02-2016 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

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

Java ist Programmiersprache des Jahres

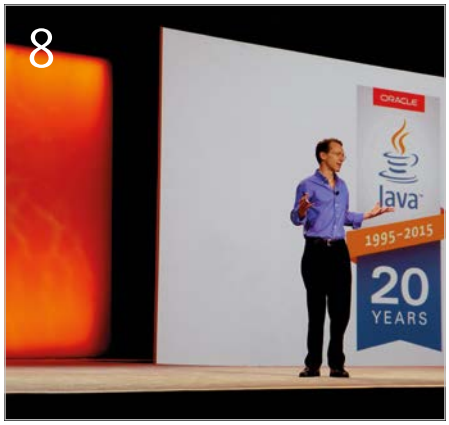
Sicherheit
- Social Login
- Single Sign-on

Richtig testen
Statische Code-Analyse

Richtig entwickeln
Lösung mit JavaFX und JVx

4 191978 304903 02

iJUG Verbund



Neues von der JavaOne

Die gängigsten Open-Source-Tools zur Code-Analyse im Praxis-Einsatz

Reaktive Anwendungen mit asynchronen, Event-getriebenen Architekturen gewinnen stark an Bedeutung

Effiziente Software-Entwicklung mit JavaFX und JVx

Roland Hörmann, SIB Visions GmbH

Vor Kurzem wurde der Autor von einem großen deutschen IT-Dienstleister mit der Aussage konfrontiert, dass die Entwicklung von Backoffice-Lösungen nicht kostengünstig und zeitnah durchführbar sei. Das Problem wurde sowohl in aktuellen Technologie-Entwicklungen als auch in organisatorischen Abläufen vermutet.

Die Wunschvorstellung wäre gewesen, dass die Umsetzung von Anforderungen im Backoffice-Bereich nicht länger als die Erstellung von Excel-Sheets in Anspruch nehmen sollte. Wer an dieser Stelle denkt, dass Scrum und andere agile Methoden die Effizienz gravierend verbessern, der sollte das stark anzweifeln. Die Effizienz wird schlussendlich an der Umsetzungszeit der Kunden-Anforderungen gemessen. Mit den richtigen Mitteln ist das alles kein Problem.

Doch welche Mittel sind das genau? Seit Längerem geht der Trend in Richtung Web-Entwicklung mit geeigneten Full-Stack-JavaScript-Frameworks oder leichtgewichtigen Bibliotheken. Es besteht natürlich auch Bedarf an Desktop- und nativen mobilen Anwendungen. Für alle Plattformen gibt es ausreichend Frameworks und Bibliotheken, nachdem das Angebot so riesig ist, fällt die Auswahl der richtigen Mittel extrem schwer.

Selbst wenn die richtige Technologie gefunden ist, bietet das noch keine Garantie für Effizienz, denn es braucht mehr als nur Frameworks und Bibliotheken, um effizient zu sein. Wenn unstrukturiert, ohne standardisiertes Vorgehen und ohne Rücksicht auf die Qualität entwickelt wird, kann nicht von Effizienz gesprochen werden.

Das richtige Mittel muss also ein Mix aus Technologie(n) und einer standardisierten Vorgehensweise bei höchsten Qualitätsansprüchen sein. Am besten ist alles in einem leichtgewichtigen Framework vereint. Wer den heiligen Gral in der Software-Entwicklung vermutet, liegt damit gar nicht so falsch. Doch alles schön der Reihe nach.

In den Gesprächen mit dem IT-Dienstleister stellte sich heraus, dass vom Wunsch

```
CRUDTable table = new CRUDTable();

Grid grid = new Grid();
grid.setTable(table);

Screen screen = new Screen();
screen.add(grid);
screen.show();
```

Listing 1: Pseudo-Code „Bearbeitungsmaske"

nach effizienter Entwicklung von Desktop-Anwendungen die Rede war. In einigen wenigen Fällen waren Desktop-Anwendungen nicht ausreichend und daher Web-Anwendungen gefragt. Die Erstellung von mobilen Clients war nicht weiter wichtig, es war aber klar, dass sich das in den nächsten Monaten ändern könnte.

