



Das iJUG Magazin

Java aktuell

Herbst 2011

Java aktuell
Magazin der Java-Community

Java überall

- Neu: Java SE7
- Interview mit Patrick Curran, Vorsitzender des JCP
- Für Android entwickeln
- Testen mit Arquillian
- Suchen mit Apache Solr



iJUG
Verbund

Erfahrungen, Ideen und Lösungen für Java-Entwickler

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

Sonderdruck

04/2011



- 3 Editorial
- 5 Die lange Reise von Java 7
Markus Eisele, msg systems ag
- 8 „Unbedingt die Specs lesen und Feedback geben ...“
Interview mit Patrick Curran, Vorsitzender des JCP
- 11 Das Java-Tagebuch
Andreas Badelt, DOAG Deutsche ORACLE-Anwendergruppe e.V.
- 15 Java überall
Oliver Szymanski, Source-Knights.com, stellv. Vorstandsvorsitzender des iJUG
- 17 „Java muss sich neuen Einsatz-Szenarien wie Cloud und Mobile Computing stellen ...“
Interview mit Dr. Mark Little, Red Hat
- 19 Arquillian
Frederik Mortensen
- 22 Suchen mit Apache Solr
Peter Karich, Pannous GmbH
- 26 „Ich denke, die Java-Community ist wie eine große Familie ...“
Interview mit Michael Hüttermann, Java User Group Köln
- 28 Android – Java macht mobil
Andreas Flügge, object systems GmbH
- 31 Hibernate im Projekteinsatz
Dirk Mahler, buschmais GbR
- 34 Semantisch-orientierte Programmierung mit Java
Oliver Böhm
- 37 Slice – Unterstützung für verteilte, partitionierte und heterogene Datenbanken mit OpenJPA
Bernd Müller, Ostfalia Hochschule für angewandte Wissenschaften, sowie Harald Wehr, MAN Truck & Bus AG
- 40 NetBeans Platform 7
gelesen von Jürgen Thierack
- 41 XPages – Ein neues Framework zur Entwicklung von Web-Anwendungen
Dr. Rolf Kremer, PAVONE AG
- 45 „Bereits jetzt zählen wir zu den führenden Java-Magazinen im deutschsprachigen Raum ...“
Interview mit Fried Saacke, Vorstandsvorsitzender des iJUG
- 46 Leichtgewichtige Authentifizierung mit OpenID
Sebastian Glandien, Acando GmbH
- 51 Java-Problem-Determination mit der IBM Support Assistant Workbench
Marc Bauer, IBM Deutschland GmbH
- 54 Java EE 7 – eine Reise in die Wolken
Peter Doschkinow, ORACLE Deutschland B.V. & Co. KG
- 57 Varianten-Entwicklung in 3D mit Object Teams
Dr. Stephan Herrmann, GK Software AG
- 60 Unbekannte Kostbarkeiten des SDK Heute: Der Service-Loader
Bernd Müller, Ostfalia
- 62 Rich Client Frontends für umfangreiche Unternehmensanwendungen
Björn Müller, CaptainCasa
- 61 Inserenten
- 53 Impressum



Kleiner Exkurs darüber, wo wir Java direkt oder indirekt überall antreffen, Seite 15

Dies ist ein Sonderdruck aus der Java aktuell. Er enthält einen ausgewählten Artikel aus der Ausgabe 04/2011. Das Veröffentlichen des PDFs bzw. die Verteilung eines Ausdrucks davon ist lizenzfrei erlaubt. Weitere Informationen unter www.ijug.eu



Rich Client Frontends für umfangreiche Unternehmensanwendungen

Björn Müller, CaptainCasa

Der Artikel zeigt, wie eine Community mittelständischer, vorwiegend deutscher Softwarehäuser die Frontends für ihre Anwendungen baut: über ein Framework, in dem vorne im Client eine Java-Swing-basierte Benutzeroberfläche läuft und die Anwendung selbst hinten im Server innerhalb einer JSF-Umgebung abgebunden wird. Das Community-Framework nennt sich „CaptainCasa Enterprise Client“, wird seit 3 Jahren mannigfaltig verwendet und steht im vollen Umfang kostenfrei zur Verfügung.

