Anekdoten aus 20 Jahren Java

aus Anlass des 20-jährigen Jubiläums des JavaSPEKTRUMs

Sonnenaufgang

Michael Stal

Es war einmal ein Informatiker, dessen Herz nur für die Eine schlug. Sie hatte herrlich geschwungene Formen, war äußerst gesprächig und reiste durch alle Herren Länder. Wie alle ihrer Art war sie von nicht ganz einfachem Charakter, weshalb nur geschickte Sprachenforscher ihre Worte in verständliche Sprache zu übersetzen vermochten. Ihre Verehrer waren indes bald so zahlreich und unterschiedlich, dass sich eine Gruppe unter ihnen aufmachte, ein Idealmodell zu erarbeiten.

Wie unschwer zu erraten, war der besagte Informatiker niemand anderes als der Autor dieser Zeilen. Die Angebetete hieß C++ und die Gruppe war das ANSI-ISO-C++-Standardisierungskomitee X3J16.

Damals galt Objektorientierung noch nicht als dominierendes Paradigma. UML befand sich gerade mal auf dem Weg zu den Startlöchern. Mit „Cloud-Computing" assoziierten Entwickler die von Grady Booch definierte Analyse- und Design-Modellierungssprache. Kann sich noch jemand an die Schablone erinnern, mit der sich die Wölkchen für Klassen und Objekte leichter zu Papier bringen ließen? Und unter den Programmiersprachen-Verstehern tobte ein Glaubenskrieg darüber, ob nun Smalltalk oder C++ die bessere Wahl sei, wobei das erstklassige Eiffel auf der Strecke blieb.

Doch dann kam das Licht. Ein Kollege berichtete mir irgendwann im Jahre 1995 von der neuen Programmiersprache Java aus dem Hause Sun, die C++ ähnlich, aber angeblich wesentlich leichter zu benutzen sein sollte. Und als Clou würden Programme auf einer virtuellen Maschine interpretiert. Das sei – seiner Meinung nach – die Zukunft der Objektorientierung. Da schüttelte ich aufgrund meiner Erfahrungen mit UCSD-Pascal nur müde lächelnd den Kopf und war zutiefst amüsiert über die „naiven" Wunschträume meines Kollegen. Ich leiste hiermit Abbitte, Hans!

Nur knapp ein Jahr später sollte ich zusammen mit Michael Johann das Chefredaktions-Duo von JavaSPEKTRUM bilden, und meine Programmierbeispiele im Buch „Pattern-Oriented Software Architecture – A System of Patterns" nach Java konvertieren.

Was war geschehen? Begonnen hatte mein Umdenkungsprozess mit dem Vorsatz „Ich kann ja mal unverbindlich Java lernen, um es besser in die Mangel zu nehmen". Es ist schließlich immer besser, den „Feind" genau zu kennen. Nach einem verregneten Wochenende hatte ich die Sprache, die Werkzeuge und die Klassenbibliothek intus. Allerdings sollte ich hinzufügen, dass sich die gesamte Java-Plattform anfangs noch mit einer übersichtlichen Größe begnügte. Das berühmte Buch von David Flanagan „Java in a Nutshell" eignete sich mangels Umfang allenfalls zum Eliminieren von Mücken, während man heutzutage damit einen Tyrannosaurus Rex erschlagen könnte. Noch dazu geriet ein befreundeter Kollege Uwe, ebenfalls Mitglied des X3J16-Komitees und Chefdesigner einer exzellenten kommerziellen C++-Klassenbibliothek, in Java-Abhängigkeit. Nach einigen Monaten begannen Kollegen nur noch lautlos an seinem Büro vorbeizuschleichen, in der Hoffnung, unentdeckt zu bleiben. Wer das aus Leichtsinn oder Unkenntnis nicht tat, wurde mit großer Wahrscheinlichkeit von Uwe mit den Worten „Darf ich Dir einmal kurz etwas Interessantes über Java zeigen" abgefangen. Diese „leichtsinnigen" Kollegen verließen erst nach Stunden Uwes Büro mit glühenden Ohren und entfernten sich dann so schnell sie konnten. Kurzum, der Java-Enthusiasmus war ansteckend und führte schon bald zu „Beschaffungskriminalität".