Als Beispiel nannte der Dienstleister eine Backoffice-Lösung die rund hundert Bearbeitungsmasken hatte, mit denen die Verwaltungsaufgaben eines Unternehmens durchgeführt wurden. Die Masken selbst hatten Funktionen, die als De-facto-Standard bezeichnet werden können, wie CRUD, Volltextsuche, Sortiermöglichkeiten und CSV-Export. Die Erstellung von Berichten fehlte in der Anwendung natürlich nicht. Es war auch schon selbstverständlich, dass eine Backoffice-Anwendung über eine geeignete Benutzerführung verfügt beziehungsweise benutzerfreundlich ist, dass ein einheitliches Erscheinungsbild sein muss und dass ein Hilfesystem nicht fehlen darf – natürlich sollte alles mehrbenutzerfähig und mehrsprachig sein.

Aus Sicht des IT-Dienstleisters war die Anwendung nichts Besonderes und üblich in der Branche. Aus der Sicht eines Entwicklers sieht die Sache natürlich ganz anders aus, denn mehr Funktionalität bedeutet auch eine längere Umsetzungszeit. Und genau das war dem IT-Dienstleister ein Dorn im Auge. Die Erstellung von Anwendungen mit den genannten Funktionen sollte in wenigen Tagen anstatt mehreren Monaten möglich sein. Dieser Wunsch wird auch in Zukunft immer häufiger zu hören sein. Denn die Anwendungsentwicklung ist ein enormer Kostenfaktor. Doch wie können solche Wünsche erfüllt werden?

Der Faktor „Mensch" darf beim Thema Effizienz nicht vernachlässigt werden, denn die Grundvoraussetzung effizienter Arbeit ist ein geeignetes Arbeitsumfeld, um beim Entwickeln in den richtigen „Coding Flow" zu kommen, der im Idealfall nicht unterbrochen wird. Soweit die Theorie, doch leider ist die Praxis nicht immer ideal. Aus diesem Grund müssen andere Faktoren helfen, um die Effizienz zu steigern. Eine geeignete Maßnahme wäre, ganz einfach weniger Code zu schreiben. Denn weniger Code reduziert den Aufwand für Test, Wartung und Weiterentwicklung.

Statistisch gesehen passieren auch weniger Fehler, wenn weniger Code erzeugt wird. Es bedeutet zwar nicht, dass der Code weniger komplex ist, aber auf jeden Fall ist er einfacher und schneller zu lesen, es gibt also eine raschere Fehleranalyse. Wenn von weniger Code gesprochen wird, dann ist damit nicht nur gemeint, Leerzeilen zu löschen, Variable kürzer zu benennen und Zeilenumbrüche zu entfernen. Das Ergebnis wäre zwar kürzer, aber auch unbrauchbar. Eine Code-Reduktion lässt sich durch Zentralisierung von Funktionalitäten oder den Einsatz von hilfreichen Paradigmen wie beispielsweise Don't Repeat Yourself (DRY) *[1]* oder dem Ableger Convention over Configuration (CoC) *[2]* erreichen.

Mit Zentralisierung ist nicht einfach nur gemeint, dass ähnliche Funktionen in eine zentrale Methode ausgelagert werden, um Code zu vermeiden. Dies sollte selbstverständlich sein. Überwiegend ist gemeint, dieselben Informationen nicht an mehreren Stellen in einem Projekt zu definieren. Es muss eine zentrale Stelle geben, an der diese zu finden sind, um bei Änderungen alle logisch verwandten Informationen implizit anzupassen. Mit dem DRY-Prinzip kann dies erreicht werden. Die Anwendung des Prinzips ist vom Entwickler abhängig und erfordert Konsequenz. Um den Entwickler zu entlasten, sollte auch CoC Berücksichtigung finden, denn dadurch kann auch ohne strikte Konsequenz der Sourcecode reduziert werden.