Swing im Client? Ist das nicht komplett „out of Hype“? Im Server JSF? Ist das nicht „höllisch komplex“? Und ist JSF nicht ein Framework für HTML-basierte Frontends? Alles berechnete Fragen. Es gehört schon einiges an Mut dazu, in der Hype-getriebenen Welt der UI-Technologien heute noch die Swing-Fahne zu hissen. Aber es gibt gute und vernünftige Gründe, dies auch und gerade heute zu tun! Und wenn man es schon macht, dann auch richtig, also mit einer vernünftigen, effizienten Architektur, die auf ihr Einsatzgebiet zugeschnitten ist.

Es geht nicht um „Apps“

In erster Linie reden wir hier über Benutzeroberflächen für umfangreiche Unternehmensanwendungen. Deren Bediener sind in der Regel Sachbearbeiter mit recht hohen Anforderungen an die Bedien-Performance und Bediener-Ergonomie (die berühmte Tastatursteuerung ...). In der Regel gibt es Hunderte von unterschiedlichen Dialogen in einer solchen Anwendung – zur Erfassung von Steuer- und Stammdaten, zur operativen Nutzung des Systems, zur Analyse innerhalb des Reportings etc. Ganz wichtig: Die Anwendungen haben einen Lebenszyklus, der gut und gerne 15 Jahre überschreitet: heute entwickelt, in zwei Jahren im Vertrieb, fünf Jahre vertrie-

ben, acht Jahre Nutzungszeit beim Endkunden. Nutzungs- und damit Supportzeiten von weit über zehn Jahren kommen so ganz schnell zusammen. In einem solchen Umfeld fällt es schwer, zu schnell auf aktuelle Hype-Technologien zu springen. Bei mehr als zehn Jahren Lebenszyklus kommt und geht mancher Hype – gerade im Umfeld von UI-Technologien-Frameworks, die heute „in“ und morgen wieder „out“ sind – für ihre Anwendung muss es aber weitergehen.

Es fällt auch schwer, ganz ohne Hemmungen auf HTML/Javascript-basierte Ansätze zu setzen: Zu bescheiden ist immer noch zum Beispiel die Tastatur-Unterstützung, zu hoch ist immer noch der Aufwand, ein und dieselbe Oberfläche auf verschiedenen Browsern in gleicher Qualität zu Verfügung zu stellen. Wenn ein Dialog beim Endkunden nicht läuft, dann ist der Anwendungshersteller immer in der Pflicht, das Problem zu lösen – welche Framework-Schicht darunter auch immer das Cross-Browser-Management behandeln soll.

Warum nicht Java-Swing?

Vor dem Hintergrund der langen Lebenszyklen und der Ergonomie-Anforderungen liegt es nun wieder wesentlich näher, einen

Blick auf Java-Swing zu werfen: Swing ist eine „gut abgehangene“, mittlerweile sehr performante und stabile UI-Umgebung. Swing-Anwendungen gibt es zu Tausenden, gerade im industriellen Einsatzbereich. Dort sind sie meist die „Arbeitsstiere“ – oft nicht besonders attraktiv, aber sie laufen und laufen. Mit dem Java Update 1.6.10 hat sich in der Laufzeitumgebung für Swing-Oberflächen (Applet, Webstart) sehr viel getan – zum Positiven.

Natürlich gibt es auch Nachteile, die bei Java-Swing auf der Hand liegen: Zum einen ist dies die Notwendigkeit der Installation eines Java-Client-Plug-ins. Die Installation ist problemfrei, muss aber mit der System-Administration abgesprochen werden. Nicht vergessen: Das Nutzungsumfeld, von dem wir hier sprechen, ist das von Sachbearbeitern; es geht nicht um Anwendungen, die sofort bei jedem anonymen Web-User laufen müssen.

