissen über SQL sowie Excel-Dateien als Vorlagen ohne Java, XML oder ähnliche Kenntnisse erstellen zu können.

Die Datenquellen der vorhandenen Reports konnten als SQL-Statements aus den Report-Definitionen archiviert werden, das Layout wurde in Form von Excel-Dateien extrahiert. Das Ziel war somit klar: Die Datenbank ist der zentrale Speicherort für Report-Definitionen jeglicher Art und die Erzeugung der Reports soll ebenfalls in der Datenbank stattfinden.

Java und Excel

Wenn es um Reporting geht, liegt im Java-Kontext der Gedanke an JasperReports nahe, allerdings konnte man aufgrund der genannten Ziele nicht auf diese Lösung setzen und benutzte stattdessen Apache POI [2]. Die Erzeugung der Arbeitsmappen liegt somit in der eigenen Verant-

```
public class CellDefinition {
    /** The name of the sheet inside the Excel document */
    public final String sheetname;
    /** Column Index (0-based) */
    public final int column;
    /** Row Index (0-based) */
    public final int row;
    /** Cellreference ("A1" notation),
     * only used for output */
    public final String name;
    /** Contains either type or type and a reference cell
     * as datotyp;"SHEETNAME" CELLREFERENCE */
    private final String type;
    /** A string representation of the value */
    public final String value;
}
```

Listing 1: CellDefinition

```
private void addCell(
    final Workbook workbook,
    final Sheet sheet,
    final CellDefinition cellDefinition
) {
    final int columnNum = cellDefinition.column,
        rowNum = cellDefinition.row;

    Row row = sheet.getRow(rowNum);
    if(row == null)
        row = sheet.createRow(rowNum);
    Cell cell = row.getCell(columnNum);
    if(cell != null &&
        cell.getCellType() != Cell.CELL_TYPE_BLANK) {
        cell = fill(workbook, cell, cellDefinition, false);
    } else {
        cell = row.createCell(columnNum)
        cell = fill(workbook, cell , cellDefinition, true);    }
}
```

Listing 2: Erzeugen und Füllen einer Zelle

wortung. Das Modul „Apache POI-HSSF“ ist eine reine Java-Implementierung des „Excel 97-2007“-Formats. POI steht dabei für „Poor Obfuscation Implementation“ und HSSF für „Horrible Spreadsheet Format“. Leider ist diese Erklärung nur noch

in Wikipedia zu finden, „um die Bibliothek für Geschäftsanwendungen attraktiver zu machen, die Humor dieser Art für unangemessen halten“.

Das Usermodel von POI-HSSF unterstützt das Lesen und insbesondere Erzeugen



```

CREATE TYPE t_er_cell_definition AS OBJECT (
    sheetname      VARCHAR2(512),
    cell_column    INTEGER,
    cell_row       INTEGER,
    cell_name      VARCHAR2(64),
    cell_type      VARCHAR2(512),
    cell_value     VARCHAR2(32767),
)

CONSTRUCTOR FUNCTION t_er_cell_definition(
    p_sheetname    VARCHAR2,
    p_cell_column  INTEGER,
    p_cell_row     INTEGER,
    p_cell_type    VARCHAR2,
    p_cell_value   VARCHAR2
) RETURN self AS result
)
/

```

Listing 3: Benutzerdefinierte PL/SQL „Typ t_er_cell_definition“

```

CREATE OR REPLACE
FUNCTION f_say_hello(num_rows NUMBER, p_hello_to IN VARCHAR2) RETURN table_of_
er_cell_definitions pipelined IS
BEGIN
    FOR i IN 0 .. (num_rows - 1) LOOP
        pipe row(
            t_er_cell_definition(
                'Hello',
                0,
                i,
                'string',
                'Hello,' || p_hello_to
            )
        );
    END LOOP;
    RETURN;
END f_say_hello;
/
SELECT cell_column, cell_row, cell_value
FROM table(f_say_hello(5, 'JavaAktuell'));

