KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

01/2013

Jahrgang 6

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

Früherziehung

Meine Tochter hatte in der siebten oder achten Klasse ein Jahr lang Informatik. Ich kann mich zwar nicht mehr richtig an das Schuljahr erinnern, aber ich weiß noch sehr gut, wie sehr ich mich über den Inhalt des Unterrichts aufgeregt habe.

Unter Informatik scheint man an Schulen wohl das Arbeiten mit Microsoft-Produkten zu verstehen. Zwar klagen die Schulen immer darüber, dass sie zu wenig Geld zur Verfügung haben, aber für Microsoft Office ist immer genug Geld da. Zu allem Überfluss bekommen dann die Kinder auch noch Hausaufgaben auf, die mit MS Office zu erledigen sind. Als hätte dort noch nie jemand etwas von *Open Source* gehört.

Im Unterricht wurde sogar „programmiert". In *HTML*! Ich hatte damals der Versuchung widerstanden, mein Kind darüber aufzuklären, dass ich unter einem Programm eher das Gestalten dynamischer Abläufe anstatt von statischer Deklaration verstehe. Aber nachdem der Volksmund auch von HTML-Programmierung spricht, lohnt es sich in der Regel nicht Unbeteiligte über derartige Spitzfindigkeiten aufzuklären.

Das ist deswegen so erwähnenswert, weil Programmierung doch so viel Spaß machen kann und darüber hinaus das Denken auf eine ganz bestimmte Art und Weise schult. Programmieren wird in Zukunft ja auch immer wichtiger und hilft enorm dabei, Zusammenhänge bei den Computer-gestützten Werkzeugen zu erkennen. Schließlich sind die Dinge nicht mehr ganz so einfach wie das Programmieren des Videorecorders – mal ganz unabhängig davon, dass es die ja fast gar nicht mehr gibt.

Im Internet bin ich auf eine Seite gestoßen, die behauptet, dass in den USA im Jahr 2020 etwa 400 000 Informatiker die Universitäten verlassen, aber dem gegenüber 1,4 Millionen freie Stellen stehen werden. Das ist natürlich ein Problem, vor allem deswegen, weil die Rechner überall Einzug halten und immer individueller eingestellt werden müssen.

Zum Beispiel definieren sich Autos heute kaum noch durch ihr Inneres oder Äußeres, sondern durch ihre elektronischen Systeme. Vom Navigationssystem, übers Radio bis hin zum Bremskraftverstärker muss Software geschrieben werden, die besser keine groben Fehler macht. Beim Radio wären die ja fast noch egal, aber das Navi könnte einen in eine Einbahnstraße schicken, worauf möglicherweise die Bremsen dann ihren Dienst versagen.

Eigentlich wäre das Programmieren ganz einfach. So könnte man mit Textausgaben anfangen, die im nächsten Schritt an Bedingungen geknüpft werden und schließlich in Schleifen bearbeitet werden. Selbst das ereignisgetriebene Programmieren hat seine Reize, und so könnten relativ schnell grafische Applikation gebastelt werden. Oder aber man lässt die Kinder mit virtuellen Schildkröten oder Robotern experimentieren.

Programmieren, gleich von Anfang an in der Grundschule, als Hauptfach neben Deutsch und Mathematik; später als Wahlfach oder Leistungskurs mit Ausflügen in die Geschichte der Rechenmaschinen, den Datenschutz und *Social Networking*: Das wäre doch einen Versuch wert, oder?

Ihr Michael Wiedeking
Herausgeber

Programmieren lernen

Scratch	scratch.mit.edu
Codecademie	www.codecademy.com
Khan Academy	www.khanacademy.org
CodeHS	codehs.com
Mozilla Thimble	thimble.webmaker.org

Apps

Cargo-Bot	twolivesleft.com/CargoBot
Robo Logic	digitalsirup.com/apps/app_robologic.html
Move The Turtle	movetheturtle.com

Für Fortgeschrittene

Robocode	robocode.sourceforge.net

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 Natalia Wilhelm via E-Mail unter natalia.wilhelm@bookware.de oder telefonisch unter 09131/8903-16.

Copyright

Impressum

KaffeeKlatsch Jahrgang 6, Nummer 1, Januar 2013
ISSN 1865-682X
Bookware – eine Initiative der
MATHEMA Verwaltungs- und Service-Gesellschaft mbH

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

Artikel

Kolumnen

Moderne Gesichter

von Rüdiger Keller

Mit *Java EE 7* wird auch die neue Version *2.2* von *JavaServerFaces* erscheinen. Dieser Artikel stellt zwei der größeren Neuerungen vor: Unterstützung für *HTML 5* und *Faces Flows*.

Lichtjahre voraus

von Róbert Bräutigam

Die funktionale Welt benutzt *Parser*-Kombinatoren schon seit einer Weile. Zum Glück hat die *Java*-Welt funktionale Entwicklung wieder entdeckt, und damit stehen mehr und mehr dieselben funktionalen Werkzeuge auch Java-Entwicklern zur Verfügung. *JParsec* ist ein solches Werkzeug, obwohl es ohne *Closures* ein bisschen umständlich ist.

Altlasten entsorgen

von Michael Wiedeking

Man sagt ja, dass wenn der Keller vollsteht, man einfach alles wegwerfen soll, was man länger als ein Jahr lang nicht mehr gebraucht hat, weil man es eh nie wieder brauchen wird. Das ist in der Informatik nicht anders: hat man nur einen begrenzten Platz zur Verfügung, so ist man gut bedient das Teil zu entsorgen, das man am längsten nicht mehr benutzt hat, weil auch hier wahrscheinlich die selbe Regel gilt, wie für das Gerümpel im Keller. Aber wie kann man so etwas elegant implementieren?

Batteries not included

von Lisa-Marie Wehlmann-Wiedeking

Wenn man sich etwas Teureres zu Weihnachten wünscht, muss man entweder den Wunschzettel kürzer ausfallen lassen oder Andere überzeugen, dass sie das Gewünschte auch benötigen. Wenn man ein Einzelkind ist, hat man zwar keine Geschwister zur Verfügung, aber in diesem speziellen Fall konnte der Vater aushelfen. Ein Nintendo 3DS!

Moderne Gesichter

Neues in JSF 2.2

von Rüdiger Keller

Mit *Java EE 7* wird auch die neue Version *2.2* von *JavaServerFaces* erscheinen.

Dieser Artikel stellt zwei der größeren Neuerungen vor: Unterstützung für *HTML 5* und *Faces Flows*.

Unterstützung für HTML 5

Mit *HTML 5* ist zum ersten Mal seit Langem wieder richtig Bewegung in die HTML-Entwicklung gekommen. So bietet es etliche neue Typen von *Input*-Feldern und andere neue Elemente. Zwar kann man mit *JSF* schon seit Version *1.2* direkt *Markup* in seinen *XHTML*-Seiten verwenden, aber wenn man die volle Funktionalität einer JSF-Komponente benötigt, ist man auf das beschränkt, was von JSF vorgegeben ist. So ist es momentan mit Bordmitteln nicht möglich, die neuen Input-Typen, wie *Date* oder *Number* zu nutzen. Um flexibel mit dem sich schnell entwickelnden HTML-Standard umzugehen, werden deshalb die sogenannten *Pass-Through*-Attribute und -Elemente eingeführt.

Pass-Through-Attribute

Pass-Through-Attribute sind Attribute, die nicht von der Komponente verarbeitet werden, sondern eins zu eins in die Ausgabe übernommen werden. Damit können beispielsweise die in HTML 5 neu eingeführten Attribute, wie *Placeholder* oder *Data*-Attribute, verwendet werden, was bisher so nicht möglich war.

Um Pass-Trough-Attribute anzugeben, gibt es zwei Möglichkeiten. Zum Einen können sie über einen eigenen *XML*-Namensraum *xmlns:p="http://java.sun.com/jsf/passthrough"* angegeben werden. Im Beispiel mit einer *Output*-Text-Komponente sieht das wie folgt aus.

```
<h:outputText value="Beispiel" p:foo="bar" />
```

Die Ausgabe dazu ist das Folgende.

```
<span foo="bar">Beispiel</span>
```

Zu beachten ist, dass der Namensraum nicht mit in die Ausgabe übernommen wird, weil dies den Sinn dieser Attribute zunichte machen würde.

Die Zweite Möglichkeit Pass-Through-Attribute anzugeben, ist sie als Kindelemente vom Typ *f:passThroughAttribute* zu Komponenten hinzuzufügen. Das letzte Beispiel sähe damit folgendermaßen aus, wobei die Ausgabe natürlich unverändert bleibt.

```
<h:outputText value="Beispiel">
  <f:passThroughAttribute name="foo" value="bar" />
</ h:outputText>
```

Pass-Through-Elemente

Noch weiter geht JSF mit den Pass-Through-Elementen. Diese dienen dazu nicht nur beliebige Attribute sondern sogar beliebige Elemente in den *Facelet*-Seiten nutzbar zu machen, welche trotzdem als Komponenten im Komponentenbaum abgebildet werden. Ein nicht-JSF-Element wird genau dann als Pass-Through-Element behandelt, wenn es mindestens ein Attribut aus dem speziellen Namensraum *xmlns:jsf="http://java.sun.com/jsf"* besitzt. Standardmäßig wird zu so einem Element eine Komponente vom Typ *"javax.faces.Panel"* mit einem speziellen *Renderer* vom Typ *"javax.faces.passthrough.Element"* erzeugt. Diese Komponente rendert sich dann genau so, wie in der Facelet-Seite angegeben, und nur die Attribute im *jsf-Namespace* werden interpretiert, die übrigen werden schlicht durchgereicht. Am Beispiel eines *HTML 5 Progress*-Elementes könnte dies so aussehen:

```
<progress jsf:id="progress" max="3"
          value="#{bean.progress}" />
```

Eine Ausnahme bei der Verarbeitung der Attribute gibt es bei der ID der Elemente. Sie wird immer gerendert, wenn ein *id*-Attribut mit oder ohne jsf-Namespace vorhanden ist und als Wert wird immer die JSF-generierte *Client-ID* verwendet.

Es gibt beim Erzeugen von Komponenten zu Elementen aber noch eine neue Besonderheit zu beachten. So können *Tags* vor dem Erzeugen des Komponentenbaums noch von einem *TagDecorator* umgeschrieben werden. TagDecorators gibt es seit JSF 2.0 und sie waren bisher ausschließlich benutzerdefiniert, aber mit JSF 2.2 schreibt der Standard nun einen immer aktiven TagDecorator vor. Dieser schreibt einige festgelegte Kombinationen von Elementen und Attributen um.

Beispielsweise werden *Anchor*-Elemente mit einem *jsf:action*-Attribut zu einem *h:commandLink* umgewandelt oder Input-Elemente, je nach Typ-Attribut, zu *h:inputText*, *h:commandButton* und so weiter. Bei dieser Umwandlung werden die Attribute ohne Namensraum zu Pass-Trough-Attributen und die Attribute aus dem jsf-Namensraum zu normalen JSF-Attributen.