Eine Konvention ist dabei nichts anderes als eine übliche Funktionalität, die ohne Code oder Konfiguration einfach implizit vorhanden ist. Damit ist für eine Konvention einfach kein Code notwendig. Aber natürlich passen nicht immer alle Konventionen hundertprozentig. Deswegen muss es möglich sein, die Abweichung zu codieren. Das darf allerdings nicht die Regel sein, denn sonst muss die Konvention überarbeitet werden.

Durch die Anwendung von DRY und CoC kann beispielsweise eine einzelne Bearbeitungsmaske in wenigen Code-Zeilen umgesetzt werden, mit den zuvor beschriebenen Funktionalitäten (CRUD, CSV Export, Volltextsuche …). *Listing 1* zeigt mit Pseudo-Code, wie eine Maske codiert werden könnte.

Eine Konvention besagt, dass mit der „CRUDTable" ohne weitere Codierung Datensätze gelesen, erstellt, geändert und gelöscht werden können. Wenn bestimmte Operationen nicht möglich sein sollen, müssen sie deaktiviert werden, etwa mit „setInsertEnabled(false);". Eine weitere Konvention besagt, dass ein Grid automatisch alle Daten laut „CRUDTable" anzeigen, sortieren, durchsuchen und als CSV-Datei speichern kann.

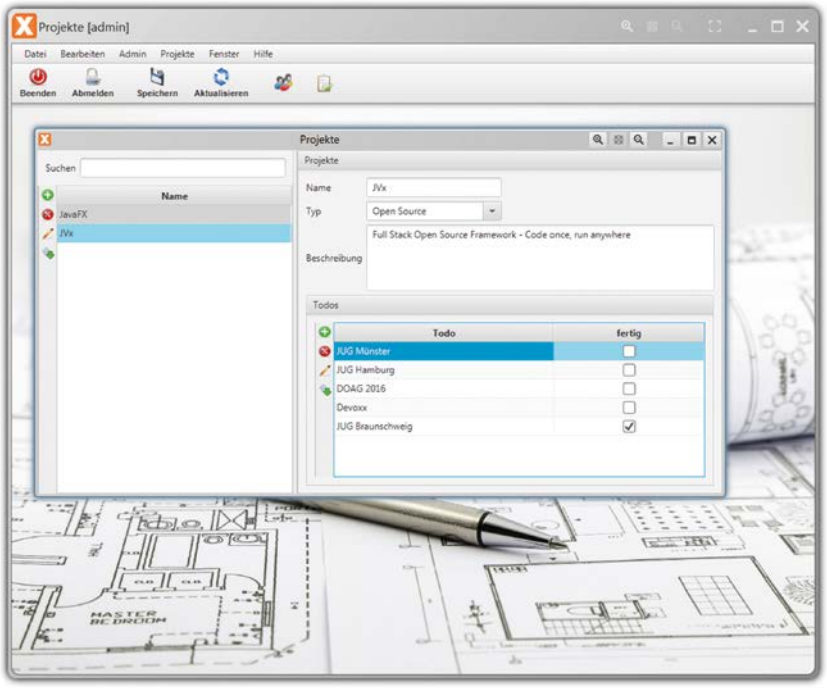


Abbildung 1: Projekt-Verwaltung als JavaFX-Applikation

Das Beispiel zeigt, dass nicht sehr viel Sourcecode nötig ist, um eine Maske zu erstellen. Es können auch nicht viele Fehler im eigenen Code sein, da durch die Anwendung der Konventionen die Komplexität vom Entwickler ferngehalten wird. Erst mit Sonderwünschen, also den Abweichungen von den Konventionen, und durch die Business-Logik würde zusätzlicher Sourcecode entstehen. Ein Entwickler muss sich jedoch ausschließlich darum kümmern. Das erleichtert einerseits die Arbeit und erhöht andererseits die Effizienz.