Zum anderen sind diese Nachteile bekannt: Swing ist für UI-Entwickler zwar „normal komplex“. Swing ist aber definitiv nicht der richtige Level, den ein Anwendungsentwickler benötigt, um effizient einen Dialog nach dem anderen zu entwickeln. Der Aufwand in Swing, wirklich schöne Oberflächen zu bauen, ist recht groß. Man muss schon (und leider) einige



Kniffe beherrschen, um beim Benutzer einen Wow-Effekt hervorzurufen. Schade ist: Swing kann praktisch alles, um diese Wow-Effekte hervorzurufen. Aber man muss wissen, wie es geht, und es dann machen. Schließlich: Swing ist eine reine Umgebung zur Gestaltung von Oberflächen. Ob die Anwendung gleich auf dem Client läuft, ob sie zentral läuft – all dies liegt nicht in der Verantwortung von Swing. Swing per se ist also keine „Rich-Client-Umgebung“, weil die Art und Weise der Anbindung einer Server-seitigen Anwendung nicht festgelegt ist.

Frontend-getriebene Architektur?

Nun wurde in der Community aus recht pragmatischen Gründen ein Java-Swing-basierter Client-Ansatz gewählt. Und nun kommt die spannendste Frage: „Wie wird die Client-Oberfläche an die Server-seitige Anwendungslogik angebunden?“

Die typische, unbedachte Architektur ist eine, die an die Fat-Client-Implementierung der 1990er Jahre erinnert: Vorne im Client wird das GUI programmiert, das über Remote-Schnittstellen (wie Web-Services) auf die Server-seitige Anwendungslogik zugreift. Dieser Frontend-getriebene Ansatz erscheint „natürlich“: Im Rahmen der GUI-Entwicklung hat man kompletten Durchgriff auf alle Features der GUI-Umgebung. Und das Backend muss sowieso Schnittstellen nach außen zur Verfügung stellen, nicht nur für GUI-Zwecke. Aber es gibt entscheidende architektonische Nachteile, die dann auftreten, wenn die Anzahl der Dialoge steigt: Die Anzahl der Schnittstellen zwischen GUI-Client und Anwendungsserver steigt beständig. Jedes Komfort-Feature im Frontend geht in der Regel einher mit dem Aufruf einer neuen Schnittstelle zum Server hin.

Dazu ein Beispiel: die Erfassung einer Bestellung erfordert zunächst die typische „CRUD-Schnittstelle“ (Create, Retrieve, Update, Delete) für das Objekt „Bestellung“. Nun soll der Benutzer aber bei der Eingabe auch die Verfügbarkeit der bestellten Ware sehen. Also wird entweder die CRUD-Schnittstelle erweitert oder eine Schnittstelle „Verfügbarkeit prüfen“ implementiert und vom GUI aufgerufen. Nun soll dem Benutzer auch beim Eintippen von Positionen automatisch der Preis berechnet werden.

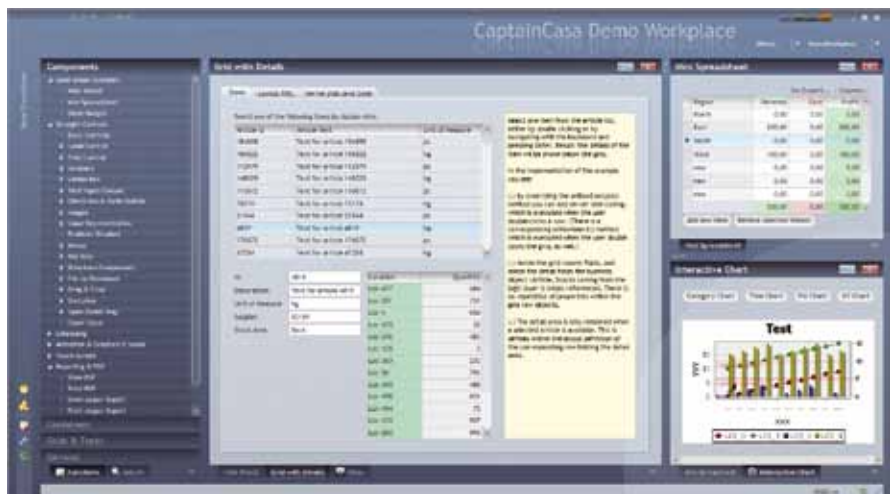


Abbildung 1: Beispieloberfläche CaptainCasa Enterprise Client

Wiederum wird eine neue Schnittstelle eingerichtet. Und so weiter ...