```

Listing 4: Table function „f_say_hello“

```

FUNCTION f_create_report(
    p_method_name IN VARCHAR2,
    p_template IN BLOB,
    p_args IN t_vargs
) RETURN BLOB IS LANGUAGE JAVA NAME 'de.enerko.reports2.PckEnerkoReports2.
createReport(java.lang.String, oracle.sql.BLOB, oracle.sql.ARRAY) return oracle.sql.BLOB';

```

Listing 5: Call specification „f_create_report“ für „PckEnerkoReports2.createReport“

```

private static String[] extractVargs(final ARRAY arguments) throws SQLException {
    final String[] $arguments;
    if(arguments == null)
        $arguments = new String[0];
    else {
        $arguments = new String[arguments.length()];
        final ResultSet hlp = arguments.getResultSet();
        int i = 0;
        while(hlp.next())
            $arguments[i++] = hlp.getString(2);
    }
    return $arguments;
}

```

Listing 6: Verarbeitung von PL/SQL-Arrays

von Arbeitsmappen und Zellen, die Auswertung von Formeln, benutzerdefinierte Funktionen zur Verwendung in Formeln und vieles mehr. Arbeitsmappen können von Grund auf neu erzeugt oder auf Basis bestehender Arbeitsmappen (Templates) definiert werden.

Apache-POI-HSSF benötigt mindestens eine Java-1.5-Laufzeit-Umgebung. Für das Projekt ist das ein kleiner Nachteil, da erst ab Oracle 11g R1 eine 1.5-JVM in der Datenbank implementiert ist. Falls eine 10er-Datenbank benutzt werden soll, kann beispielsweise das Java-Excel-API [3] eine Alternative darstellen. Der große Nachteil hier ist, dass Formeln zwar gelesen und geschrieben, aber nicht ausgewertet werden.

Apache-POI-HSSF (und die einzige, weitere Abhängigkeit „commons-codec“) müssen als Bibliotheken in die Datenbank geladen sein. Dies kann über das Tool „loadjava“ erfolgen, das Teil des Oracle-Clients ist, oder über das Package „dbms_java“, wenn die Jar-Dateien in einem vom Datenbank-Server lesbaren Verzeichnis liegen. Man hat die Möglichkeit, Java-Quellen oder vorkompilierte Klassen zu laden.

Im Falle von Java-1.5-Code gibt es einige Besonderheiten der Oracle-JVM. Das „For-each“ und einige weitere, insbesondere generische Sprach-Konstrukte sind nicht unterstützt. Quellen mit diesen Konstrukten lassen sich nicht in der Datenbank kompilieren. Werden Bibliotheken und Klassen, die diese Konstrukte benutzen, extern kompiliert, lassen sie sich allerdings sehr wohl verwenden.

Definition einer Zelle

Eine Zelle einer Arbeitsmappe kann mit wenigen Attributen beschrieben werden ([siehe Listing 1](#)). Die Zell-Definition ist bewusst nicht typisiert und der Wert wird als String abgespeichert, um später beim Abruf der Werte aus der Datenbank maximale Flexibilität zu haben. Bereits mit dieser Definition ist es möglich, die Zellen einer Excel-Datei zu füllen ([siehe Listing 2](#)).

Indem die Methode „addCell“ als Teil der Report-Klasse und nicht als Teil der Definition der Zelle implementiert ist, kann man die Report-Klasse gegebenenfalls durch ein Interface ersetzen und mehrere Implementierungen (etwa zusätzlich durch das Java-Excel-API) anbieten.

PL/SQL-Objekte

PL/SQL-Typen können genau wie Java-Klassen Konstruktoren, Typ-Methoden (statische Methoden) und Objekt-Methoden haben. Listing 3 zeigt die Spezifikation des Typs, der eine Zelle einer Arbeitsmappe repräsentiert und mit der Klasse „CellDefinition“ aus Listing 1 korrespondiert.