Das Epizentrum des Java-Bebens lag im kalifornischen Mountain View und sollte die Welt der Softwareentwicklung für immer umkrempeln. Alles ging Schlag auf Schlag. Erst zaghaft, dann immer öfter führten Unternehmen Java in ihre Projekte ein. Mit Erscheinen der Enterprise Edition drang Java äußerst erfolgreich in neues Terrain vor, und streifte damit sein „Applet"-Image für immer ab, das es ohnehin zu Unrecht trug. Denke ich an die erste EJB-Version, kommen mir heute noch die Tränen.

Bill Gates lobte Java überschwänglich, nur um es kurz darauf mit seiner eigenen Java-Variante zu versuchen. Diesem Weg war aber nur ein kurzfristiger Erfolg beschieden, primär wegen eines verlorenen Rechtsstreits mit der Java-Mutter Sun Microsystems. Darüber hinaus ignorierten die meisten Java-Entwickler den vergifteten Apfel ohnehin.

Um die Jahrtausendwende erschien Microsofts .NET-Framework. Dieses besaß genau jene Eigenschaften, die man kurz zuvor noch als entscheidende Nachteile von Java kritisiert hatte. Nach den unvermeidlichen „.NET versus Java"-Religionskriegen herrscht heutzutage ein friedliches Nebeneinander, übrigens mit ein Grund dafür, dass wir das IntegrationsSPEKTRUM als – Vorsicht, Wortspiel – „integralen Bestandteil" von JavaSPEKTRUM aufgenommen haben. Darüber hinaus fördert Konkurrenz bekanntlich die Innovation.

Und sie lebten glücklich und zufrieden bis an ihr Lebensende. Wirklich? Um die Zukunft von Java ist mir tatsächlich nicht bange. Zum einen war die .NET CLR (Common Language Runtime) als Laufzeit-Plattform für verschiedene Programmiersprachen konzipiert worden, ganz im Gegensatz zur Java VM. Trotzdem benutzen heute die meisten Programmiersprachen bevorzugt die JVM als Heimatbasis. Java als Plattform käme also notfalls auch ohne Java als Sprache aus.

Zum anderen hat die Java-Plattform für Technologien wie zum Beispiel Internet der Dinge, Cloud-Computing, Mobilgeräte oder Big Data eine herausragende Bedeutung. Dafür sorgt auch eine riesige Community von Informatikern, die bereits im Studium Java mit der Milch ihrer Alma Mater aufsaugen. Java ist heute jedenfalls in einem reifen Alter und hat dank vorzüglicher Erziehung das Gröbste hinter sich. Wie heißt es in einer Schokoladenwerbung: „Merci, dass es Dich gibt!"

In diesem Sinne: ein Toast auf das prächtige Geburtstagskind Java und seinen stolzen Papa James!

Java ist cool

Holger Sirtl, Microsoft

Die Mitwirkung im Fachbeirat des JavaSPEKTRUM ist für einen Microsoft-Mitarbeiter natürlich etwas Besonderes. Deckt die Tonalität der Gespräche mit Mitgliedern der Java-Community doch ein weites Spektrum ab – von enthusiastisch freundlich bis ablehnend kritisch. Langweilig sind solche Kontakte selten. Doch der Reihe nach.

Mit Fug und Recht kann ich von mir sagen, dass ich eigentlich den größeren Teil meines studentischen und beruflichen Wirkens im Java-Umfeld unterwegs war. Dies war jedoch zu Beginn zunächst nicht abzusehen. Durfte ich mich doch im ersten Teil meines Informatik-Studiums mit Modula-2 auseinandersetzen. Für jemanden, der sein Studium nicht zuletzt auch als Vorbereitung auf die berufliche Praxis sah, eher ernüchternd. Schon damals war an Stellenausschreibungen im IT-Umfeld abzusehen, wo die Reise hinging: Java! – das war's: modern, objektorientiert, plattformübergreifend, sprich: cool. Im Hauptstudium war es dann endlich soweit: Java stand auf dem Lehrplan. Nebenbei bemerkt war dies in den mir nachfolgenden Jahrgängen dann auch schon im Grundstudium der Fall. Angetrieben also von der damals viel zitierten Vision, in Zukunft auch Dinge wie Kaffee-, Wasch- und sonstige Maschinen in Java programmieren zu können, produzierte ich jede Menge Java-Code.