Mit JSF 2.2 sind damit nicht nur alle heute verfügbaren HTML-Elemente nutzbar sondern auch alle zukünftigen. Außerdem können durch den TagDecorator-Mechanismus Facelet-Seiten geschrieben werden, welche direkt im Browser dargestellt werden können, weil sie für die Darstellung nicht von Komponenten-Tags abhängen, die erst von JSF interpretiert werden müssen. Dies kann bei der Zusammenarbeit mit Web-Designern wertvoll sein, die keine JSF-Kenntnisse haben. Das folgende Beispiel zeigt, wie eine Facelet-Seite, die davon Gebrauch macht, im Quellcode aussehen könnte.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:jsf="http://java.sun.com/jsf"
 xmlns:f="http://java.sun.com/jsf/core">
 <head jsf:id="head">
  <title>Beispiel</title>
 </head>
 <body jsf:id="body">
  <form jsf:id="form">
   <label jsf:for="email">E-Mail-Adresse</label>
   <input type="email" jsf:id="email"
    jsf:value="#{bean.email}" />
   <input type="button" value="Drück mich"
    jsf:id="button" jsf:action="#{bean.aktion}" />
  </form>
 </body>
</html>
```

Faces Flows

Ebenfalls neu sind die *Faces Flows*. Bisher sind JSF-Anwendungen monolithisch und bestehen aus einer Vielzahl von Seiten, zwischen denen beliebig navigiert werden kann, ohne dass es eine weitergehende Strukturierung gibt. Hier setzen die Flows an, welche auf den Umsetzungen in *ADF Task Flows*, *Spring Web Flows* und *Apache MyFaces CODI* basieren. Typischerweise würde man Flows nutzen, um Teilprozesse, Formularstrecken, wie eine Überweisung im Online-Banking, oder den Bezahlvorgang eines Online-Shops umzusetzen.

Ein Flow ist eine gekapselte Einheit, die, ähnlich einer Methode in Java, einen einzigen Einstiegspunkt sowie genau definierte Eingabeparameter und Rückgabewerte hat. Ein Flow wird dabei als ein Graph von Knoten modelliert, zwischen denen durch Navigation gewechselt werden kann. Wird ein Flow aufgerufen, wird der Einstiegsknoten dieses Flow-Graphen aktiv.

Flow-Knoten

Naheliegend ist, dass die Knoten eines Flows *JSF-Views* sein können, die vorgeben, welche *Page* beziehungsweise welches Facelet angezeigt wird, wenn sie aktiv sind. Aber dies ist nicht die einzige Art von Knoten, daneben sind auch die Folgenden weiteren vorgesehen:

- Methodenaufruf
- *Switch*
- *Flow*-Aufruf
- *Flow-Return*

Methodenaufrufsknoten dienen dazu, die Logik vom *User-Interface* zu trennen. Typischerweise wird über einen *EL*-Ausdruck die aufzurufende Methode bestimmt und deren Rückgabewert zur Weiternavigation im Flow genutzt. Außerdem ist es möglich eine Standardnavigation anzugeben, zum Beispiel falls die Methode keinen Navigations-*String* zurückgibt.

Ein *Switch*-Knoten dient zur bedingten Verzweigung, ähnlich wie die *switch*-Anweisung aus Java. Dabei können mehrere Paare von *EL*-Ausdrücken und Navigationsregeln angegeben werden. Diese werden der Reihe nach ausgewertet, bis einer der Ausdrücke *true* ergibt. Dann wird die zu diesem Ausdruck gehörende Navigationsregel benutzt um weiter zu navigieren. Zusätzlich kann wieder eine Standardnavigation angegeben werden, die benutzt wird, wenn keiner der Ausdrücke *true* zurückgibt.

Ein Flow-Aufrufsknoten dient dazu einen *Sub-Flow* aufzurufen, ihm seine Eingabeparameter zu übergeben und seine Rückgabeparameter nach seiner Beendigung entgegenzunehmen.

Der *Flow-Return*-Knoten wird benutzt um den aktuellen Flow zu beenden und in den aufrufenden Flow zurückzuspringen. Er gibt auch an, welche Werte als Rückgabewert übergeben werden.

Scopes und Sonstiges

Um die Flows zu realisieren, wird auch ein neuer *Scope* eingeführt, der sogenannte *FacesFlowScope*. Er ist als *CDI-Scope* implementiert, was auch zeigt, dass JSF in Zukunft weg vom eigenen *Bean*-Management und hin zum JEE-Standard *CDI* geht. An dieser Stelle sei auch noch erwähnt, dass der *View-Scope* bei JSF 2.2 nun auch bei Einsatz von CDI zur Verfügung steht. Alle in einem

Flow benutzten *Beans* und Werte werden also typischerweise in dessen *Flow-Scope* gespeichert. Im Gegensatz zum *Conversation-Scope* beginnt und endet ein Flow-Scope automatisch, ohne zusätzliche Maßnahmen des Entwicklers.

Wie genau Flows definiert beziehungsweise notiert werden, ist noch offen. Diskutiert werden eine Angabe per XML-Konfigurationsdateien oder die Definition über eine Java-API, wobei allerdings wieder zwischen dem Gebrauch des *Builder-Patterns* und einer *Fluent-API* abgewogen wird.

Schlussendlich ist auch vorgesehen, dass man Flows auch separat von einer Web-Anwendung in *JARs* verpacken kann, die dann vollständig gekapselt als wiederverwendbare Bestandteile in beliebigen Web-Anwendungen genutzt werden können. Dieses Feature bietet vielleicht auch neue Möglichkeiten, um es großen Entwicklerteams zu erlauben, effizienter gemeinsam eine Web-Anwendung zu entwickeln.

Schluss

Als Referenzen für diesen Artikel diente die Web-Seite von *J-Development*, „What's new in JSF 2.2" [1], die beiden JavaOne Vorträge von Ed Burns zum Thema [2], [3] und die JSF-Dokumentation aus dem aktuellen *Nightly-Build*, welcher vom *Java.net Maven Repository* bezogen werden kann [4]. Zu beachten ist, dass die Version 2.2 des JSF-Standards wohl erst Mitte 2013 fertig gestellt wird und sich in der Zwischenzeit noch Änderungen zu den hier vorgestellten Features ergeben können.

Referenzen

[1] Tijms Arjan *What's new in JSF 2.2?*, http://jdevelopment.nl/jsf-22

[2] Burns Ed *What's New in JSF: A Complete Tour of JSF 2.2*, JavaOne 2012, 03.10.2012, https://oracleus.activeevents.com/connect/sessionDetail.ww?SESSION_ID=3870

[3] Burns Ed *Standardizing Web Flow Technology with JSF Faces Flows*, JavaOne 2012, 01.10.2012, https://oracleus.activeevents.com/connect/sessionDetail.ww?SESSION_ID=4627

[4] Java.net *JSF 2.2 Snapshot Builds*, https://maven.java.net/index.html#nexus-search;gav~javax.faces~javax.faces-api~2.2-SNAPSHOT~~

Kurzbiographie

Rüdiger Keller ist Diplom-Informatiker und seit Anfang 2007 als Software-Entwickler, Trainer und Consultant bei der MATHEMA Software GmbH angestellt. Er beschäftigt sich gerne mit neuen Entwicklungen im Java- und JEE-Umfeld. Zu seinen aktuellen Lieblingsthemen gehören Scala und GWT. Sein Blog zu Software-Entwicklungsthemen ist unter http://ruedigerkeller.blogspot.com zu finden.

Lichtjahre voraus

Einfach Domänenspezifische Sprachen (DSL's) entwerfen mit JParsec

von Róbert Bräutigam

Die funktionale Welt benutzt *Parser*-Kombinatoren schon seit einer Weile. Zum Glück hat die *Java*-Welt funktionale Entwicklung wieder entdeckt, und damit stehen mehr und mehr dieselben funktionalen Werkzeuge auch Java-Entwicklern zur Verfügung. *JParsec* ist ein solches Werkzeug, obwohl es ohne *Closures* ein bisschen umständlich ist.

Problemstellung

Häufig ergeben sich Probleme in einem Projekt, die mit einer Domänenspezifischen Sprache (*DSL*)[1] besser lösbar sind. Meistens benutzen wir Sprachen die schon da sind, zum Beispiel *Property*-Dateien für Konfigurationen, eine *XML*-basierte Sprache für Inhalte und *SQL* für Datenbankzugriffe.

Manchmal würde aber eine eigene Sprache, die speziell für das Problem im Projekt konstruiert wurde, besser passen. Leider war es bisher schwer eine eigene Sprache zu entwickeln, da die meisten Werkzeuge selbst zwei verschiedene DSL's benötigten, eine für die lexikalische und eine für die syntaktische Analyse. Man musste davon dann Code generieren lassen, dazu kam noch Fehlerbehandlung und die eigentlichen Code-Fragmente zur Bearbeitung.

JParsec ermöglicht es DSL's einfach nur mit Java-Code zu schreiben inklusive einer brauchbaren, eingebauten Fehlerbehandlung.

Compiler

Um eine eigene Sprache zu entwerfen, muss man natürlich einen *Compiler* für die Sprache schreiben. Grundsätzlich gibt es mindestens 3 Stufen um eine textbasierte Sprache zu verarbeiten:

- *Lexikalische Analyse*: Diese verarbeitet den Text und erzeugt eine Liste von *Token*. Die Token sind schon mit einer Syntax-nahen Bedeutung versehen sowie „Operator", „Schlüsselwort", „Nummer", „String" usw. Meistens werden Kommentare und Leerzeichen in dieser Stufe weggeworfen.
- *Syntaktische Analyse* (manchmal einfach *Parsing* genannt): Nimmt die Liste von Token und erzeugt einen „Syntaxbaum". Ein Syntaxbaum ist eine leicht verarbeitbare Repräsentation des Quellcodes.
- Als Letztes verarbeitet der Compiler den Syntaxbaum um die eigentliche Ausgabe zu erzeugen, z. B. *Object*-Code.

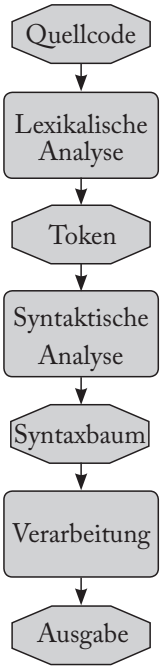


Abbildung 1

JParsec hilft die ersten zwei Stufen mithilfe von Parser-Kombinatoren zu implementieren. Parser-Kombinatoren sind übergeordnete Funktionen, die Parser-Funktionen als Parameter akzeptieren, kombinieren (zu Sequenzen, Schleifen usw.) und eine neue Parser-Funktion erzeugen.

Das Beispiel

Es ist relativ schwer die Syntax von Web-Ressourcen oder Web-Services nur mit *XML Schema Definition* (*XSD*)[2] zu definieren. XSD ist wortreich, umständlich zu schreiben und es fehlen mehrere Eigenschaften, die es sehr viel leichter machen würden, diese zu versionieren oder rückwärtskompatible Änderungen einzuführen.

[1] Englisch: Domain Specific Language

[2] Die XML Schema Definition ist eine XML-basierte Sprache, mit der man andere XML-Sprachen definieren kann.

Deshalb definieren wir hier exemplarisch eine (nicht vollständige) Sprache, die es möglich macht Typen einfach zu definieren und zu versionieren, und gleichzeitig ermöglicht sie in XSD zu übersetzen (Compilieren).

Als Erstes soll es möglich sein *primitive* Typen zu definieren:

```
type AccountNumber[1.0] = String
```

Die Definition fängt mit dem Schlüsselwort *type* an, dann kommt ein *Identifier*, also ein Name mit den üblich erlaubten Zeichen, weiterhin eine Versionsdefinition mit Klammern und der eigentliche Typ (*String*). Dazu erlauben wir es noch, die selben Eingrenzungen zu verwenden, wie in XSD für die *primitiven* Typen (hier nur zwei Beispiele für Länge):

```
type AccountNumber[1.0] =
  String minlength 20 maxlength 34
```

Um neue Versionen einfach definieren zu können, erlauben wir mehrere Definitionen zum selben Namen:

```
type AccountNumber  [1.0] = String
                    [1.1] = String maxlength 34
```

Hier erlaubt die Version 1.0 alle Strings, derweil die Version 1.1 nur Strings bis 34 Zeichen Länge erlaubt. Diese Definition soll völlig gleichwertig sein mit

```
type AccountNumber  [1.0] = String
type AccountNumber  [1.1] = String maxlength 34
```

also wenn beide Versionen voll ausgeschrieben sind.

Wir möchten auch die Dokumentation in Form eines Kommentars erlauben:

```
type AccountNumber[1.0] = String
  "Internationale Kontonummer (IBAN)"
```

Dokumentation ist einfach ein String mit Anführungszeichen nach der Definition.

Mit diesen *primitiven* Typen könnten dann komplexe Typen definiert werden (was in diesen Artikel nicht enthalten ist):

```
type CashTransfer[1.0] = {
  sourceAccount: AccountNumber[1.0]
  targetAccount: AccountNumber[1.0]
  amount: BigDecimal
}
```

Wie genau die Versionen von komplexen Typen zusammenpassen, Ko- und Kontra-Varianz bei Anfragen und Antworten (und wie diese definiert werden) wird in diesem Artikel nicht diskutiert.

Lexikalische Analyse

In JParsec wird die lexikalische und die syntaktische Analyse mit der gleichen Methode gemacht. Beide werden mit Parser-Objekten implementiert, wobei der lexikalische Parser normalerweise Token und der syntaktische Parser Syntaxbaum-Objekte zurückgibt.

Der erste lexikalische Parser soll einfach Kommentare und Leerzeichen wegwerfen.