Der „Object-Relational Developer’s Guide“ [4] bietet eine sehr schöne Übersicht darüber, was Objekte in der Datenbank leisten können. Dieser Typ könnte nun einzeln als „java.sql.Struct“ beziehungsweise als Liste innerhalb eines „java.sql.Array“ an die Report Engine übergeben werden, widersprüche aber der Anforderung, Reports durch Statements oder Funktionen definieren zu können. Daher werden Zell-Definitionen entweder direkt aus SQL-Statements erzeugt („connection.createStatement().executeQuery("")“) oder durch den Aufruf sogenannter „pipelined functions“. Diese sind eine Besonderheit innerhalb von Oracle-Datenbanken: „Pipelined oder Table-Funktionen sind Funktionen, die eine Menge von Zeilen eines bestimmten Typs produzieren, die genau wie physikalische Tabellen abgefragt werden können“ [5]. Listing 4 zeigt ein typisches „Hello World“ und seinen Aufruf.

Java-Funktionen aus SQL und PL/SQL aufrufen

Innerhalb der Datenbank geschieht der Aufruf eines Java-Programms ähnlich wie in einer „normalen“ JVM, nämlich über eine statische Methode, da vor dem Aufruf noch kein Objekt existiert. Der Aufruf geschieht dabei über eine „Call Specification“, in der der vollständig qualifizierte Name der statischen Java-Methode sowie die vollständig qualifizierten Typen der Parameter angegeben werden müssen (siehe Listing 5).

In diesem Beispiel sieht man, dass auch problemlos benutzerdefinierte PL/SQL-Typen („t_vargs“) an Java übergeben werden können. Listings 6 und 7 zeigen die Verarbeitung beziehungsweise die Erzeugung von benutzerdefinierten PL/SQL-Typen innerhalb von Java.

Report-Erzeugung

An dieser Stelle stehen alle Teile zur Verfügung, die zur Erzeugung von Reports benötigt werden: Definitionen der Zellen auf

```
private static ARRAY convertListOfCellsToOracleArray(final List<CellDefinition> cellDefinitions)
throws SQLException {
    final StructDescriptor resultStruct = StructDescriptor.createDescriptor("T_ER_CELL_DEFINITION", connection);
    final ArrayDescriptor arrayDesc = ArrayDescriptor.createDescriptor("TABLE_OF_ER_CELL_DEFINITIONS", connection);

    final STRUCT[] rv = new STRUCT[cellDefinitions.size()];
    int i=0;
    for(CellDefinition cellDefinition : cellDefinitions)
        rv[i++] = new STRUCT(resultStruct, connection, cellDefinition.toSQLStructObject());
    return new ARRAY(arrayDesc, connection, rv);
}
```

Listing 7: Erzeugung von PL/SQL-Arrays mit benutzerdefinierten Typen

```
try {
    connection = (OracleConnection) DriverManager.getConnection("jdbc:default:connection");
    reportEngine = new ReportEngine(connection);
} catch (SQLException e) {
    throw uncheck(e);
}

public class ReportEngine {
    private final OracleConnection connection;
    private final Map<String, FreeRefFunction> customFunctions = new HashMap<String, FreeRefFunction>();

    public ReportEngine(OracleConnection connection) {
        this.connection = connection;
        this.addCustomFunction("Enerko_NormInv", new NormInv());
    }
}
```

Listing 8: „ReportEngine.java“

```
DECLARE
    v_report BLOB;
BEGIN
    -- Create the report
    v_report :=
        pck_enerko_reports2.f_create_report(
            'f_say_hello', t_vargs('5', 'JavaAktuell')
        );
    -- Store it into a server side file
    pck_enerko_reports2.p_blob_to_file(
        v_report, 'enerko_reports', 'f_say_hello.xls'
    );
END;
/
```

Listing 9: Erzeugung eines Reports und Speicherung als BLOB

```
SELECT *
FROM table(
    pck_enerko_reports2.f_eval_report(
        f_say_hello, null, t_vargs('5', 'JavaAktuell')
    )
) src;
```

Listing 10: „On the fly“-Evaluierung von Reports/Arbeitsmappen

	A	B	C	D	E
1	Hello, JavaAktuell				
2	Hello, JavaAktuell				
3	Hello, JavaAktuell				
4	Hello, JavaAktuell				
5	Hello, JavaAktuell				
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