Theorie und Praxis

Die Anwendung von Software-Paradigmen ist natürlich mit jeglicher Technologie möglich und effiziente Software-Entwicklung sollte üblich sein. Leider ist das nur die Theorie und in der Praxis aufgrund diverser Einflüsse nicht die Regel, wie auch die Gespräche mit dem IT-Dienstleister verdeutlichten.

Nur einige wenige Frameworks wenden die zuvor genannten Paradigmen an und haben diese verinnerlicht. Erst durch den Einsatz solcher Frameworks werden Entwickler entlastet und können maximal effizient arbeiten. Eines dieser Frameworks ist das Open-Source-Java-Framework JVx *[3]*. Es wurde auf den Grundprinzipien von Best-Practice-Paradigmen erstellt und hat zum Ziel, den Sourcecode einer Anwendung auf ein Minimum zu reduzieren.

Doch JVx selbst ist kein GUI-Framework und benötigt etwas Unterstützung. An dieser Stelle kommt JavaFX ins Spiel. Als offizieller Nachfolger von Swing ist JavaFX bestens für die Erstellung von Backoffice-Anwendungen geeignet und enthält alle notwendigen GUI-Komponenten wie „TableView". Auch wenn JavaFX ganz gut ohne JVx zurechtkommt, müssten alle Paradigmen vom Entwickler selbst angewandt werden. Doch das widerspricht der gewünschten Effizienz.

Effizienz in der Praxis

Ein konkretes Beispiel mit JavaFX und JVx zeigt, wie effizient eine Bearbeitungsmaske *(siehe Abbildung 1)* erstellt werden kann, die noch dazu alle Anforderungen abdeckt. Im linken Bereich der Maske ist nun das Grid mit der Liste aller Projekte ersichtlich. Passend zum Pseudo-Code aus *Listing 1* ist es möglich, mittels Suchfeld (oberhalb des Grid) nach Projekten zu filtern. Über die Buttonleiste links vom Grid können neue Projekte erstellt, bestehende geändert, gelöscht oder als CSV-Datei exportiert werden. Im rechten Bereich befindet sich die Detail-Ansicht mit dem Namen, dem Typ und der Beschreibung zum aktuell gewählten Projekt. Darunter wird eine weitere Liste mit To-dos dargestellt. Der Funktionsumfang der Maske ist in ERP-Systemen durchaus üblich.

Ausgehend von *Listing 1* werden der Sourcecode vervollständigt und die Detail-Ansicht ergänzt. Dazu muss die „CRUDTable" mit Daten in Verbindung gebracht werden. Dies wird mit „table.setTable("PROJEKTE");" gelöst. Dadurch wird die Datenbanktabelle PROJEKTE mit dem Grid verknüpft. Mehr Code ist nicht notwendig, um vollständiges „CRUD" zu ermöglichen.

```
EditorLabel name = new EditorLabel(table, "NAME");
EditorLabel typ = new EditorLabel(table, "TYP");
EditorLabel beschreibung = new EditorLabel(table, "BESCHREIBUNG");
screen.add(name);
screen.add(typ);
screen.add(beschreibung);
```

Listing 2: Pseudo-Code der Detail-Ansicht

```
CRUDTable todos = new CRUDTable("TODOS");
Grid gridTodos = new Grid(todos);
gridTodos.setFilterable(false);
screen.add(gridTodos);
```

Listing 3: Pseudo-Code für To-dos in abgekürzter Schreibweise

```
todos.setMasterReference(table,
     new String[] {"PROJEKTE_ID", "ID"}
```

Listing 4: Master/Detail-Beziehung herstellen

Im nächsten Schritt wird die Detail-Ansicht eingefügt. Dabei handelt es sich um drei Editoren mit je einem Label und einem Grid, das jedoch keine Filterung ermöglicht. *Listing 2* enthält den Sourcecode für die Editoren. Das Grid mit den To-dos wird in *Listing 3* hinzugefügt.