```
public class NachrichtenSprache {
  public static final Parser<Void>
  DELIMITERS = Parsers.or(
    Scanners.JAVA_LINE_COMMENT,
    Scanners.JAVA_BLOCK_COMMENT,
    Scanners.WHITESPACES
  ).skipMany();
}
```

Die drei Scanner-Konstanten (*line comment*, *block comment* und *whitespaces*) sind vordefinierte Parser-Objekte, die gewöhnliche Java-Kommentare (mit "//" und mit "/*...*/") und alle Leerzeichen parsen und nichts (*Void*) zurückgeben. *Parsers.or* ist ein Parser-Kombinator, er nimmt die drei Parser und gibt einen Parser zurück, welcher alles verarbeiten kann, was die drei vorher genannten Parser können. *skipMany* ist auch ein Kombinator, der versucht einen Parser mehrmals nacheinander auf den *Input* anzuwenden. Da dieser beliebig viele Treffer erlaubt (auch 0), wird der Parser nie fehlschlagen.

Auf den ersten Blick kann es einem merkwürdig vorkommen, dass Parser-Objekte statisch definiert, aber eigentlich reine Funktionen sind und keinen Zustand haben. Deshalb ist es aber möglich sie einfach zu kombinieren.

Der *DELIMITERS*-Parser ist also ein Parser-Objekt, das alle Java-Kommentare und Leerzeichen wegwirft. Jetzt müssen wir nur die nützlichen Zeichen und Wörter definieren, und davon gibt es vier Sorten: Operatoren (wie Punkte, Klammern usw.), Schlüsselwörter (wie etwa *type*), Nummern und in Anführungszeichen stehende *Strings*. Dazu benutzen wir auch den *Parser.or*-Kombinator:

```
// Erkennt nur Operatoren und Schlüsselwörter
public static final Parser<?>
OPERATORS_AND_KEYWORDS =
  Terminals.caseInsensitive(
    Operators.valuesArray(),
    Keywords.valuesArray()
  );

// Erkennt Operatoren, Schlüsselwörter, Nummern und Strings
public static final Parser<?>
TOKENIZER = Parsers.or(
  Terminals.IntegerLiteral.TOKENIZER,
  Terminals.StringLiteral.DOUBLE_QUOTE_
    TOKENIZER,
  OPERATORS_AND_KEYWORDS
);
```

Das erste Parser-Objekt verarbeitet Operatoren und Schlüsselwörter. Nehmen wir an, dass beide in Aufzählungen (*enum*) mit einer *valuesArray()*-Methode aufgelistet sind, welche die Werte (oder Namen) als ein *String-Array* zurückgibt:

```
public enum Operator
  extends ValuesAsArray
{
  Dot("."),
  Equals("="),
  BracketOpen("["),
  BracketClose("]");

  private String literal;

  private Operator(String literal) {
    this.literal = literal;
  }

  public String toString() {
    return literal;
  }
}
public enum Keywords
  extends ValuesAsArrays
{
  Type,
  MinLength,
  MaxLength;
}
```

Das *TOKENIZER*-Objekt kombiniert die Operatoren und Schlüsselwörter noch mit vordefinierten Nummern- und String-Parsern. Zusammen mit dem *DELIMITERS*-Objekt haben wir die lexikalische Analyse des Codes schon fertig, obwohl wir diese erst nach der Implementierung des Parsers für die syntaktische Analyse nutzen können.

Die Rückgabewerte sind hier nicht explizit definiert, weil es unterschiedliche Typen für die verschiedenen Parser sind. *Terminal* Parser-Objekte geben Strings zurück, bei Operatoren und Schlüsselwörtern werden *Token*-Objekte zurückgegeben. Für beide gibt es standardisierte syntaktische Parser, die zu beiden Varianten passen und später leicht kombinierbar sind.

Syntaktische Analyse (Teil 1)

Die einfachste Methode die syntaktische Analyse zu definieren, wenn man den Syntaxbaum nicht vorgegeben bekommt, ist bei der kleinsten unabhängigen Einheit anzufangen. Für jede Einheit, die selbst Daten enthält, kann man eine neue Klasse schreiben und diese dann in den Syntaxbaum einfügen.

Die kleinste unabhängige Einheit ist hier die Versionsdefinition:

```
public class Version {
  private int major;
  private int minor;

  // Erkennt: <major> '.' <minor>, und erzeugt Version Objekt
  public static final Parser<Version>
  CORE_PARSER = Parsers.sequence(
    Terminals.IntegerLiteral.PARSER.followedBy(
      Parsers.token(Operator.Dot)
    ),
    Terminals.IntegerLiteral.PARSER,
    new Map2<String, String, Version>() {
      public Version map(
        String major, String minor
      ) {
        return new Version(
          Integer.parseInt(major),
          Integer,parseInt(minor)
        );
      }
    }
  );
  // Erkennt: '[' <CORE_PARSER> ']', und liefert Version
  public static final Parser<Version>
  PARSER = Parsers.between(
    Parsers.token(Operator.BracketOpen),
    CORE_PARSER,
    Parsers.token(Operator.BracketClose)
  );

  ... Konstruktor, Setter, Getter ...
}
```

Das Versions-Objekt hat eine *major*- und eine *minor*-Versionsnummer. Der *CORE_PARSER* verarbeitet nur die zwei Nummern mit einem Punktzeichen dazwischen. Um das zu tun, benutzt er den *Parsers.sequence*-Kombinator, wobei das erste Parser-Objekt in der Reihe selbst eine Kombination mit *followedBy* ist. Eine Sequenz-Kombination nutzt alle Parser der Reihenfolge nach und ist erfolgreich wenn alle Parser erfolgreich waren. Der *followedBy*-Kombinator ist wie eine Sequenz von zwei Parsern, bei der aber der Rückgabewert des zweiten Parsers verworfen wird.

Parser haben einen Rückgabewert (Typ angegeben durch Klassenparameter). *Terminal*-Parser geben Strings zurück (den Wert des Textes) und alle Kombinatoren definieren, wie der Rückgabewert zusammengestellt wird. Wenn wir eigene Rückgabewerte erstellen wollen, können wir *Map*-Objekte benutzen, bei denen die einzelnen Rückgabewerte von den kombinierten Parsern typsicher zur Verfügung stehen. *Map*-Objekte simulieren eigentlich *Closures*, was ein Workaround dafür ist, dass es so etwa in Java (noch) nicht gibt. *CORE_PARSER* nutzt *Map2* um die zwei *Integer*-Werte in das Versions-Objekt zu packen.

Das *PARSER*-Objekt kombiniert dann *CORE_PARSER* so, dass es zwischen Klammern stehen muss und den Rückgabewert von *CORE_PARSER* behält (was eine Eigenschaft des *between*-Kombinators ist).

Die Methode *Parsers.token* kreiert einen Parser von einer speziellen *TokenMap*-Funktion, die einfach entscheidet, ob das nächste Token akzeptiert werden soll oder nicht. Dazu muss die *Operator-Enumeration* eine *TokenMap* implementieren:

```
public enum Operator
  extends ValuesAsArray
  implements TokenMap<Operator>
{
   ... Vorherige Methoden ...

  public Operator map(Token token) {
    if (
      (token == null)
      ||
      (!literal.equals(token.value().toString())
    ) {
      return null;
    }
    return this;
  }
}
```

Wenn die *map*-Methode *null* zurückgibt, wird der *Parsers.token*-Parser fehlschlagen, andernfalls den entsprechenden Operator zurückgeben. Also wird *Parsers.token(Operator.Dot)* nur dann erfolgreich parsen, wenn das nächste Token ein Punktzeichen repräsentiert.

Test

An dieser Stelle können wir schon Versionsnummern parsen:

```
Version version = Version.PARSER
  .from(TOKENIZER, DELIMITERS)
  .parse("[1.0]");
```

Dies wird ein Versionsobjekt zurückgeben mit *major*-Nummer 1 und *minor*-Nummer 0. Weil die lexikalische Analyse auch gleich gemacht wird, kann der Parser auch das Folgende korrekt parsen:

```
Version version = Version.PARSER
  .from(TOKENIZER, DELIMITERS)
  .parse("[ 1 . 0 ]");
```

Es darf auch Kommentare beinhalten:

```
Version version = Version.PARSER
  .from(TOKENIZER, DELIMITERS)
  .parse("/* Kommentar */ [1.0]");
```

Obwohl das noch nicht viel ist, weil der *Version.PARSER* keinen Zustand hat, können wir ihn leicht wiederverwenden und später mit anderen Parsern kombinieren.

Syntaktische Analyse (Teil 2)

Parsen wir als nächstes die *minlength*- und *maxlength*-Randbedingungen:

```
public interface TypeConstraint {
  public static final Parser<TypeConstraint>
  PARSER = Parsers.or(
    MinLength.PARSER,
    MaxLength.PARSER
  );
}

public class MinLength
  implements TypeConstraint
{
  public int length;

  public static final Parser<MinLength>
  PARSER = Parsers.token(Keyword.MinLength).next(
    Terminals.IntegerLiteral.PARSER.map(
      new Map<String, MinLength>() {
        public MinLength map(String length) {
          return new MinLength(
              IntegerparseInt(length)
          );
        }
      }
    )
  );

  ... Konstruktor, Setter, Getter ...
}

public class MaxLength implements TypeConstraint {
  ... Genau so wie MinLength ...
}
```

Der *TypeConstraint.PARSER* kann beliebig mit neuen Bedingungen erweitert werden. Der *Parsers.or* kann beliebig viele Parser kombinieren, vorausgesetzt alle Parser geben einen Subtyp von *TypeConstraint* zurück.

Der Kombinator *next* in *MinLength.PARSER* funktioniert genauso wie *followedBy*, nur behält er den Rückgabewert des zweiten Parsers anstatt des Ersten.

Ein *primitive* Typ ist dann so definiert:

```
public class PrimitiveType {
  private String type;
  private List<TypeConstraint> constraints;

  public static final Parser<PrimitiveType>
    PARSER = Parsers.sequence(
      Terminals.Identifier.PARSER,
      TypeConstraint.PARSER.many(),
      new Map3<String, List<TypeConstraint>,
        PrimitiveType>() {
        public PrimitiveType map(
          String type, List<TypeConstraint> constraints
        ) {
          return new PrimitiveType(type, constraints);
        }
      }
    );