Die Programmierung in Java setzte sich dann, einzig unterbrochen von einem kurzen Intermezzo in C, im Beruf fort. Tatsächlich war ich immer wieder von der Universalität von Java beeindruckt, mit der diese Programmiersprache eine Konstante in höchst unterschiedlichen Softwareprojekten war: von Implementierungen im Banken- und Versicherungssektor über den behördlichen Bereich bis hin zu Projekten der Energieversorger-Branche und dem öffentlichen Nahverkehr – stets spielte Java eine zentrale Rolle. Und ich empfand es immer als große Stärke von Java, im Mittelpunkt einer lebendigen Community zu sein. Einer Community, die stets Innovationen in Form von Frameworks, Server-Implementierungen usw. hervorbringt und damit die gesamte Java-Plattform weiterbringt und modern hält.

Bei Microsoft nehme ich dies weiterhin wahr. Die Java-Plattform ist sehr gut aufgestellt, um im Rahmen von Trendthemen wie Internet-of-Things, agile Methoden, automatisiertes Deployment, DevOps eine gewichtige Rolle zu spielen. Dass Java von Anfang an auf Portabilität ausgelegt war, hatte zur Konsequenz, dass Java-Entwickler mit der Paketierung von Anwendungssystemen beispielsweise sehr gut zurechtkommen und für neue Ansätze wie containerbasiertes Deployment, Softwareverteilung in verteilte Systemlandschaften usw. sehr offen sind. In Zeiten verkürzter Innovations- und Softwarelebenszyklen, neuer Ausführungsplattformen und Formfaktoren für Geräte ist dies ein wertvolles Gut. Es freut mich, dass hier auch vonseiten Microsoft eine Öffnung stattgefunden hat und Java beispielsweise auf Microsoft Azure wunderbar flexibel eingesetzt werden kann. In vielen Gesprächen mit Partnern und Kunden bestätigt sich immer wieder, dass sich Java-Entwickler dank des portablen JREs auch dort wie auch auf vielen anderen Plattformen zuhause fühlen.

Ich beglückwünsche die Programmiersprache Java zu ihrem 20-jährigen Geburtstag und zur Tatsache, dass sie jung und modern geblieben ist. Ich wünsche ihr, dass dies so bleibt und im Wettbewerb mit Programmiersprachen wie C# auch weiterhin eine fruchtbare, konstruktive Co-Existenz bestehen bleibt, von der alle profitieren.

Die Medusa zeigt ihr Gesicht

Wolfgang Rohde, Siemens PLM

Wir schreiben das Jahr 2001 und Java hat seinen Siegeszug um die Welt angetreten. Deshalb mussten auch alte C++-Dinosaurier Schulungen zwecks Horizonterweiterung besuchen.
Ich entschied mich für einen Drei-Tages-Crash-Kurs „Java fuer C/C++ Developer".

▼ **Tag 1: Java-Grundlagen**
Eine Programmiersprache ohne Pointer. Abgesehen davon, dass ich meine Zeiger und die elegante Schreibweise (->) vermisse, hat Java einige sympathische Züge. Beispielsweise gefallen mir die vielfältigen Bibliotheken, die als Standard im Sprachumfang integriert sind und beständig ausgeweitet werden.

▼ **Tag 2: Die Medusa zeigt ihr Gesicht**
Soweit, keine Probleme bei der Umstellung auf Java. Auch wenn ich manchmal noch mit den intrinsischen Referenz-Typen hadere. Ich beginne mit Arrays zu arbeiten und habe mich schon bequem eingerichtet; mich um nichts kümmern zu müssen – okay fast nichts. Und dann passiert es: Null-Pointer-Exception. Wie kann das sein? Es gibt doch keine Pointer in Java. Ich bin geschockt.

▼ **Tag 3: Super Café mit super Ausblick auf die Berge gefunden**
Ich kontempliere über einige Cappuccino, über Null-Pointer und dem Versprechen „Alles wird gut und um alles wird sich gekümmert".