Die Bindung der Editoren an die Projekte erfolgt mit der „CRUDTable" und der Angabe des Spaltennamens, aus dem die Daten gelesen und gespeichert werden. Durch CoC wird festgelegt, dass das Label aus dem Spaltennamen gebildet wird. Somit wird aus „NAME" das Label „Name". Natürlich muss auch das Label durch den Entwickler veränderbar sein (siehe „name.setLabel("Projektname");"), aber wenn die Spalten-Bezeichnung gut gewählt wurde, sollte die Konvention sehr gut funktionieren.

Die Verknüpfung der Editoren mit der Tabelle wurde bereits durchgeführt. Es fehlt nun noch die Verknüpfung der To-dos mit den Projekten, da nur die To-dos des gewählten Projekts angezeigt werden sollen. Im Fachjargon wird von Master/Detail-Beziehung gesprochen. Um die Verknüpfung herzustellen, ist der Code aus *Listing 4* vollkommen ausreichend.

Die Sourcecode-Zeile besagt, dass alle To-dos, bei denen der Wert der Datenbankspalte „PROJEKTE_ID" mit der Spalte „ID" aus der Projekte-Tabelle übereinstimmt, mit einem Projekt verknüpft werden. In knapp zwanzig Zeilen Sourcecode wurde die Maske umgesetzt; der Zeitaufwand von knapp fünf Minuten unterstreicht die Effizienz. Im konkreten Beispiel bedeutet wenig Sourcecode auch weniger Fehler und minimalen Testaufwand.

Da geht doch noch mehr

Die Maske enthält einige Details, die im Sourcecode und im Screenshot nicht ersichtlich sind. Der Editor für den Projekttyp ist eine Auswahlliste, obwohl wir keine definiert haben. Im Grid mit den To-dos wird eine Checkbox für die Spalte „FERTIG" angezeigt, obwohl auch diese nicht explizit festgelegt wurde.

Der Projektname ist in der Datenbank auf hundert Zeichen begrenzt, der Editor erlaubt ebenfalls nicht mehr. Ohne CoC müssten all diese Features manuell codiert werden. Das würde zu mehr Sourcecode führen und ein Entwickler wäre damit einige Stunden beschäftigt. Diese Zeit kann sicherlich besser eingesetzt werden, um gegebenenfalls die Business-Logik zu codieren.

Damit aus dem Projekttyp eine Auswahlliste wird, können die Metadaten der Datenbanktabelle ausgelesen werden. Durch eine Fremdschlüssel-Beziehung ist eigentlich ganz klar definiert, woher die Daten für ein bestimmtes Feld bezogen werden können. Auch die Längenbeschränkung des Projektnamens ist bereits in den Metadaten der Datenbanktabelle verankert. Die Checkbox für To-dos ergibt sich ebenfalls aus den Metadaten, wenn die möglichen Werte in der Datenbank hinterlegt wurden (Check-Constraints oder Enum-Datentyp). Wenn die Projekte noch über ein Fertigstellungsdatum verfügen würden, wäre auch damit klar definiert, dass ein Datum im Editor angezeigt werden muss.

Fazit

Gerade bei Backoffice-Lösungen oder generell großen Datenbank-Applikationen sollten die hier beschriebenen Methoden in Betracht gezogen werden. Damit können der Code und die Komplexität von Applikationen soweit reduziert werden, dass Applikationen mit beispielsweise zweihundert Masken von einem Team mit zwei Entwicklern in wenigen Monaten umgesetzt werden können – anstatt in einem Jahr. Die Praxis hat gezeigt, dass die Wartungsfälle mit beschriebener Vorgehensweise kaum der Rede wert sind und die überschüssige Zeit in neue Projekte investiert werden kann.