    ... Konstruktor, Setter, Getter ...
}
```

Der Kombinator *many* versucht einen Parser so oft wie möglich nacheinander zu benutzen, und gibt eine potenziell leere Liste von Rückgabewerten zurück.

Die gesamte Deklaration sieht dann so aus:

```
public PrimitiveTypeDeclaration {
  private String name;
  private Version version;
  private PrimitiveType type;
  private String documentation;

  // Erkennt: <version> '=' <primitive type with constraints...>
  public static final Parser<Pair<Version, PrimitiveType>>
  CORE_PARSER =
    Parsers.tuple(
      Version.PARSER.followedBy(
        Parsers.token(Operator.Equals)
      ),
      PrimitiveType.PARSER
    );

  // Erkennt: 'type' <name> <CORE_PARSER>*
  public static final Parser<List<PrimitiveTypeDeclaration>
  PARSER =
    Parsers.sequence(
      Parser.token(Keyword.Type).next(
        Terminals.Identifier.PARSER
      ),
      CORE_PARSER.many1(),
      Terminals.StringLiteral.PARSER,

      new Map2<String, List<Pair<Version,
        PrimitiveType>>, String>() {
        public List<PrimitiveTypeDeclaration> map(
          String name,
          List<Pair<Version, PrimitiveType>>
            declarations,
          String documentation
        ) {
          List<PrimitiveTypeDeclaration> result =
            new LinkedList<>();
          for(
            Pair<Version, PrimitiveType> declaration: declarations
          ){
            result.add(new PrimitiveTypeDeclaration(
                name,
                declaration.a,
                declaration.b,
                documentation
            ));
          }
          return result;
        }
      });
```

Da alle Deklarationen eigentlich auch mehrere Deklarationen sein können, verarbeitet der *CORE_PARSER* nur die Version und die Typangabe. Der PARSER benutzt dann *CORE_PARSER* mit dem *many1*-Kombinator, welcher den *CORE_PARSER* mindestens einmal anwendet und dann wie *many* weitermacht. Der Rückgabewert von *CORE_PARSER* ist ein Versionstyp-Paar, da es kein wirkliches Objekt für diese Kombination gibt. Das verarbeitet der PARSER dann um eine Liste von Deklarationen zu erstellen.

Eine Datei kann aber mehrere Deklarationen enthalten. Dazu benötigen wir noch eine Klasse, die meistens *Unit*[3] heißt (obwohl es in diesem Fall nur ein Interface und kein wirkliches Objekt im Syntaxbaum ist):

```
public interface Unit {
  public static final Parser<List<PrimitiveTypeDeclaration>>
  PARSER =
  PrimitiveTypeDeclaration.PARSER.many().map(
    new Map<List<List<PrimitiveTypeDeclaration>>() {
      public List<PrimitiveTypeDeclaration> map(
        List<List<PrimitiveTypeDeclaration>>
          doubleDeclarations
      ) {
        List<PrimitiveTypeDeclaration> result =
          new LinkedList<>();
        for (List<PrimitiveTypeDeclaration>
          declarations : doubleDeclarations) {
          result.addAll(declarations);
        }
        return result:
      }
    }
  );
}
```

[3] Von *Compilation Unit*, also kompilierende Einheit.

Weil alle Deklarationen mehrere sein können, muss der *Unit.PARSER* eine Liste von Listen verarbeiten und sie dann einfach abflachen.

Endparser

In der Klasse *NachrichtenSprache* kann man dann einen Parser kombinieren, der lexikalische und syntaktische Analyse macht, und den definierten Syntaxbaum zurückgibt:

```
public class NachrichtenSprache {
  ... Vorherige Definitionen ...

  public static final
  Parser<List<PrimitiveTypeDeclaration>> PARSER =

    Unit.PARSER.from(TOKENIZER, DELIMITERS);
}
```

Mit der Methode *NachrichtenSprache.PARSER.parse()* kann man leicht Strings oder auch *Streams* parsen lassen. Zum Beispiel wird der folgende Satz in dieser Sprache

```
// Kontonummer
type AccountNumber  [1.0] = String
                    [1.1] = String minlength 20