Die hohe Anzahl von Schnittstellen wird verschärft durch die Weiterentwicklung der Anwendung. Es entstehen neue Versionen von Schnittstellen; die alten müssen aber weiter unterstützt werden.

Neben dem immer größer werdenden Aufwand für die Implementierung und das Management dieser Schnittstellen gestaltet sich auch die Implementierung des UI-Clients immer schwieriger: Es muss vermieden werden, dass bei einer einzelnen Benutzeraktion („Benutzer drückt einen Button“) eine Vielzahl von Schnittstellenaufrufen zum Server hin gemacht wird. Jeder Aufruf kostet eine gewisse Latenzzeit im Netzwerk – die Antwortzeit beim Benutzer wird bestimmt durch die Summe aller Latenzzeiten aller Aufrufe. Es tritt ein Phänomen auf, das man aus den 1990ern kannte: Im lokalen, schnellen Netzwerk (LAN) funktioniert die Anwendung gut, im langsameren Netzwerk (WAN) funktioniert sie nicht mehr, beziehungsweise äußerst träge.

Server-getriebenes UI

Der natürlich erscheinende, Frontend-getriebene Architekturansatz hat also seine Grenzen, die mit zunehmendem Umfang der Implementierung einer komplexen Anwendung zutage treten. Wie aber sieht eine Alternative aus? Ganz einfach: Im Frontend wird nicht mehr explizit codiert – es gibt also „vorne“ keine GUI-Anwendungsentwicklung mehr. Stattdessen wird ein vom Server aus gesteuerter generischer

GUI-Client geschrieben unter dem Motto: Der GUI-Client erhält vom Server eine XML-Layoutbeschreibung und zeigt diese an. Nun tätigt der Benutzer seine Eingaben und Aktionen. „Irgendwann“ – nicht bei jedem Druck einer Taste, aber beispielsweise beim Drücken eines Buttons – gehen die im Layout geänderten Daten und die Aktion zum Server. Dort werden sie verarbeitet und der Server schickt eine aktualisierte XML-Layoutbeschreibung zum Client zurück.

Dieses Verarbeitungsprinzip kennen wir bereits von den grün-schwarzen Terminals aus den 1980er Jahren (3270-Terminals, da ohne XML) und natürlich auch vom normalen HTML-basierten Browser. Es ist das Prinzip eines „Thin Clients“, manche nennen es auch „Formular-Processor“.

Nun kommt aber ein entscheidendes Merkmal: Wie smart ist der XML-Austausch von Layoutbeschreibungen, wenn es um die Übertragung von Änderungen am Layout geht? Wenn sich in einem Dialog Teile des Layouts ändern, etwa weil durch eine Benutzeraktion weitere Daten sichtbar werden? HTML kennt hier prinzipiell nur den Ansatz, das gesamte Layout noch einmal zu senden – es findet also keinerlei „Delta-Verarbeitung“ statt. Und es muss viel AJAX/JavaScript-Technologie zum Einsatz kommen, um dieses nachträglich einzupflegen.

Es gilt also hier, beim Austausch von Layout-Beschreibungen von vornherein ein klares „Delta-Konzept“ in der Kommunikation zu verankern. Wenn sich an einer Oberfläche auf dem Weg vom Client zum Server