Fazit: Java ist zweifelsfrei eine interessante Sprache, mit einem hohen Anspruch (fast) alle Problem der Softwareentwicklung zu lösen. Aber hat Java die Welt der Softwareentwicklung wirklich einfacher gemacht? Nicht wirklich! Im Laufe der letzten Jahre wurde offensichtlich, dass Java um mehrere komplexe Sprachkonzepte erweitert wurde, um akute Probleme in der Softwareentwicklung zu lösen. Java hat die Komplexität weiter in Richtung Fachlichkeit verschoben, allerdings zum Preis der Ausführungsgeschwindigkeit.

Glücklicherweise konnte der Verlust durch Innovationen im Hardwarebereich mehr als ausgeglichen werden :)

Java-Spracharchitektur hilft den Profis

Monika Kaiser, SAP AG

Ich hatte meine erste Begegnung mit Java im Jahr 1999. Ich wechselte innerhalb der SAP in ein Team, das auf dem Gebiet der Java-Entwicklung Pionierarbeit in unserer Firma leistete.

Mein erstes Entwicklungstool, soweit ich mich erinnern kann, war Notepad. Das heißt, Annehmlichkeiten wie Code-Completion oder Navigation zu Methodendefinition oder Auflisten der Klassenhierarchie waren nicht vorhanden. Ich verbrachte sehr viel Zeit damit, bei jeder Java-Exception mühselig die Zeilen abzuzählen, um herauszufinden, an welcher Stelle im Coding diese Exception geworfen wurde. Softwareentwicklung glich in etwa der Spurensicherung in einem Kriminalfall mit einer Vergrößerungslupe als einzigem Arbeitsinstrument.

Dann kam der erste Fortschritt – ich hatte mir eine UltraEdit-Lizenz besorgt und war richtig stolz auf die Zeilennummerierung und das Syntax-Highlighting für meine .java-Dateien. Und irgendwann später hatte ich die erste Eclipse-Installation auf meinem Desktop.

Trotz aller Unzulänglichkeiten, die ich wegen fehlender Tools erlebte, war ich von Java fasziniert – nicht, weil die Programme auf Windows und Linux laufen konnten (ich hatte bis jetzt immer in Windows entwickelt), sondern, weil die Sprache sich mir so klar und wohlstrukturiert darstellte.

Jede Klasse ist einem wohldefinierten Paket zugeordnet und implementiert in der Regel ein Interface als Kommunikationsschnittstelle nach außen. Dank der Javadoc-Annotationen dienen diese APIs gleichzeitig als Dokumentation für Nutzer der Klassen, der Pakete und der Bibliotheken. Das heißt, trotz der fehlenden Tools und dank der Java-Spracharchitektur konnte unser Team sehr effizient arbeiten. Sobald die APIs definiert und durch einen Review-Prozess finalisiert waren, konnte das Team wie bei einer Multi-Core-Verarbeitung parallelisiert und weitgehend konfliktfrei die Entwicklung durchführen.

In meinen Augen ist dies auch der Grund, warum Java insbesondere für professionelle, über mehrere Standorte und Zeitzonen verteilte Softwareentwicklung so attraktiv wurde und bleibt.

Once a time in Java-Land

Thomas Ronzon, w3logistics AG

Wie ich Java lernte

Ich hatte 1998 schon meine ersten Projekterfahrungen mit der Programmierung in C für verschiedene Unix-Systeme gemacht. In der Firma, in der ich damals war, gab es „Consultants“, also Experten für ein Thema, welche vor Ort an einen Kunden verkauft wurden. In meiner Freizeit hatte ich schon erste Erfahrungen mit HTML gesammelt (damals etwas ganz Tolles!), und so war ich sehr froh, dass mein Chef jemanden als „HTML-Consultant“ für einen Kunden (ebenfalls Softwarehaus, aber ohne Java-Erfahrung) suchte.

Froher Erwartung fuhr ich mit einem anderen Kollegen zu diesem Kunden. Leider stellte sich bereits nach ein paar Stunden heraus, dass der Kunde keinen HTML-Experten, sondern Java-Enterprise-Entwickler suchte. Er wollte eine Webapplikation bauen und dabei „Alles richtig machen“. Deshalb wollte er auch, dass wir das gesamte System objektorientiert mit Rational Rose [RR] und dem Rational Unified Prozess [RUP] durchziehen. Mein Problem war nur – ich kannte weder Objektorientierung – noch kannte ich Java! Und was ist eine JVM?