// Überweisungsdaten
type Purpose[1.0] = String maxlength 100
```

in den Syntaxbaum in Abbildung 2 übersetzt.

Als letzten Schritt erzeugt der Compiler die eigentliche Ausgabe von diesem Baum, z. B. könnte er die entsprechende XSD und/oder auch Java-Code generieren.

Fazit

Mit JParsec verschwinden die großen Aufwände, die für eine eigene DSL in der Vergangenheit nötig waren, und man kann sich trauen etwas selbst zu bauen. Leider werden *Map*-Funktionen ohne *Closures* schnell unlesbar. Aber dabei hilft es, den Parser in kleine Teile zu zerschneiden oder gar einfach den Compiler in Scala zu entwickeln – wenn es möglich ist.

Weiterführende Literatur

- The Codehouse *JParsec*, http://jparsec.codehaus.org

Kurzbiographie

Robert Bräutigam ist als Senior-Consultant für die MATHEMA Software GmbH tätig. Er ist seit 1999 als Entwickler und Architekt im Java Enterprise-Umfeld beschäftigt und interessiert sich für leichtgewichtige Lösungen in Technologien sowie im Projektmanagement.

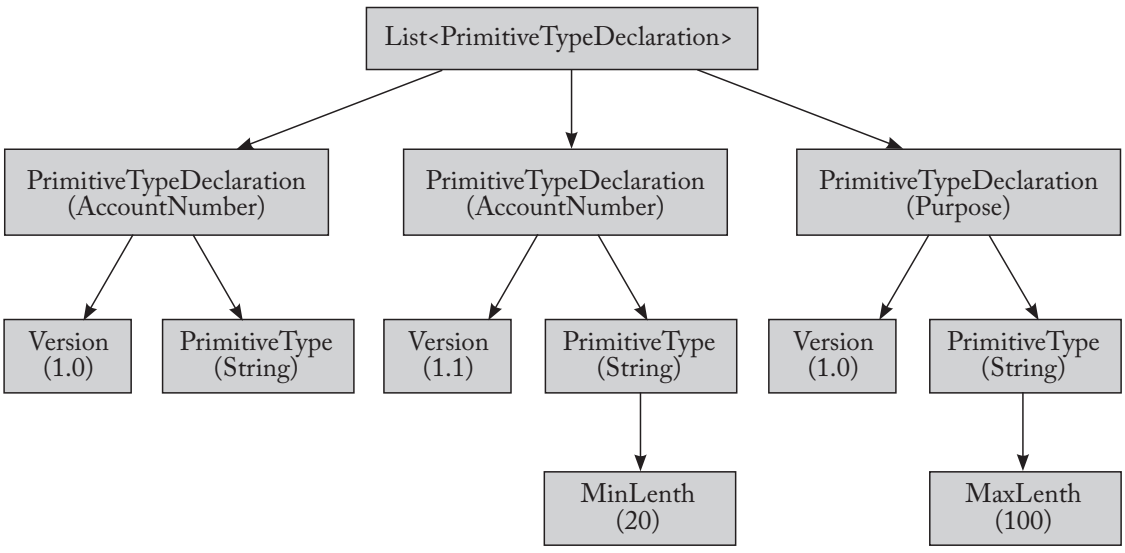


Abbildung 2

Des Programmierers kleine Vergnügen

Altlasten entsorgen

von Michael Wiedeking

Man sagt ja, dass wenn der Keller vollsteht, man einfach alles wegwerfen soll, was man länger als ein Jahr lang nicht mehr gebraucht hat, weil man es eh nie wieder brauchen wird. Das ist in der Informatik nicht anders: hat man nur einen begrenzten Platz zur Verfügung, so ist man gut bedient das Teil zu entsorgen, das man am längsten nicht mehr benutzt hat, weil auch hier wahrscheinlich die selbe Regel gilt, wie für das Gerümpel im Keller. Aber wie kann man so etwas elegant implementieren?

Der Einfachheit halber wollen wir uns vorstellen, dass wir Platz für $n = 2^k$ Elemente haben. Der Platz ist begrenzt, weil er sehr teuer ist. Nichtsdestoweniger wollen wir uns den Weg zum billigeren Speicherplatz sparen, weswegen wir die wichtigen Sachen in diesem teuren Platz lagern, um die Wege kurz zu halten.

Ein *Cache* hat genau so ein Problem. Nachdem man nicht alles im Speicher halten kann, richtet man einen Cache endlicher Größe ein und hofft, dass die Dinge, die in diesem gehalten werden, möglichst oft verwendet werden. Allerdings ist der Platz begrenzt und deshalb kommt es gelegentlich vor, dass erst Platz geschaffen werden muss, bevor ein neues Element zwischengespeichert werden kann.

Es gibt viele Möglichkeiten darüber zu entscheiden, welches Element zum Platzschaffen entsorgt werden soll. Aber wie schon oben angedeutet, bietet es sich oft an jenes Element zu entfernen, das am längsten nicht mehr benutzt worden ist, weil es vermutlich lange nicht mehr oder im Idealfall nie mehr benutzt werden wird.

Sind die kostbaren Plätze von 0 bis $n - 1$ nummeriert, so könnte man eine Liste anlegen, in der Paare verwaltet werden, die diese Platznummer und das darin enthaltene Element verwalten. Wird auf ein Element zugegriffen, so wird diese Liste durchsucht, bis das passende Element gefunden wird und dieses Paar wird dann einfach an den Anfang der Liste gestellt. So ist das Suchen zwar relativ aufwändig, aber das zuletzt Benutzte ist trivial zu finden: es steht dann am Ende der Liste.

Alternativ könnte man über die Platznummer nicht nur das Element sondern auch das „Alter“ des Eintrags verwalten. Jedes Mal, wenn auf das Element zugegriffen wird, wird das Alter hochgezählt. Benötigt man einen freien Platz, so verwendet man den Eintrag, der das kleinste Alter hat, wobei leider die ganze Altersliste durchsucht werden muss. Hierbei muss man allerdings darauf achten, dass das Alter überlaufen kann und es dadurch zu Fehlinterpretationen kommen kann. Oder aber man zieht das niedrigste Alter von allen Einträgen ab, wobei man dummerweise die Liste erneut durchlaufen muss.

Natürlich kann man beide Verfahren kombinieren, was sich dann aber negativ auf den Speicherverbrauch auswirkt. Wie immer muss man also abwägen, ob der Aufwand den Nutzen rechtfertigt. Oder aber man macht es, wenn denn n klein genug ist, einfach richtig.

Eine schöne Möglichkeit das Element mit der Eigenschaft *Least Recently Used* (*LRU*) zu finden, ist über *Bit*-Matrizen. Hat man etwa $n = 4$ Plätze zur Verfügung, bedient man sich einer 4×4-Matrix, die wie folgt initialisiert wird.

	0	1	2	3
0	0	1	1	1
1	0	0	1	1
2	0	0	0	1
3	0	0	0	0

Wann immer man nun auf das Element i zugreift, setzt man alle Elemente der Zeile i auf 1 und schließlich alle Elemente der Spalte i auf 0. Das nachfolgende Beispiel zeigt diese Veränderung für das Element $i = 1$:

	0	1	2	3		0	1	2	3
0	a	b	c	d		a	**0**	c	d
1	e	f	g	h		**1**	**0**	**1**	**1**
2	i	j	k	l		i	**0**	k	l
3	m	n	o	p		m	**0**	o	p

Das Schöne an diesem Verfahren ist – die korrekte Initialisierung vorausgesetzt –, dass immer eine Zeile nur aus Nullen besteht, nämlich die von dem Element, das tatsächlich zuletzt verwendet worden ist. Wird ein Element markiert, dann wird dessen Zeile zunächst mit Einsen markiert und dann die entsprechende Spalte gelöscht. Damit bekommt das Element sozusagen $n - 1$ „Leben". Zusätzlich werden aber auch alle anderen Elemente, auf die vor diesem Element zugegriffen wurde, um ein Leben erleichtert, nämlich um das in der Spalte des neu markierten Elements.

Da im Laufe der Zeit alle Elemente durchlaufen werden, muss zwangsläufig immer eins dasjenige sein, das zuletzt benutzt wurde. Und das ist genau das, bei dem alle Bits der korrespondierenden Zeile Null sind. Denn auf alle anderen ist ja nach dem zuletzt benutzten zugegriffen worden, womit alle Lebens-Bits der Reihe nach ausgelöscht worden sind.

	–		1		3		2		0		1	
	0	1	2	3	0	1	2	3	0	1	2	3
0	0	1	1	1	0	**0**	1	1	0	**0**	**1**	**1**
1	0	0	1	1	**1**	**0**	1	**1**	1	**0**	**0**	**0**
2	0	0	0	1	0	**0**	0	**1**	**1**	**1**	**0**	1
3	0	0	0	0	0	**0**	0	0	**1**	**1**	**1**	**0**

Ist nun $n = 8$, dann lässt sich diese Matrix m noch bequem in einem 64-Bit-Maschinenwort unterbringen.[1] Innerhalb von m repräsentieren dann die Bits 0 bis 7 die Zeile 0, die Bits 8 bis 15 die Zeile 1 und die Bits $8i$ bis $8i + 7$ die Zeile i. Die Bytes 0 bis 7 repräsentieren also die Zeilen 0 bis 7 und innerhalb eines Bytes entsprechen die Bits 0 bis 7 den entsprechenden Spalten.

Um nun die Zeile i komplett auf 1 zu setzen, kann man sich des Bit-weisen Oders | bedienen. Die Hexadezimalzahl 0xFF hat genau 8 gesetzte Bits, die nur noch an die richtige Zeile – sprich: Byte – geschoben werden muss.[2]

$$m = m \mid (\mathrm{0xFF} \ll (8 * i));$$

Jetzt müssen nur noch alle Bits in der korrespondierenden Spalte auf 0 gesetzt werden. Dazu bedient man sich des Bit-weisen Und &. Die hexadezimale Bit-Maske 0x01 hat genau ein gesetztes Bit und repräsentiert die Spalte 0 im niederwertigsten Byte 0. Also hat die Maske 0x0101010101010101 für jedes Byte in m in Spalte 0 ein gesetztes Bit. Verschiebt man nun diese Maske um i Positionen nach links, so stehen diese Einsen an der Spalte i. Um nun diese Bits zu löschen, braucht man genau das Gegenteil, so dass die Maske überall 1 ist, außer an der Spalte i der einzelnen Zeilen bzw. Bytes. Das erreicht man durch das Komplement ~, mit dem die Bits invertiert werden können.[3]

$$m = m \;\&\; \sim(\mathrm{0x0101010101010101} \ll i);$$

Um den *LRU*-Platz zu finden, muss man lediglich noch nach dem Byte suchen, das nur aus Nullen besteht. Und das haben wir bereits im September-Vergnügen 2009 [1] souverän gemeistert, ohne über die Bytes in einer Schleife iterieren zu müssen.

[1] Die Größe ist natürlich nicht auf 64-Bit begrenzt. Hier wurde nur exemplarisch ein 64-Bit-Wort gewählt, weil die Bit-Breite eine Quadratzahl ist, die noch auf den meisten Rechnern zur Verfügung steht. Dieses Verfahren lässt sich leicht auch auf *Bit-Arrays* quasi beliebiger Länge anwenden, da sich alle beteiligten Operationen leicht übertragen lassen und effizient zu implementieren sind.

[2] Achtung: Hierbei muss man peinlichst darauf achten, dass die Konstanten den korrekten Typ haben. In Java und C# etwa sind die Konstanten nur vom Typ *int*, obwohl – oder völlig unabhängig davon, dass – m für 64-Bit vom Typ *long* sein müsste. Obwohl das Ergebnis *long* ist, ist das Verschieben von 0xFF trotzdem eine *int*-Operation, kommt also nie über die 32-Bit-Grenze hinaus! In diesen Sprachen muss man also 0xFFL benutzen, um klar zu machen, dass man eine *long*-Maske braucht.

[3] In Anlehnung an die vorige Fußnote sei an dieser Stelle noch erwähnt, dass hier grundsätzlich die selben Überlegungen gelten. Allerdings würde hier wenigstens ein *Compiler* darauf hinweisen, dass eine derart große Konstante keinen Platz in dem implizierten Typ (*int*) hat. Also muss etwa in Java und C# auch hier das *long-Literal-L* an die Konstante gehängt werden.

Referenzen

[1] Wiedeking, M. *Des Programmierers kleine Vergnügen – Die allerletzte Null,* KaffeeKlatsch, Jahrgang 2, Nr. 9, S. 31–33, Bookware, September 2009 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2009-09.pdf

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.

Deutsch für Informatiker

Zitate

von Alexandra Specht

Wie wir in den letzten Monaten erfahren haben, kann man eine Menge falsch machen, wenn man etwas aus den Werken anderer verwendet. Falls Sie selbst zitieren möchten, hilft Ihnen dieser Artikel wenigstens dabei, die Satzzeichen richtig zu verwenden.

Ein Zitat (*das, lateinisch citatum „Angeführtes", „Aufgerufenes"*) ist eine wörtlich übernommene Stelle aus einem Text, kann aber auch ein Hinweis auf eine bestimmte Textstelle sein.

Zitate stehen generell in Anführungszeichen. Ein Zitat, welches in einem anderen Zitat auftritt, steht in einfachen Anführungszeichen.

Oft folgt eine wörtliche Wiedergabe einem Begleitsatz. Dieser erhält dann einen Doppelpunkt und das Zitat selbst einen Punkt oder Frage- oder Ausrufezeichen vor dem schließenden Anführungszeichen.

> *Der Entwickler behauptete: „Der Teufel steckt im Detail."*

Frage- und Ausrufezeichen, die zum Begleitsatz gehören, bleiben jedoch erhalten.

> *Wer sagte nochmal: „Um sein Ziel zu erreichen, zitiert selbst der Teufel aus der Bibel"?*

Zu beachten ist, dass das Satzschlusszeichen innerhalb der Anführung entfällt, es sei denn, es ist ebenfalls ein Frage- oder Ausrufezeichen:

> *William Shakespeare behauptete doch tatsächlich: „Hohle Töpfe haben den lautesten Klang!"!*

Wenn die wörtliche Wiedergabe einem Begleitsatz vorangeht, wird sie nicht durch einen Punkt abgeschlossen und nach dem schließenden Anführungszeichen steht ein Komma.

> *„Das Zitat ist der Sarg des Gedankens", behauptete einst André Brie.*

Eventuelle Frage- und Ausrufezeichen der wörtlichen Wiedergabe bleiben erhalten:

> *„Wird dem Huhn man nichts tun?", sorgt sich Christian Morgenstern in einem seiner Gedichte. „Hoffen wir es!", fügt er hinzu.*