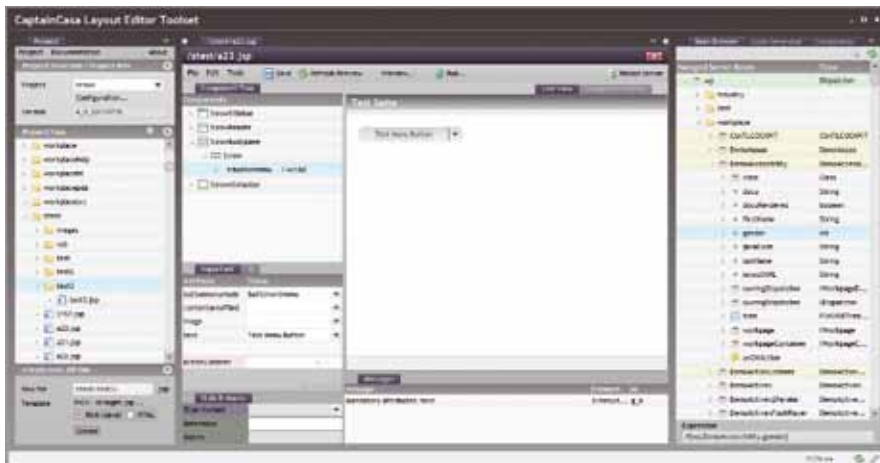


Abbildung 2: Layout-Editor zur WYSIWYG-Erstellung von Layouts

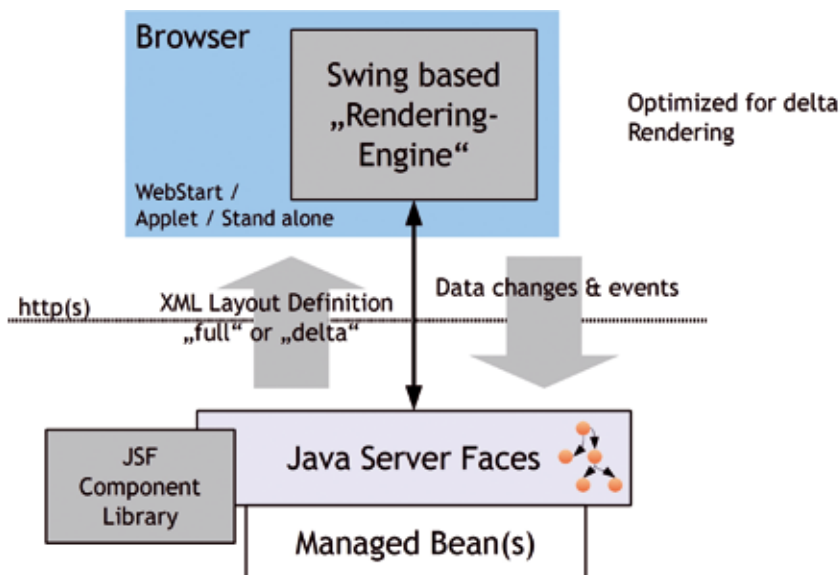


Abbildung 3: Swing-JSF-basierter Thin Client

und zurück nur wenig ändert, dann darf auch nur wenig kommuniziert werden. Dies bedeutet eine signifikante Reduktion des ausgetauschten Datenvolumens und auch eine signifikante Steigerung der Performance am Client. Schließlich muss nicht ein Re-Rendering einer kompletten Seite erfolgen, sondern es müssen nur die Änderungen eingearbeitet werden.

Bevor es mit Technologie weitergeht, wird noch einmal die Server-getriebene Denkweise dieser Frontend-Architektur betont: Im Client selbst erfolgt keine explizite Oberflächen-Entwicklung durch die Anwendung – diese findet komplett im Server statt. Der Client ist eine reine „Rendering-Engine“. Der Client weiß nicht, ob es sich bei einem dargestellten

Formular um eine Bestellerfassung oder die Pflege eines Artikels handelt – er rendert das Layout aus und gibt zu definierten Zeitpunkten nach hinten kund, was der Benutzer gerade macht.

Aus Entwicklungssicht bedeutet dieser Server-getriebene Ansatz eine enorme Vereinfachung – es findet nur noch Server-seitige Anwendungsentwicklung statt. Es gibt keinen Zwang mehr, jede „Kleinigkeit“ als Schnittstelle nach außen zu legen, damit sie von der Oberfläche angezeigt werden kann. Die Frontend-Entwicklung läuft quasi im Server und setzt direkt auf lokalen Schnittstellen der Anwendung auf.

Diese Architektur löst viele Probleme der Frontend-getriebenen Architektur: Es

gibt nur noch eine Schnittstelle zwischen Client und Server – nämlich die Übertragung der Layout-Daten. Das Prinzip, das eine Nutzeraktion auch nur zu einem Roundtrip führen darf, ist automatisch eingehalten. Und: Fehlerkorrekturen und Weiterentwicklungen in der Anwendung führen nicht zu einem Re-Deployment des Clients für alle Benutzer, da die Änderungen sich alle auf dem Server abspielen.