Links

[1] DRY: *https://de.wikipedia.org/wiki/Don%E2%80%99t_repeat_yourself*

[2] CoC: *https://en.wikipedia.org/wiki/Convention_over_configuration*

[3] JVx: *http://sourceforge.net/projects/jvx*

Roland Hörmann

roland.hoermann@sibvisions.com

Roland Hörmann ist Gründer und CEO der SIB Visions GmbH sowie Lektor an der FH Technikum Wien. Er verfügt über langjährige Erfahrung in der Entwicklung von Enterprise-Lösungen. Sein Interessenschwerpunkt liegt auf effizienter Software- und Framework-Entwicklung im Umfeld von klassischen Business-Applikationen. Er ist regelmäßig Speaker auf Oracle-User-Group-Veranstaltungen (DOAG, AOUG) sowie auch bei Java-User-Group-Events.

Elasticsearch – ein praktischer Einstieg

gelesen von Daniel Grycman

Auf dem Buchrücken versprechen der Autor Florian Hopf und der dpunkt Verlag, dass der Leser nach der Lektüre das nötige Rüstzeug besitzt, um eigene Anwendungen auf Basis von Elasticsearch zu entwickeln. Diese Rezension beleuchtet, ob das auch wirklich gelungen ist.

Das Buch ist in zwölf Kapitel aufgeteilt. Den Abschluss bilden zwei Anhänge, das Literaturverzeichnis und der Index. Gleich zu Beginn geht Florian Hopf auf die Geschichte von Elasticsearch ein und schafft dabei einen Bezug zu Lucene und Solr, wobei er gleichzeitig eine Abgrenzung zu Solr vornimmt.

Die ersten vier Kapitel führen in die grundlegenden Konzepte und Basics von Elasticsearch und Lucene ein, ebenso vertieft der Autor den Umgang mit Texten und Relevanz-Berechnung. Es folgt ein Kapitel zur Indizierung von verschiedenen Datenquellen. Das sechste Kapitel beschäftigt sich mit allen Aspekten der Verteilung von Daten. Die zwei folgenden Kapitel gehen auf die Modellierung und Aggregation von Daten ein. Im neunten Kapitel wird der Zugriff auf Elasticsearch mittels Java und JavaScript beschrieben.

„Elasticsearch in Produktion" ist das Thema des zehnten Kapitels, wobei Florian Hopf nicht nur die Inbetriebnahme erläutert, sondern auch Themen wie „Monitoring" und „Datensicherung" Platz gibt. Das vorletzte Kapitel widmet sich der Speicherung von Logging-Dateien, wozu Elasticsearch als Teil des ELK-Stacks benutzt werden kann. Zusätzlich wird Graylog als Alternative präsentiert. Das zwölfte und zugleich letzte Kapitel gibt einen Ausblick in die Zukunft von Elasticsearch und stellt in aller Kürze einige Themen vor, auf die im Buch aufgrund ihrer Komplexität nicht eingegangen werden kann.

Bemerkenswert ist das Expertenwissen von Florian Hopf zu Apache Solr, Lucene und Elasticsearch. Es ist ein Buch für Praktiker aus Sicht eines Praktikers. Es finden sich viele Beispiele, die zum direkten Ausprobieren einladen. Das Buch basiert auf der Version 1.6 von Elasticsearch, was für dieses Einsteigerwerk kein Problem darstellt, da in den neueren Versionen Funktionen aus der Version 1.6 erhalten geblieben sind.

Fazit: Dieses Werk eignet sich für den Leser, der auf der Suche nach einem deutschsprachigen Einsteigerbuch für Elasticsearch ist. Mithilfe der Lektüre sollten sich erste Projekte in Elasticsearch zeitnah umsetzen lassen.

Titel:	Elasticsearch
Autor:	Florian Hopf
Verlag:	dpunkt Verlag
Umfang:	262 Seiten
Preis:	32,90 Euro, auch als eBook erhältlich
ISBN:	978-3-86490-289-5