Der Begleitsatz kann auch durch die wörtliche Wiedergabe umschlossen werden. Der Begleitsatz wird hier in Kommas gesetzt und das Satzschlusszeichen (Punkt, Ausrufe- oder Fragezeichen) steht vor dem abschließenden Anführungszeichen:

> *„Die Anzahl der Dummheiten", sagte einst André Gide, „die ein intelligenter Mensch im Lauf eines Tages sagen kann, ist unglaublich."*

Es kann aber auch der Begleitsatz die wörtliche Wiedergabe umschliessen. Dann steht sie nach einem Doppelpunkt, erhält keinen Schlusspunkt und nach dem schließenden Anführungszeichen ein Komma. Ein eventuelles Frage- oder Ausrufezeichen der wörtlichen Wiedergabe bleibt wiederum erhalten:

> *Wenn jemand nach Erhard Blanck zu dir sagt: „Nach einem Zitat zu greifen, ist allemal besser, als sich gar keine Gedanken zu machen", dann sollte dir das zu denken geben, wenn du gerade eines zum Besten gegeben hast.*

So, das soll für heute genügen. Jetzt wissen Sie zumindest schon einmal aus der Presse, wie peinlich es ist, beim Stehlen von geistigem Eigentum erwischt zu werden, ach Entschuldigung, ich meinte natürlich beim nicht ganz wissenschaftlichen Arbeiten. Und aus dieser Kolumne, wie sie die Satzzeichen setzten, wenn Sie korrekt arbeiten und fremdes Gedankengut angeben.

Kaffeesatz

Batteries not included

von Lisa-Marie Wehlmann-Wiedeking

Wenn man sich etwas Teureres zu Weihnachten wünscht, muss man entweder den Wunschzettel kürzer ausfallen lassen oder Andere überzeugen, dass sie das Gewünschte auch benötigen. Wenn man ein Einzelkind ist, hat man zwar keine Geschwister zur Verfügung, aber in diesem speziellen Fall konnte der Vater aushelfen. Ein Nintendo 3DS!

Im Allgemeinen bin ich mit meinem Vorgängermodell sehr zufrieden. Allerdings sind all die neuen *Must-Have*-Spiele nur auf dem 3DS anwendbar. Clever eingefädelt. Natürlich wollen richtige Spiel-Fans – oder beinahe Süchtige – die neuen Errungenschaften der Spieleentwickler ausprobieren und müssen sich so wohl oder übel den sündhaft teuren 3DS anschaffen.

Diese neuen Spiele sind alle in 3D, aber man kann den 3D-Effekt auch abstellen. Natürlich wollen diese Leute nur ihr neues Gerät verkaufen, deshalb kann man auch all die alten Spiele in 2D noch verwenden. Aber sie wissen auch, dass der Preis des Gerätes nah an der Schmerzgrenze liegt.

Aus diesem Grund haben sie sich etwas sehr raffiniertes ausgedacht: Die Spiele kosten um die vierzig Euro, das Gerät zweihundert.

Nun gab es zufällig um Weihnachten herum ein Paket, das für 220€ eines der absoluten Super-Spiele beinhaltete. Also 10% vom Gerät-Preis gespart oder auch 50% vom Spiel-Preis.

Sowas gefällt dem Weihnachtsmann und als unter dem Weihnachtsbaum in dem unvermutet kleinen Päckchen der neue DS zum Vorschein kam, war die Freude groß – auch als klargestellt wurde, dass er Vater und Kind gleichermaßen gehörte.

Wie gesagt ein unvermutet kleines Päckchen: Es war kein Ladekabel enthalten. Dies stand zwar auf der Verpackung, wurde dem zuständigen Weihnachtswichtel beim Kauf allerdings nicht gesagt. Also war das Gerät nach 5 Levels leer – und die Läden zwei Tage zu.

Als erstes kam uns die Idee, dass vielleicht das Kabel des alten Gerätes passen könnte, aber Fehlanzeige!

Dass das Ladekabel nicht enthalten ist und kein anderes passt, ist schon sehr ärgerlich!

Diese 10 € machen offensichtlich aus, ob sich jemand den Artikel gönnt oder nicht. Trotzdem. Ein böser Brief an Nintendo wäre doch sehr befriedigend, dafür, dass man zwei Tage vor dem Weihnachtsgeschenk sitzt und die Ausarbeitung der Kanten bewundern kann. Oder im Laden dem Verkäufer heimleuchten. Wenigstens der hätte sich ja mal rühren können.

Kurzbiographie

Lisa-Marie Wehlmann-Wiedeking ist vierzehn Jahre alt und Neuntklässlerin. Wenn sie ein schönes Thema findet, das in den KaffeeKlatsch passt, schreibt sie sehr gerne Artikel. Sie hat sich vorgenommen zu bloggen (allerdings nicht zu Software-Entwicklungs-Themen) und möglichst bald eines ihrer Bücher fertig zu stellen.

Jahresinhaltsverzeichnis 2012

Artikel

Januar

Schweizer Java
Eine Einführung in Scala, Teil 4

von Rüdiger Keller — Ausgabe 01/2012, Seite 6

Wie bereits im letzten Artikel angekündigt, beschäftigen wir uns diesmal mit der Standardbibliothek *Scalas*, um genau zu sein, mit der enthaltenen *Collections*-Bibliothek.

Mind the gap!
Plattformübergreifende mobile Entwicklung mit PhoneGap

von Werner Eberling — Ausgabe 01/2012, Seite 11

In keinem Bereich der Software-Entwicklung scheinen gerade so viele neue Projekte aus dem Boden zu sprießen, wie in der Welt der mobilen Endgeräte. Doch viele dieser Projekte haben mit der Problematik der heterogenen Systemlandschaft zu kämpfen. Dieser Artikel stellt mit *PhoneGap* eine Moglichkeit vor, systemübergreifend zu entwickeln, ohne auf den Zugriff auf Kamera, Kontaktliste oder GPS-Empfänger verzichten zu müssen.

Shakespeare fährt Taxi
Einführung in die esoterische Programmierung anhand der Sprachen Shakespeare und Taxi

Karina Hümpfner & Ramona Zeller — Ausgabe 01/2012, Seite 15

Was haben Shakespeare und Taxi mit Programmierung zu tun? Was bedeutet esoterische Programmiersprache? Und was ist eigentlich der Sinn dahinter? Oder gibt es am Ende gar keinen? Wer Antworten auf diese und weitere Fragen zum Thema sucht, wird in folgendem Artikel fündig werden.

Februar

JSF++;
Erfahrungen bei Upgrades

von William Siakam — Ausgabe 02/2012, Seite 7

In der Theorie oder besser gesagt im Traumfall ähnelt das Szenario eines JSF-Umstiegs auf eine höhere Version Folgendem: Die Anwendung herunterfahren; die alten JSF-Bibliotheken durch Neue ersetzen; die Anwendung neu bauen; starten – und alles läuft wie es soll. Schön wär's.

Password to go
Der persönliche Passwortgenerator für unterwegs, Teil 1

von Frank Goraus — Ausgabe 02/2012, Seite 11

Sicherheit ist heutzutage ja ein wichtiges Thema. Einen Teil davon machen persönliche Passwörter aus – Schlüsselworte, die den Zugriff auf geheime Daten nach Möglichkeit nur deren Besitzer erlauben sollen. Doch hier hapert es meist schon an zu einfach gewählten Passwörtern, die Unbefugte nicht nachhaltig genug ausschließen. Deshalb soll dieser zweiteilige Artikel aufzeigen, wie man sich schon mit einfachen programmatischen Mitteln sicherere Zugangsschlüssel erzeugen kann. In diesem ersten Teil geht es dabei um die Grundidee eines Verschlüsselungssystems für Passwörter.

März

Password to go
Der persönliche Passwortgenerator für unterwegs, Teil 2

von Frank Goraus — Ausgabe 03/2012, Seite 6

Sicherheit ist heutzutage ja ein wichtiges Thema. Einen Teil davon machen persönliche Passwörter aus. Im Prinzip Schlüsselworte welche den Zugriff auf geheime Daten nach Möglichkeit nur deren Besitzer erlauben sollen. Doch hier hapert es meist schon an zu einfach gewählten Passworten, welche Unbefugte nicht nachhaltig genug ausschließen. Im ersten Teil dieses Artikels haben wir uns deshalb mit einem System zur Erzeugung sicherer Zugangsschlüssel an sich beschäftigt. In diesem Teil wollen wir das Ganze nun in eine *Android*-App verpacken. Aber keine Angst, der Artikel ist zugleich als Einstieg in die Android-Entwicklung aufgebaut und richtet sich so vor allem an Android-Neulinge.

Android UI Design Patterns
(Einheits-)Gewänder für den Androiden, Teil 1 – Dashboard

von Johannes Köstler — Ausgabe 03/2012, Seite 14

Mit der wachsenden Anzahl von Apps im Google Play Store verringert sich für jede Einzelne der Zeitraum, in dem sie den Benutzer von sich überzeugen kann. Das Konkurrenzprodukt nur zwei Klicks entfernt wissend – muss eine Applikation vor allem intuitiv zu bedienen sein, um nicht vorschnell abgewiesen zu werden. Zur Erreichung dieses Zieles stehen mit den *Android UI Design Patterns* effektive Werkzeuge zur Verfügung.

April

Legendär

Eine kurze, unvollständige und größtenteils falsche Geschichte von Programmiersprachen

von James Iry — Ausgabe 04/2012, Seite 6

1801: Joseph Marie Jacquard nutzt Lochkarten, um einen Webstuhl *hallo welt* in einen Wandteppich weben zu lassen. Zeitgenössische *Reddit*-Nutzer zeigen sich unbeeindruckt, da *Tail-Call*-Optimierung, Nebenläufigkeit und Großbuchstaben fehlen…

Android UI Design Patterns

(Einheits-)Gewänder für den Androiden, Teil 2 – Action Bar

von Johannes Köstler — Ausgabe 04/2012, Seite 9

Beim *Action-Bar-Pattern* handelt es sich um einen wahren Überflieger. Das 2010 auf der Google I/O vorgestellte Design-Konzept wurde mit der *Android*-Version 3.0 (*Honeycomb*) eine eigene UI-Komponente des Android-Frameworks und löste die klassische Menüführung vollständig ab. Klingt erstmal super – wäre da nicht die Tatsache, dass über 90 Prozent der sich im Umlauf befindlichen Geräte noch mit einer Android-Version kleiner der Version 3.0 laufen. Aber eine gute Nachricht gleich vorweg – Sie müssen nicht mehrere Versionen Ihrer App entwickeln um alle Benutzer zu erreichen.

Der dunkle Ritter

Das Batman.js-Framework

von Andreas Schubert — Ausgabe 04/2012, Seite 15

Batman.js ist ein *Model-View-Controller*-Framework zur Erstellung von hoch dynamischen *Single-Page-HTML*-Applikationen. Es ist in *CoffeeScript* geschrieben und stellt ein mächtiges System fur *View-Bindings* und *Observable Properties* bereit. In diesem Artikel wird anhand einer kleinen Anwendung die Verwendung von Batman.js demonstriert.

Mai

Erwacht aus dem Winterschlaf

Anwendungsentwicklung mit dem Spring.NET-Framework, Teil 2

von Thomas Haug — Ausgabe 05/2012, Seite 8

Der Zugriff auf Datenbanken ist in vielen Enterprise-Anwendungen ein wesentlicher Bestandteil. Aus diesem Grund bietet das *Spring-Framework* hierfür ausgefeilte Möglichkeiten

Fensterln mit der Datenbank

SQL Window Functions

von Andreas Heiduk — Ausgabe 05/2012, Seite 14

Einfache *CRUD*-Operationen machen den Großteil der Datenbankzugriffe bei typischen Informationssystemen aus. *OR-Mapper* wie *Hibernate* oder *EclipseLink* können diese effizient und vor allem bequem behandeln. Dies verstellt oftmals den Blick dafür, wann es klüger wäre, doch direkt mit *SQL* zu arbeiten.

Projektwoche

Ein Online-Filmverleih mittels PHP und einer SQL-Datenbank

von Philipp Helmert — Ausgabe 05/2012, Seite 17

Jeder gelernte Fachinformatiker fur Systemintegration oder Anwendungsentwicklung kennt das. Im letzten Ausbildungsjahr wird meist im vorletzten Block eine Projektwoche angesetzt, in der sich die angehenden Netzwerkler oder Entwickler selbstständig an einem Projekt, für das sie eine Woche Zeit haben, beweisen können.

Juni

Typen wie du und ich

Typklassen vorgestellt

von Rüdiger Keller — Ausgabe 06/2012, Seite 6

Typklassen wurden für *Haskell* entwickelt, um Typen in Klassen einzuteilen. Seit einiger Zeit werden sie auch in *Scala* immer öfter eingesetzt. Dieser Artikel zeigt, was Typklassen sind und welche Unterschiede es zu den aus Java und Scala bekannten Klassen gibt.

(Einheits-)Gewänder für den Androiden

Teil 3 – Selection

von Johannes Köstler — Ausgabe 06/2012, Seite 9