Bei einer „Frontend-getriebenen“ Architektur ist der Startaufwand klein. Für ein Frontend auf Basis von Java-Swing sind lediglich eine Entwicklungsumgebung zur Java-Swing-Programmierung sowie eine Schnittstellen-Technologie nach hinten erforderlich – schon kann es mit dem ersten Dialog losgehen.

Bei einer „Server-getriebenen“ Architektur ist der Startaufwand zunächst höher: Man braucht die „Rendering Engine“ im Client, diese muss – optimalerweise – über http/s an den Server angebunden sein. In diesem ist eine Server-seitige Dialogverarbeitung zu implementieren, die dann wiederum die eigentliche Anwendung anbindet – eine Menge von aufeinander abzustimmenden Komponenten, die erst einmal zur Verfügung stehen müssen.

CaptainCasa Enterprise Client

Genau diese Infrastruktur ist nun eine, die in der CaptainCasa-Community über eine Vielzahl von Softwarehäusern gemeinschaftlich erarbeitet, genutzt und aktiv gepflegt wird. Im Vorfeld wurde ja bereits erwähnt, dass die Client-Technologie der Wahl „Java-Swing“ heißt. Die beschriebene Rendering-Engine im Client ist also in Java-Swing implementiert.

Interessant ist nun die Frage, wie diese Rendering-Engine an eine Server-seitige Anwendung angeschlossen ist – irgendwo muss ja die XML-Layoutbeschreibung für den Client erstellt werden und irgendwo müssen ja Ereignisse vom Client im Server verarbeitet werden. Hier nun kommt Java Server Faces (JSF) ins Spiel. JSF ist eine Standardtechnologie, die nichts Anderes macht als eine Server-seitige Dialogverarbeitung. Dialoge werden im Server zur Laufzeit als Objektbaum gehalten. Im Falle von HTML entsteht aus dem Objektbaum eine HTML-Seite, indem der Baum



rekursive abgearbeitet wird und jede Komponente des Baums ihren Anteil an HTML erstellt.

JSF ist als Standard nicht auf HTML festgelegt – und das ist gut so. Denn bei CaptainCasa rendert sich der Server-seitige Komponentenbaum nicht in HTML aus, sondern er erstellt genau das XML, das wiederum der Swing-basierte Client benötigt. Beim rekursiven Erstellen des XMLs auf dem Server wird eine Delta-Ermittlung durchgeführt, die dafür sorgt, dass das zum Client gesendete XML nur die Änderungen an einem Layout enthält – und nicht jeweils das gesamte Layout übermittelt wird.

JSF hat den riesigen Vorteil, Standard zu sein. Fragen der Nutzer nach Skalierbarkeit, Failover-Konzepten, Deployment etc. werden über einen Standard beantwortet. Auf dem Server läuft nichts anderes als Standard.

JSF ist komplex, weil es viele Aufgaben flexibel löst und weil es einen ganzen Blumenstrauß von mehr oder minder wichtigen Seitenaspekten enthält, in dem man sich erst einmal zurecht finden muss. JSF darf deswegen nicht ungefiltert an eine Anwendungsentwicklung gegeben werden.

Im Rahmen des CaptainCasa Enterprise Clients wird JSF automatisch gefiltert: Die Anwendungsentwicklung erstellt Seiten über einen WYSIWYG-Editor. In diesem ist das XML-Layout definiert, über Expressions erfolgt die Bindung zu Server-seitigen Anwendungs-Dialogobjekten. Layoutbeschreibung und Dialogobjekt sind modular und multipel verwendbar, das heißt sie können als wiederverwendbare Einheiten in anderen Oberflächen verwendet werden.

Fazit

Wenn es um umfangreiche, langlebige Unternehmensanwendungen geht, sollte man sich gründlich fragen, ob der UI-Hype von heute Grundlage einer Architektur der nächsten zehn Jahre sein sollte. Java-Swing als bewährte UI-Umgebung ist aus diesem Betrachtungswinkel immer noch sehr aktuell. Bei der Anbindung des UI-Clients zum Server hin sollten Sie sich bewusst sein, ob Sie eine Frontend-getriebene (Fat Client) oder eine Server-getriebene (Thin Client) Architektur wählen. Pauschal gilt: Je umfangreicher eine Anwendung wird, desto mehr liegen die Vorteile auf Seiten des Server-getriebenen Ansatzes.

