01-2015 | Frühling | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

Fünf Jahre Java aktuell

Praxis
Mehr Typsicherheit mit Java 8

Programmieren
Der selbe GUI-Code für Desktop, Web und native mobile App

SAP HANA
Was für Java-Anwendungen drin ist

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

4 191978 304903 01

iJUG Verbund

Happy Birthday Java aktuell

Wege zum Aufbau von modernen Web-Architekturen

Groovy-Entwickler argumentieren, dass Java in die Jahre gekommen sei

Java – gestern, heute, morgen

Java in den letzten fünf Jahren: Eine gesetzte Person, die ihre Midlife-Crisis überwunden hat und mit neuem Schwung die nächsten Jahre angeht. Während Java im Jahr 2009 noch eine feste Größe war und es wenig Fortschritt gab, konnte Oracle nach einigen Unruhen in der Community aufgrund der Sun-Übernahme dann doch zeigen, dass es auf die Community hört.

Trends waren: JVM als Plattform für unterschiedliche Sprachen; Öffnung für dynamische Sprachen wie JRuby und Groovy, aber auch funktionale wie Clojure und Scala. Nicht zuletzt die funktionale Erweiterung durch Lambdas in Java selber.

Java EE ist leichtgewichtiger geworden. In der Zukunft wird sich hier noch einiges tun: Von der Vereinheitlichung der unterschiedlichen Komponenten-Modelle zu einem einzigen über die stärkere Unterstützung moderner HTTP-2.0-Möglichkeiten bis hin zu asynchroner Kommunikation, angetrieben durch das Internet der Dinge.

Polyglot Programming und Polyglot Persistence werden noch wichtiger werden und Anforderungen aus dem Bereich „Continuous Delivery" dazu führen, dass leichtgewichtiges und entwicklerfreundliches Arbeiten weiter an Bedeutung gewinnt, wofür heute ja schon DropWizard oder Spring Boot sorgen. Für Architekten und Entwickler wird es eine Herausforderung sein, die Breite des Java-Ökosystems noch zu überblicken, sodass ich hier eine vermehrte Spezialisierung erwarte. Es liegt also eine spannende Zeit vor uns!

Richard Attermeyer
richard.attermeyer@opitz-consulting.com

Richard Attermeyer ist Senior Solution Architects bei der OPITZ CONSULTING Deutschland GmbH. Er beschäftigt sich seit vielen Jahren als Entwickler, Architekt und Coach mit den Themen „Enterprise Applikationen" und „Agile Projekte".

http://ja.ijug.eu/15/1/5

Java 2014 ± 5

Die Entwicklung von Java sollte nicht nur unter technischen Gesichtspunkten betrachtet werden. Je erfolgreicher sich eine Programmiersprache etabliert, desto stärker gewinnen übergreifende Faktoren an Einfluss. Diese kommen in der öffentlichen Diskussion häufig zu kurz.

Ein wichtiger Punkt, der gemeinhin mit einem Achselzucken abgetan wird, ist der, dass sich der Abstand zwischen den beiden wesentlichen Gruppen der Java-Anwender weiter vergrößert hat. Während auf der einen Seite aktiv und engagiert an der Fortentwicklung der Sprache und ihrer Umgebung gearbeitet wird, gibt es auf der anderen Seite eine große Gruppe von Anwendern, die ihre Aufgaben mit den lange etablierten Mitteln lösen müssen.

In gewisser Weise lässt sich dieser Abstand sogar messen, wenn man die jeweils aktuelle Java-Version mit der in den Projekten eingesetzten vergleicht. Allerdings ist das nur die halbe Wahrheit, weil die Verwendung einer bestimmten Java-Version noch nicht bedeutet, dass die für diese adäquaten Techniken auch verwendet werden. Java ist mit jedem Release gewachsen und komplexer geworden. Dementsprechend müsste sich zum Beispiel die Dauer von Schulungen seit dem Jahr 2000 mindestens verdoppelt haben, um den gleichen Überblick über die gesamte Java-Plattform zu vermitteln. Da das jedoch (gewöhnlich) nicht erfolgt ist, bleibt es oft der Initiative des Einzelnen überlassen, sich diese Kenntnisse anzueignen – oder auch nicht.

Dabei ist es ganz natürlich, dass unter wirtschaftlichen Gesichtspunkten die Entwicklung großer und aufeinander aufbauender Anwendungssysteme nicht dem relativ kurzen Release-Zyklus von drei Jahren folgen kann. Wie in anderen Bereichen (siehe Linux) wird wohl auch für Java das Schlagwort „Long Term Support" (LTS) an Bedeutung gewinnen. Anwendungen werden auch in Zukunft zwanzig oder dreißig Jahre laufen und über die Jahre wachsen. Das ist keine Innovationsfeindlichkeit oder Trägheit – dafür gibt es objektive Gründe.