Die vierte Android-Version *Ice Cream Sandwich* wurde im Juni mit dem „Parsons School of Design's User Experience Award" in Gold als „Best System Experience" ausgezeichnet. Einen nicht unwesentlichen Beitrag zu diesem Erfolg leistete dabei das überarbeitete Menükonzept. Nachdem der letzte Artikel die Ersetzung der Optionsmenüs durch die *Action Bar* beschrieben hat, widmet sich dieser Artikel der Ersetzung der Kontextmenüs durch die *Contextual Action Bar* und geht im Praxisbeispiel auf die Portierung der Contextual Action Bar auf Android-Versionen vor *Honeycomb* ein.

Nickeldioxid?

NIO.2 einmal vorgestellt

von Frank Goraus — Ausgabe 06/2012, Seite 17

Mit dem *JDK7* kamen mal wieder einige Neuerungen in Java hinzu. Eine davon ist eine neue *API* für die Arbeit mit Dateien und Dateisystemen, welche in diesem Artikel vorgestellt werden soll, und anhand kleiner Beispiele erläutert wird.

Juli

Ein Fall für REST

Einführung ins RESTful HTTP mit Fallstudie

von Róbert Bräutigam — Ausgabe 07/2012, Seite 6

REST hat schon den Gipfel der überzogenen Erwartungen in den Hype-Zyklus hinter sich. Weil der Gipfel aber noch nicht so fern zurückliegt, findet man immer noch widersprüchliche Informationen und nicht genügend Musterlösungen im Netz. Dieser Artikel befasst sich mit der Theorie von REST und demonstriert bestimmte *RESTful HTTP*-Lösungsmuster an einem Beispiel aus der Praxis.

Der Super-Duper-Happy-Pfad

2 ½ Blicke auf großartige Open-Source-Frameworks für .NET

von Timothée Bourguignon — Ausgabe 07/2012, Seite 13

Eine Web-Applikation zu schreiben sollte purer Spaß sein. Ein neues Projekt aufzubauen sollte reibungslos funktionieren ohne Gänsehaut beim Gedanken, verschiedene Frameworks und Tools zusammen zu führen. Die .NET-Welt außerhalb von Microsoft wuchert nicht so wild wie bei anderen Technologien, aber im *Open-Source*-Universum erscheinen doch ab und zu beeindruckende Projekte: *Nancy* und *Simple.Data* sind zwei davon. Werden diese zusammen und in Kombination mit *MongoDB* genutzt, entsteht bald der „Super-Duper-Happy-Pfad".

August

Web2touch

Web-Entwicklung für mobile Endgeräte mit jQuery Mobile

von Werner Eberling — Ausgabe 08/2012, Seite 5

Bei der Entwicklung einer Web-Anwendung kommt man in vielen Bereichen heute nicht mehr darum herum, sein Angebot auch für mobile Endgeräte zu optimieren. *JQuery Mobile* bietet hierfür eine interessante und leicht zu erlernende Plattform an, die in diesem Artikel vorgestellt werden soll.

Outsourcing war gestern! Die Zukunft liegt in der Crowd!

Wege und Trends in der Software-Entwicklung

von Mandy Goram — Ausgabe 08/2012, Seite 11

Einmal mehr treibt die Informationstechnologie die Entwicklung neuer Arbeitsstrukturen und -modelle in der Unternehmenswelt an. Bereits seit einigen Jahren herrscht das *Outsourcing* in den Unternehmen. Es führt zur Kostenreduktion als auch zur Mitarbeiterentlastung und Stärkung der Kernkompetenzen. Mit dem *Crowdsourcing* kommt einiges ins Rollen in Sachen flexibler Arbeitszeit.

Das Wowbagger-Syndrom

Typische Symptome bei Software-Entwicklern

von William Siakam — Ausgabe 08/2012, Seite 47

Auch wenn man sich nicht gut darin tut Menschen in Stereotypen zu klassifizieren und nach Vorurteilen zu gehen, stößt man erstaunlich oft auf wiederkehrende Verhaltensmuster bei bestimmten Menschengruppen. Worüber sich sagen lässt, dass es keinen Rauch ohne Feuer gibt. In diesem Fall geht es um eine relativ verbreitete Angewohnheit, die bei vielen Informatikern und insbesondere bei Software-Entwicklern feststellbar ist. Ich nenne es das Wowbagger-Syndrom.

September

Android-Applikationen von der Stange

Auslagerung wiederkehrender Code-Muster mit Hilfe der ADT-Templates

von Johannes Köstler — Ausgabe 09/2012, Seite 6

Die aktuelle Version des *Android Development Tools* (*ADT*) *Plug-ins* für *Eclipse* brachte neben tiefgreifenden Änderungen am *Layout-Editor* und einer verbesserten Unterstützung für *Android XML*-Dateien erstmals die Möglichkeit, bei der Erstellung von Projekten und *Activies* Code-Grundgerüste an Hand von *Templates* zu generieren. Dies erspart dem Entwickler einen gerade bei Android nicht geringen Anteil an *Boilerplate*-Code. Bisher existiert allerdings nur eine Handvoll Templates, die allesamt nicht verändert werden können. In der kommenden Version *21* des *ADT-Plug-ins* ist die Erstellung und Integration eigener Templates jedoch möglich, weshalb in diesem Artikel schon einmal ein Blick auf deren Struktur und Erstellung geworfen wird.

Oktober

Var-um?

Über Sinn und Unsinn variabler Typen in C#

von Tobias Krügel — Ausgabe 10/2012, Seite 6

Seitdem der Typ *var* das Licht der *C#*-Welt erblickte, setzt er seinen Siegeszug unaufhaltsam fort. Still und heimlich erobert er selbst den letzten Winkel des auch noch so eleganten Codes. Und niemand gebietet Einhalt. Doch was treibt Entwickler eigentlich dazu an, den Kernprinzipien der streng typisierten Sprache C# zumindest oberflächlich den Rücken zu kehren und von der expliziten Typdeklaration abzusehen? Dieser Artikel beleuchtet die Ursachen für die weite Verbreitung variabler Typen in C# und regt dazu an, sich Gedanken über deren tatsächliche Notwendigkeit zu machen.

Was bin ich?

Eigentypen in Java und Scala

von Róbert Bräutigam — Ausgabe 10/2012, Seite 9

Ohne tief in die Theorie von Typsystemen eingehen zu wollen, kann man in der Praxis trotzdem sehr schnell vor ganz interes-

santen Problemen stehen, die scheinbar nicht ohne Trickserei zu lösen sind. Ein gutes Beispiel hierfür ist das *Self-Type*-Problem.

November

Klettergarten
Common Table Expressions und rekursive Abfragen in SQL

von Andreas Heiduk Ausgabe 11/2012, Seite 8

Selbst in einfachen Schemata verstecken sich sehr oft einige hierarchische Strukturen wie Schritt, Unterschritt und Teilschritt. Ebenso oft werden diese mit *SQL* „platt geklopft" und ausprogrammiert. Dabei geht es auch anders: Flexibel!

Web-Anwendungen auf Klick
Einführung in Apache Click

von Rustam Khakimov Ausgabe 11/2012, Seite 12

Kein *MVC* und *JSP* sondern ein seitenorientiertes Design und „natürliche" *Rich-Client*-Programmierungsweise bei der Entwicklung von Web-Anwendungen. *Apache Click* bietet hierfür eine interessante Lösung. Dieser Artikel stellt das *Framework* vor und zeigt anhand von zahlreichen Beispielen Entwicklung und Test einer Web-Applikation.

Dezember

Nicht nur Spinnen bauen Netze
Web-Entwickung für Java-Entwickler

von Frank Goraus Ausgabe 12/2012, Seite 6

Jeder fängt mal klein an. Und so kann einen die Vielfalt an Technologien in der Web-Welt förmlich erschlagen. Hinzu kommt noch, dass man bei Web-Anwendungen teilweise anders an Probleme herangehen muss als dies bei *Desktop-/Rich Client*-Anwendungen der Fall ist. In diesem Artikel wird deshalb an einem kleinen Beispiel mit *Servlet*-Technologie gezeigt, was man alles benötigt um mit der Web-Entwicklung in *Java* zu beginnen.

In echt jetzt?
FnordMetric – App/Event-Tracking in Echtzeit

von Andreas Schubert Ausgabe 12/2012, Seite 12

FnordMetric ist ein Web-basiertes *App/Event-Tracking-System*, aufbauend auf *Redis* und *Ruby Eventmachine*, welches die Überwachung etwa einer Applikation in Echtzeit ermöglicht. Dabei ist *FnordMetric* sehr konfigurierbar und ziemlich schnell. Dieser Artikel soll einen kurzen Einblick in das Tool geben und zeigen wie man *FnordMetric* installiert, konfiguriert und einsetzt.

Kolumnen

Des Programmierers kleine Vergnügen

Duff's Device

von Michael Wiedeking Ausgabe 01/2012, Seite 20

Es ist immer wieder erstaunlich, wie einfallsreich die Menschen sind. So entlocken sie den Tools, die ihnen zur Verfügung stehen, die absonderlichsten Werke, die dem jeweiligen gewünschten Zweck mehr oder weniger dienlich sind. Das gilt natürlich auch für Programmierer und deren Programmiersprachen.

Überläufer

von Michael Wiedeking Ausgabe 02/2012, Seite 18

Wer einen Tageskilometerzähler im Auto hat, mag es schon einmal miterlebt haben: Einen Überlauf. Anstatt der beeindruckenden 1218 km von Nürnberg nach Hamburg und wieder zurück, zeigt der nach einem solchen beispielsweise nur noch bescheidene 218 km davon an. Dieses Schicksal bleibt einem gelegentlich auch bei Zahlen im Rechner nicht erspart, und so stellt sich die Frage, wie man diese Überläufe vorhersagen kann.

Vorzeichen wechsle dich

von Michael Wiedeking Ausgabe 03/2012, Seite 17

In *FORTRAN* gibt es die Funktion *isign(x, y)*, mit deren Hilfe sich das Vorzeichen von *y* auf *x* übertragen lässt. Wie immer lässt sich das mit einer bedingten Anweisung bewerkstelligen, aber der eifrige Vergnügen-Leser weiß inzwischen, dass Berechnungen dieser Art auch sehr leicht ohne Sprünge funktionieren.

Null, Zero, Nichts

von Michael Wiedeking Ausgabe 04/2012, Seite 19

Wenn man Probleme mit der Performanz hat, so kann es sich durchaus lohnen, auch auf Sonderfälle zu achten. Dazu gehört beispielsweise – wenn man es mit Gleitkommazahlen zu tun hat – der Vergleich mit Null.

Geteiltes Leid

von Michael Wiedeking Ausgabe 05/2012, Seite 21

Wer hat nicht schon einmal davon gehört, dass man eine Division durch eine Zweierpotenz einfach durch ein geeignetes Verschieben der Bits nach rechts ersetzen kann. Dies wäre auch wirklich eine tolle Sache, wenn es denn stimmen würde. Denn diese Aussage mag zwar für positive Zahlen stimmen, aber bei den negativen kann man durchaus seine Überraschung erleben.

Deutsch für Informatiker

Frage „Wie heißt Vorrundenaus auf Niederländisch?“ (Na, wissen Sie es?) eine schöne deutsche Wortschöpfung ist, nämlich: „Heimrobben.“ Gut, dass soll mein Beitrag zur EM 2012 sein.

Durchgebeugt

von Michael Wiedeking — Ausgabe 07/2012, Seite 18

Das Problem mit dem Duden ist, dass er einem zwar sagt, wie ein Wort grundsätzlich geschrieben wird, überlässt es dann aber der Phantasie, was man weiter damit macht. Zugegeben wird bei Substantiven noch der Genitiv und der Plural beschrieben, aber bei den Verben ist man doch mehr oder weniger auf sich selbst gestellt.

Fragen und Antworten

von Alexandra Specht — Ausgabe 08/2012, Seite 53

Uns hat ein sehr netter Leserbrief erreicht. In diesem wurden wir auch gefragt, wie gerade in der IT mit zusammengesetzten Wörtern verfahren wird. Das möchte ich gerne beantworten.

Herbstcampi, -campen, und -campoden

von Michael Wiedeking — Ausgabe 09/2012, Seite 15

Gerade erst ist der Herbstcampus vorbei, da stellt sich auch schon die Frage, wie er sich denn in die Reihe der anderen einreiht. Denn der diesjährige Herbstcampus war der bestbewertete Campus aller vorherigen … tja von was? Von den Campussen? Den Campi? Vielleicht den Campen? Oder gar den Campoden?

Best, bestens, am bestensten

von Alexandra Specht — Ausgabe 10/2012, Seite 15

Liebe Leserinnen und Leser, heute lassen wir einen geneigten Leser in dieser Rubrik zu Wort kommen. Wir erfahren auf amüsante Art, wie wir was besser nicht steigern. Aber lesen Sie selbst.