HTML-basierte Anwendungen, sondern auch für generische Swing Clients.

Björn Müller
bjoern.mueller@captaincasa.com

Layout Definition (JSF):

```
<t:rowbodypane>
  <t:row>
    <t:label text="Your Name" width="120" />
    <t:field text="#{DemoHelloWorld.name}" />
  </t:row>
  <t:row>
    <t:coldistance width="120" />
    <t:button actionListener="#{DemoHelloWorld.onHello}"
      text="Hello!" />
  </t:row>
  <t:rowdistance height="50" />
  <t:row>
    <t:label text="Result" width="120" />
    <t:field background="#F0F0F0" enabled="false"
      text="#{DemoHelloWorld.output}" width="100%" />
  </t:row>
</t:rowbodypane>
```

Zugehöriges, Server-seitiges Java Programm (Managed Beans):

```
public class DemoHelloWorld
{
  String m_name;
  String m_output;

  public void setName(String value) { m_name = value; }
  public String getName() { return m_name; }

  public String getOutput() { return m_output; }

  public void onHello(ActionEvent ae)
  {
    if (m_name == null)
      m_output = „No name set.“;
    else
      m_output = „Hello World, „+m_name+“!“;
  }
}
```

Listing 3

Hierbei trifft man natürlich keine Pauschal-Entscheidung für alle Oberflächen einer Anwendung, sondern hat seine Kern-Dialoge im Blick – diejenigen, die einen mitsamt der Anwendung über den gesamten Lebenszyklus begleiten werden. Und: Java Server Faces ist eine verlässliche, standardisierte Umgebung für die Server-seitige Dialogverarbeitung, nicht nur für HTML-basierte Anwendungen, sondern auch für generische Swing clients.

Björn Müller

bjoern.mueller@captaincasa.com

Björn Müller arbeitete zunächst zehn Jahre in der Anwendungsentwicklung, Basisentwicklung und Architekturentwicklung für SAP. 2001 erfolgte die Gründung der Casabac Technologies GmbH als Pionier im Bereich von Rich Internet Applications auf Basis von Client-Scripting (AJAX). 2007 gründete er die CaptainCasa Community, eine Verbindung mittelständischer, deutschsprachiger Softwarehäuser mit dem Fokus auf Rich Client Frameworks für umfangreiche, langlebige Anwendungssysteme.





Bestellen eines kostenlosen Exemplares der Zeitschrift Java aktuell

Anschrift:

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

ggf. Rechnungsanschrift

E-Mail

Telefonnummer

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

Jetzt Abonnement sichern:

- Abonnement Newsletter: Java aktuell – der iJUG-Newsletter, kostenfrei
- Java aktuell – das iJUG-Magazin Abo: vier Ausgaben zu 18 Euro im Jahr

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

Senden Sie das ausgefüllte Formular an:

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

oder faxen Sie es an:

0700 11 36 24 39

oder bestellen Sie online:

go.ijug.eu/go/abo

*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 Widerrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.

Impressum

Herausgeber:
Interessenverbund der Java User
Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 0700 11 36 24 38
www.ijug.eu

Verlag:
DOAG Dienstleistungen GmbH
Fried Saacke, Geschäftsführer
info@doag-dienstleistungen.de

Chefredakteur (VisdP):
Wolfgang Taschner,
redaktion@ijug.eu

Chefin von Dienst (CvD):
Carmen Al-Youssef,
office@ijug.eu

Titel, Gestaltung und Satz:
Claudia Wagner,
DOAG Dienstleistungen GmbH

Anzeigen:
CrossMarkeTeam, Ralf Rutkat,
Doris Budwill
redaktion@ijug.eu

Mediadaten und Preise:
[http://www.ijug.eu/images/
vorlagen/2011-ijug-mediadaten_
java_aktuell.pdf](http://www.ijug.eu/images/vorlagen/2011-ijug-mediadaten_java_aktuell.pdf)

Druck:
adame Advertising and Media
GmbH Berlin
www.adame.de

Java aktuell – das Abo
4 Ausgaben für 18 Euro