Wie schwierig es im Allgemeinen schon ist, die extrem wachsende Komplexität zu beherrschen, zeigen die immer häufiger werdenden Rückruf-Aktionen. Tendenziell werden sich die Entwicklungszeiten deshalb eher verlängern als verkürzen. Aufwändig entwickelte Produkte brauchen länger, um sich zu rentieren. Kostenverursachende Release-Wechsel ohne großen Nutzen sind dabei unerwünscht. Gleichzeitig muss aber der zuverlässige Betrieb gewährleistet sein.

Bei einem alten Mainframe-Programm lässt sich die notwendige Sicherheit vielleicht noch durch die Abschottung des Rechenzentrums erreichen. Vernetzte Anwendungen bleiben jedoch über den gesamten Lebenszyklus auf Sicherheits-Updates angewiesen. Das stellt auch für Open-Source-Software eine erhebliche Herausforderung dar.

Die Community hat bei ihren Aktivitäten zu oft die Erstellung neuer Software im Fokus. Das ist verständlich, weil es einfach mehr Spaß macht, Neues zu entwickeln. In der Praxis wird jedoch viel mehr Zeit damit verbracht, Code zu lesen, zu verstehen und zu ändern. Allzu oft wird Objektorientierung fälschlich mit leichter Wartbarkeit gleichgesetzt. Die „Clean Code"-Bewegung ist eine Reaktion auf die mittlerweile herangewachsenen Probleme. Unabhängig davon, wie man zu einzelnen ihrer Thesen steht, die Behauptung, dass Code ohne intensives Gegenarbeiten (= „Refactoring") verkommt, wird wohl niemand widerlegen können.

Ein ursprünglicher Vorteil von Java war, dass dem „Klasse durch Masse" von C++ durch ein „klein aber fein" entgegengetreten wurde. Wenn man in C++ ganz unterschiedliche syntaktische Strukturen hatte, um etwas auszudrücken, gab es in Java meist nur einen vernünftigen Weg. Diese Tatsache erleichterte das Erlernen der Sprache und das Lesen des Programmcodes. Die zwischenzeitlichen Erweiterungen haben diesen Vorsprung

schrumpfen lassen. Aus meiner Sicht ist es überfällig, bei der Integration neuer Features eine Pause einzulegen, den erreichten Stand kritisch zu analysieren und zu konsolidieren – also ein Refactoring zu starten.

Bei all den Weiterentwicklungen sind beispielsweise viele der System-Bibliotheken nur angepasst worden. Da findet sich noch manches To-do aus den Anfangstagen oder Code (auch vermeintliche Optimierungen aus Vor-Hotspot-Zeiten), den kein erfahrener Entwickler mehr so schreiben würde. Alternativ oder parallel zur Überarbeitung der System-Bibliotheken sollte über deren Neugestaltung nachgedacht werden, und zwar derart, dass die alternativen Neu-Implementierungen langfristig ihre Vorgänger ersetzen können. Je länger eine solche Konsolidierung hinausgezögert wird, desto größer und schwieriger wird sie. Die Altlasten, die teilweise dem Zeitdruck bei der Veröffentlichung anzulasten sind, verseuchen mangels Alternativen noch heute jede Neuentwicklung.

Ungelöst ist immer noch die Modularisierung. Das Projekt „Jigsaw" wird von einer Version zur nächsten verschoben. Dabei ist es zweifelhaft, ob dem enormen Aufwand ein entsprechender Nutzen gegenüberstehen wird. Wenn man sich daran erinnert, dass in technischen Übersichten fast immer eine zweidimensionale Struktur aus Funktionssäulen und Implementierungsschichten auftaucht, ist die Frage, ob Baumstrukturen für die Modularisierung angemessen sind, nicht völlig abwegig.