Eselsbrücken

von Michael Wiedeking — Ausgabe 11/2012, Seite 20

Wenn man sich mit einer Sprache beschäftigt, muss man auch mit deren Tücken zurechtkommen. Damit man sich gelegentlich etwas leichter mit den Besonderheiten tut, gibt es viele Eselsbrücken. Und diese gibt es natürlich nicht nur für fremde Sprachen, sondern auch für die Eigene.

Voll retro

von Michael Wiedeking — Ausgabe 12/2012, Seite 19

Natürlich wissen Sie, was ein Fernseher ist. Der Begriff ist vielleicht nicht ganz korrekt, denn man könnte ja auch den Fernsehzuschauer meinen. Aber hier ist das Fernsehgerät, der Fernsehapparat oder Fernsehempfänger gemeint.

Kaffeesatz

Leseprobleme

von Michael Wiedeking — Ausgabe 01/2012, Seite 25

Stellen Sie sich vor, Sie sind gerade in den Vereinigten Staaten gelandet, wollen durch die „Immigration“ und man lässt Sie nicht rein. Die Nummer Ihres Reisepasses, die Sie vor Ihrem Abflug in das ESTA-Online-Formular eingeben mussten, stimmt nämlich nicht mit der maschinengelesenen überein. Und das müssen Sie jetzt erst einmal erklären.

Von wegen „einfach“

von Michael Wiedeking — Ausgabe 02/2012, Seite 23

Neuerdings habe auch ich ein Smartphone. Vorher hatte ich ein Nokia 6310i, also noch eins mit Dieselmotor, aber schon mit Internet, wenngleich ich letzteres nie benutzt habe. Und ich kann sagen, dass mir der Umstieg nicht leicht gefallen ist.

Ferialzeit

von Michael Wiedeking — Ausgabe 03/2012, Seite 20

Das Internet mit seinen Einkaufsmöglichkeiten bietet schon viel Komfort. Braucht man doch, um beispielsweise in den Urlaub zu fahren, nicht unbedingt mehr die Hilfe eines Reisebüros. So lässt sich der Urlaub am heimischen Schreibtisch minutiös planen, alle Tickets buchen und sämtliche Hotels reservieren. Wenn nicht das latent ungute Gefühl dabei wäre, das man irgendetwas falsch gemacht hat.

Eingebürgert

von Michael Wiedeking — Ausgabe 04/2012, Seite 22

Das Computer-Programme inzwischen ganz schön clever sind, kann man in letzter Zeit auch immer häufiger der nicht fachlichen Presse entnehmen. Dabei ist es inzwischen völlig egal, ob sie bei Strategiespielen wie Schach, bei Spielen ums Allgemeinwissen wie Jeopardy gewinnen oder einfach nur selbstständig mit dem Auto umherfahren. Nun hat das erste dieser Programme die amerikanische Staatsbürgerschaft beantragt.

Qualitätsarbeit

von Michael Wiedeking — Ausgabe 05/2012, Seite 25

Alle zehn Jahre braucht man einen neuen Pass, neuerdings einen mit biometrischem Passbild. Ein biometrisches Passbild ist eines, dass einen besonders unvorteilhaft aussehen lässt. Wenn es gemacht wird, muss man möglichst unbeteiligt in die Kamera schauen und darf auf gar keinen Fall lachen.

Naturkonstante

von Michael Wiedeking Ausgabe 06/2012, Seite 25

Reisen ist ein Privileg. Und wenn selbst berufliche Reisen ein bisschen Zeit zum Umgucken lassen, so kann man sich nicht glücklicher schätzen. In diesem Zusammenhang begegnet einem natürlich auch immer wieder Software der unterschiedlichsten Art, mit der man aber in der Regel überraschend gut zurechtkommt.

Flash-Back

von Michael Wiedeking Ausgabe 07/2012, Seite 19

Wenn man durch das Silicon Valley fährt, ist das wie eine kleine Entdeckungsfahrt. An jeder Ecke findet man jemanden aus der IT-Branche, der Rang und Namen hat. Biegt man dort rechts ab, so steht man beispielsweise vor Yahoo, kurz dahinter ist dann Oracle und Google ist auch nicht weit. Und PerkinElmer, die ich ganz vergessen hatte.

No Photo!

von Michael Wiedeking Ausgabe 08/2012, Seite 54

Sommerzeit ist Reisezeit. Dieses Mal ging es, wenn auch nur kurz, nach Rom. Und wenn man schon einmal in Rom ist, empfiehlt sich – so man ins Museum gehen mag – auch ein Besuch der Vatikanischen Museen. Ich war zwar schon einmal dort, irgendwann in den achtziger Jahren, allerdings just zu der Zeit, als die Sixtinische Kapelle restauriert wurde. Also wollte ich dieses Mal nachholen, was ich damals versäumt hatte: Einmal die Sixtinische Kapelle in natura sehen. Allerdings stehe ich mit diesem Wunsch nicht alleine da: etwa vier Millionen Menschen kommen jährlich diesem Wunsch nach. Das sind bis zu 25 000 am Tag. Mir schien es auch, als wären genau so viele mit mir dort gewesen.

Wer hat's erfunden?

von Michael Wiedeking Ausgabe 09/2012, Seite 16

Wenn alles so bleibt wie es im Moment ist, dann wird Samsung eine Milliarde Dollar an Apple überweisen, vielleicht noch ein paar hundert Millionen dazu und möglicherweise kein Galaxy mehr in den USA verkaufen. Ich habe den Prozess nicht wirklich verfolgt, kann mich aber des Eindrucks nicht erwehren, dass da irgendetwas nicht mit rechten Dingen zugeht.

Unsachgemäßer Umgang

von Michael Wiedeking Ausgabe 10/2012, Seite 17

Das der sterbliche Benutzer eines Rechners gelegentlich Probleme mit dessen Umgang hat, ist meist dem darauf laufenden Betriebssystem geschuldet. Und für ebendiese Probleme sind wir Software-Entwickler meist die Ursache. Aber selbst wenn wir uns wirklich viel Mühe geben, bleibt immer noch das Gerät an sich, das man beliebig unsachgemäß behandeln kann.

Das Unsuchbare suchen

von Michael Wiedeking Ausgabe 11/2012, Seite 22

Stellen Sie sich vor, Sie gehen auf eine Bushaltestelle zu, zücken Ihr Handy und es sagt Ihnen, ob da jetzt ein Bus kommt. Oder Sie wollen zum Metzger und Ihr Telefon sagt Ihnen, wo gerade die kürzeste Schlange wartet.

Kaffeedatensatz

von Michael Wiedeking Ausgabe 12/2012, Seite 20

Eigentlich trinke ich ja keinen Kaffee und deshalb habe ich mich nie wirklich dafür interessiert. Trotzdem habe ich früher immer einen Filter und eine Kaffeekanne besessen, für den Fall der Fälle. Und wenn dann tatsächlich mal Besuch gekommen ist und dem nach Kaffee war, konnte mir meist meine Nachbarin aushelfen oder die Tankstelle.

Fehlerteufelchen

Auf Safari

von Sascha Gross Ausgabe 02/2012, Seite 16

Benutzer bewusst bei Web-Anwendungen auf eine Menge von *Browsern* zu beschränken ist nicht schön, aber durchaus gängige Praxis. Wenn aber *Safari-Surfer* unbewusst ausgeschlossen werden, ist das schon ärgerlich.

Ungewolltes Singleton mit Spring in einer Web-Anwendung

von Sascha Gross Ausgabe 08/2012, Seite 49

Ein *Singleton* zu schreiben, ist gar nicht so einfach, oder anders ausgedrückt, man kann dabei ziemlich viele Fehler machen. Hier soll es aber nicht darum gehen, welche Fehler man machen kann, sondern wie man es schafft das Verhalten eines *Singleton* ungewollt in einer *Java*-Web-Anwendung mit *Spring* nachzustellen.

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)

JUG DA
Java User Group Darmstadt
http://www.jug-da.de
Kontakt: javausergroupdarmstadt@googlemail.com

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
http://www.rheinjug.de
Kontakt: Herr Heiko Sippel (info@rheinjug.de)

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
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
(alexander.culum@web.de)

JUG Deutschland e.V.
Java User Group Deutschland e.V.
c/o asc-Dienstleistungs GmbH
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
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
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)
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
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)

Schweiz

JUGS
Java User Group Switzerland
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
c/o GEDOPLAN GmbH

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

.NET User Group Dortmund (Do.NET)
c/o BROCKHAUS AG
http://do-dotnet.de
Kontakt: Paul Mizel (pmizel@do-dotnet.de)

Die Dodnedder
.NET User Group Franken
http://www.dodnedder.de
Kontakt: Herr Udo Neßhöver, Frau Ulrike Stirnweiß
(dodned@googlemail.com)

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

.NET DGH
.NET Developers Group 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
http://www.dnug-koeln.de
Kontakt: Herr Albert Weinert (info@der-albert.com)

.NET User Group 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
http://www.dotnet-oldenburg.de
Kontakt: Herr Hilmar Bunjes
(hilmar.bunjes@dotnet-oldenburg.de)
Frau Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)

.NET User Group Paderborn
c/o Net at Work Netzwerksysteme GmbH,
http://www.dotnet-paderborn.de
(raacke@dotnet-paderborn.de)
Kontakt: Herr Mathias Raacke

.NET Developers Group Stuttgart
Tieto Deutschland GmbH
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
Kontakt: Frau Catrin Busley

.NET Developer-Group Ulm
c/o artiso solutions GmbH
http://www.dotnet-ulm.de
Kontakt: Herr Thomas Schissler (tschissler@artiso.com)

Österreich

.NET Usergroup Rheintal
c/o Computer Studio Kogoj
http://usergroups.at/blogs/dotnetusergrouprheintal/default.aspx
Kontakt: Herr Thomas Kogoj (thomas@kogoj.com)

.NET User Group Austria
c/o Global Knowledge Network GmbH,
http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx
Kontakt: Herr Christian Nagel (ug@christiannagel.com)

Software Craftsmanship Communities

Deutschland

Softwerkskammer – Mehrere regionale Gruppen unter einem Dach, http://www.softwerkskammer.de

MATHEMA

meet the experts of enterprise infrastructure

Neues in Java 7

Die nächste Java Generation,
4. – 5. März 2013, 2. – 3. Mai 2013,
835,– € (zzgl. 19 % MwSt.)

Objektorientierte Analyse und Design mit UML und Design Patterns

Methoden und Prinzipen für die Entwicklung von OO-Modellen und die Dokumentation mit der UML
2. – 4. April 2013, 22. – 24. Juli 2013,
1.180,– € (zzgl. 19 % MwSt.)

iPhone Programmierung

Mobile Anwendungen für das Apple iPhone
10. – 12. April 2013, 24. – 26. Juni 2013,
1.180,– € (zzgl. 19 % MwSt.)

Enterprise JavaBeans (EJB)

Enterprise JavaBeans im Detail
8. – 12. April 2013, 29. Juli – 2. August 2013,
1.870,– € (zzgl. 19 % MwSt.)

Sicherheitskonzepte unter Java

Mechanismen für Datenschutz und Datensicherheit in Netzwerken
27. – 28. März 2013,
19. – 20. Juni 2013,
925,– € (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

MATHEMA

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?

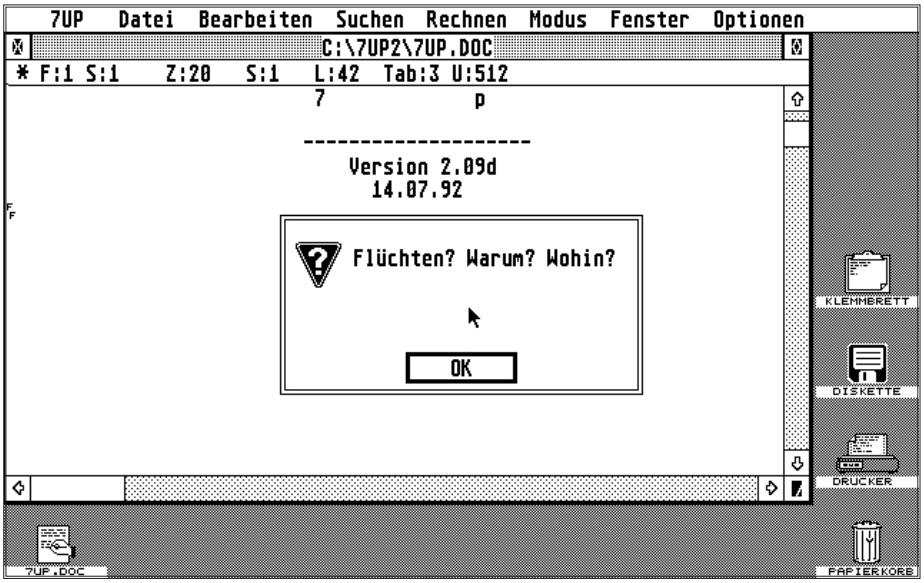
Wir bieten Ihnen erstklassigen Wissensaustausch, ein tolles Umfeld, spannende Projekte in den unterschiedlichsten Branchen und Bereichen sowie herausfordernde Produktentwicklung.

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

Das Allerletzte



In den Achtzigern gab es diesen fantastischen Atari ST, und dazu eine Menge hochqualitativer Software. Unter anderem gab es einen gut ausgebauten Texteditor, dessen Autor eine gehörige Portion Humor besaß, und hoffentlich immer noch besitzt. Wenn man auf die Escape-Taste tippte, erschien die im Screenshot gezeigte Alertbox.

Von Bodo Wenzel

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 im Februar.

Herbstcampus

Wissenstransfer par excellence

2.– 5. September 2013

in Nürnberg