Abbildung 1: Das Ergebnis des Hello-World-Reports aus Listing 4

Michael Simons
michael@simons.ac



SQL, PL/SQL und Java-Ebene, Datenquellen für Statements und Funktionen auf Java-Ebene sowie eine Factory-Klasse, die auf Apache-POI-HSSF basierende Reports erzeugt und im passenden Format ausgibt (siehe Listing 8). Die Factory-Klasse hält dabei nicht nur die interne JDBC-Verbindung zur Datenbank, sondern auch eine Map mit benutzerdefinierten Excel-Funktionen, die als Java-Klassen implementiert sind.

Das Package „pck_enerko_reports2“ stellt darüber hinaus überladene Funktionen zur Verfügung, die einen Report auf Basis eines Templates erzeugen können. Ein Template kann jede Excel-Arbeitsmappe sein, die als BLOB innerhalb der Datenbank zur Verfügung steht.

In einigen Fällen ist keine Excel-Arbeitsmappe als Ausgabe erwünscht, sondern ein Report, der wie eine Tabelle abgefragt werden kann. In diesem Fall kommen wieder „pipelined functions“ ins Spiel. Benutzerdefinierte Typen können, wie in Listing 7 gezeigt, auf Java-Ebene instanziert und genutzt werden, um neu erstellte Reports oder vorhandene Excel-Arbeitsmappen wie physikalische Tabellen abfragen zu können (siehe Listing 10).

Fazit

Die ENERKO-Report-Engine ist seit Anfang 2010 in mehreren Projekten, unter anderem bei Vertriebsmanagement-

Aufgaben im Strom- und Gasbereich, im Einsatz und erzeugt komplexe Reports mit teilweise weit mehr als 50.000 Zellen, aufwändigen Grafiken und Makros. Sie wird zudem in der Preiskalkulation verwendet und dient zur Auswertung von Rechen-Schemata, die als Excel-Arbeitsmappen zur Verfügung gestellt werden. SQL-Entwickler nutzen die Engine gern, da sie die gewohnte Arbeitsumgebung nicht verlassen müssen, und die Kunden freuen sich über Auswertungen innerhalb eines bekannten Werkzeugs. Es hat sich also gelohnt, um die Ecke zu denken, gewohnte Bahnen zu verlassen und mit der Report-Engine zwei Welten zu verbinden.

Im Rahmen des Artikels ist die Report Engine vollständig überarbeitet und von Altlästen befreit worden. Sie steht nun unter Apache-2-Lizenz auf GitHub [6] zum Download bereit; Jar-Dateien zur direkten Verwendung innerhalb der Datenbank stehen im Central Repository zur Verfügung. Die Schnittstelle für benutzerdefinierte Funktionen soll in Zukunft noch erweitert werden. Darüber hinaus ist geplant, zusätzliche Infrastruktur-Klassen für die Verwaltung von Vorlagen zu implementieren.

Links

- [1] Oracle Database: http://en.wikipedia.org/wiki/Oracle_database#History
- [2] Apache POI: <http://poi.apache.org>
- [3] Java Excel API: <http://jexcelapi.sourceforge.net>

Michael Simons ist Software-Architekt bei ENERKO Informatik [7] in Aachen und entwickelt dort GIS-, EDM- und Vertriebsmanagement-Systeme für Stromnetz-Betreiber und Energie-Lieferanten. Dabei setzt er auf Java SE, Java EE sowie Oracle-Datenbanken. Er bloggt über Lösungen für tägliche und nicht alltägliche Probleme der Entwicklung auf <http://info.michael-simons.eu>.

Unsere Inserenten

aformatik Training und Consulting GmbH & Co. KG, www.aformatik.de	S. 3
Eclipse Foundation Inc. www.eclipse.org/	S. 19
TEAM GmbH www.team-pb.de/	S. 25
DOAG e.V. www.doag.org	U2
iJUG e.V. www.ijug.eu	U3
Trivadis GmbH, www.trivadis.com	U4



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. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer



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 Rechnung. 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.