Für die Zukunft sieht sich die Java-Community verschiedenen Herausforderungen gegenüber. Ein großes Problem ist – wie in vielen anderen Bereichen auch –, dass langfristig erfolgversprechende Strategien auf kurze Sicht oft unattraktiv sind und gleichzeitig die hohe Volatilität der Entwicklung Risiken fast unkalkulierbar macht. Ich sehe die folgenden wichtigen Tendenzen, die sich gegenseitig nicht völlig ausschließen müssen:

- *Die Entwicklung verläuft weiter wie bisher*
  Das ist (wie beim Wetter) die wahrscheinlichste Variante. Java wird dabei immer komplexer, der Abstand zwischen den oben erwähnten Gruppen der Anwender wächst weiter. Über kurz oder lang wird eine andere, neue Programmiersprache Java den Rang ablaufen. Letzteres ist auch deshalb wahrscheinlich, weil nach zwanzig bis fünfundzwanzig Jahren die Welt für ein neues Heilsversprechen reif ist, selbst wenn insgeheim jeder weiß, dass es nur ein kleiner Schritt vorwärts sein wird.
- *Die Entwicklung spaltet sich auf*
  Etwa so, wie das in gewissen Maß beim Firefox zu sehen ist, in einen erneuerungsfreudigen und einen eher bestandswahrenden (LTS-)Zweig. In Ansätzen praktizieren das bereits einige Anbieter von eigenen Java-Implementierungen. Zusammen mit der vorherigen Tendenz kann das bedeuten, dass auf die Dauer nur der (eventuell rein kommerzielle) LTS-Zweig überlebt. Für die IT-Industrie muss das nicht schlecht sein, das zeigt ein Blick auf Sprachen wie Ada oder Cobol, für Entwickler ist das eher weniger cool.
- *Java erneuert sich*
  Das ist das wünschenswerte, aber mit Abstand unwahrscheinlichste Szenario. Auf der Basis einer umfassenden Entsorgung der Altlasten in Syntax und Bibliotheken erfolgt ein evolutionärer Übergang zu einem echten „Java2". Die Default-Implementierungen von Interface-Methoden in Java 8 sind ein ganz kleiner Schritt in diese Richtung. Die vielen interoperablen Implementierungen anderer Programmiersprachen auf der JVM beweisen, dass dieser Weg technisch realisierbar ist.

Ganz gleich, wie es kommt – es wird interessant bleiben. Und hoffentlich wird wenigstens ein Teil der vielen bisher gewonnenen Erfahrungen in die Weiterentwicklung einfließen.

Java lebt!

Fünf Jahre Java aktuell. Ich gratuliere recht herzlich und möchte meine Eindrücke aus den letzten fünf Jahren und meine Vision, wohin sich Java und die Community entwickeln könnten, mit den Glückwünschen verbinden.

„Java ist tot" waren meine ersten Gedanken zur Übernahme von Sun durch Oracle. Ich denke, vielen von Euch ging es ähnlich. Natürlich sah ich auch Chancen, aber im Grunde war dort nur ein schwarzes Loch von Ahnungslosigkeit, wie es mit Java weitergehen sollte. Und genau dort kam die Gemeinschaft ins Spiel. Sie versuchte einen Software-Riesen zu steuern und zu lenken. Stellte sich ihm in den Weg bei radikalen Einschnitten und unterstützte ihn bei guten Ideen und Weiterentwicklungen.

Natürlich konnte nicht jedes Projekt oder jede Idee gerettet werden. Auch Oracle hat nichts zu verschenken und will strategisch am Markt agieren. Aber schlussendlich hat die Gemeinschaft in meinen Augen gut funktioniert und ich möchte mich bei allen bedanken, die mitgewirkt haben, noch immer wirken und auch bald ein Teil dieser Gemeinschaft sind. Ich denke, mit dem aktuellen Release von Java hat Oracle die Weichen in Richtung Zukunft gestellt.

Java lebt! Alles Gute, Java aktuell! Ich freue mich, die nächsten fünf Jahre mit euch zu verbringen.

Jürgen Lampe
juergen.lampe@agons-solutions.de

Dr. Jürgen Lampe ist IT-Berater bei der A:gon Solutions GmbH in Frankfurt. Seit mehr als fünfzehn Jahren befasst er sich mit Design und Implementierung von Java-Anwendungen im Banken-Umfeld. In dieser Zeit hat er sich immer wieder intensiv mit Performance-Problemen beschäftigt. An Fachsprachen (DSL) und Werkzeugen für deren Implementierung ist er seit seiner Studienzeit interessiert.

http://ja.ijug.eu/15/1/6

Gunther Petzsch
gunther.petzsch@saxsys.de

Gunther Petzsch ist Sun-zertifizierter Java-Entwickler. Er lebt und arbeitet in Dresden. Dort studierte er auch erfolgreich Wirtschaftsinformatik an der Hochschule für Technik und Wirtschaft Dresden. Danach arbeitete er unter anderem für das Bundeskriminalamt in Wiesbaden. Aktuell ist Gunther Petzsch als Senior Consultant für die Saxonia Systems AG tätig.

http://ja.ijug.eu/15/1/7