Da wir aber den Kunden nicht verlieren wollten, bekam ich von meiner Firma ein Notebook und das Buch „Java in a Nutshell“, um abends nach der Arbeit im Hotelzimmer Java zu lernen.

Tja, was soll ich sagen – der Kunde hat es bis zum Schluss nicht gemerkt. Man muss eben nur einen Schritt weiter sein! Stress war es trotzdem! .-)

Wie sich die Kunden verhielten

Da es sich bei unserem Kunden ebenfalls um ein Softwarehaus handelte, hatten wir mit dem Endkunden relativ wenig zu tun. Eine Sache bereitete uns aber große Probleme. So sollten wir Silverstream [SILVERSTREAM] als Applikationsserver einsetzen, welcher auch das GUI übernahm. Zu dieser Zeit war der Internet Explorer 3 der Browser, welcher beim Kunden eingesetzt wurde. Dieser hatte jedoch so viele Beschränkungen und Fehler, dass wir Entwickler den Internet Explorer 4 benutzen wollten.

Dies führte beim Kunden zu einer solchen Aufregung, dass der Betriebsrat eingeschaltet werden musste! Die Begründung war einfach: „Man kann einen Sachbearbeiter nicht in so kurzen Abständen an neue Software (neue IE-Version) gewöhnen!“

Nachdem sich die Wellen gelegt hatten und wir versprochen hatten, den Browser nicht mehr so schnell zu wechseln, durften wir dann aber doch den IE 4 benutzen.

Wie es weiterging

Ein paar Projekte später (so gegen 2002) suchte ein Kunde (ebenfalls ein Softwarehaus) erfahrene Webentwickler zur Entwicklung einer Oberfläche für ihre Lagerverwaltung. Da aber das gesamte Restteam nur aus PL/SQL- und maximal C-Entwicklern bestand, wurden mein Kollege und ich zusammen mit zwei neuen Mitarbeitern des Kunden eingeteilt, die Benutzerdialoge zu erstellen.

Verwendet werden sollte eine Java-Tag-Library für JSPs, welche der Kunde bei einer ungarischen Firma, welche er dann auch gleich gekauft hatte, hatte entwickeln lassen. Neben der Tag-Library lieferte die Firma ein schönes Handbuch mit Beispielen, sodass wir innerhalb kürzester Zeit die ersten Dialoge erstellen konnten.

Als diese jedoch etwas anspruchsvoller wurden, bekamen wir Fehler beziehungsweise konnten selbst die Beispiele im Handbuch nicht nachvollziehen. Der Grund war einfach – das Handbuch beschrieb Funktionen, welche gar nicht implementiert waren! Den Beweis brachte schon ein Blick in die TLD-Datei [TLD] – selbst hier waren viele Dinge nicht einmal definiert.

Da aber keiner im restlichen Projektteam wusste, was zu tun war, kam der Projektleiter auf die Idee, dass wir doch einfach die restlichen Teile selbst in JavaScript schreiben und in die JSP einbetten sollten (den Quellcode der Taglib hatten wir nämlich nicht).

Ich erinnere mich an Masken, welche 5000! Zeilen zusätzlichen JavaScript-Code hatten ...

Links

[RR] http://www-03.ibm.com/software/products/de/ratirosefami
[RUP] http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Vorgehensmodell/Rational-Unified-Process-(RUP)/index.html
[SILVERSTREAM] http://www.serverwatch.com/server-reviews/article.php/1433211/SilverStream-An-enterprisestrength-application-server-and-development-environment.htm
[TLD] https://docs.oracle.com/javaee/5/tutorial/doc/bnamu.html

Nur gute Standards bleiben bestehen

Erik Dörnenburg, ThoughtWorks GmbH

Es ist Mitte Mai 2000. Zusammen mit einigen Kollegen sitze ich in einem übervollen Vortragsraum bei Apples World Wide Developer Conference (WWDC) und erwarte gespannt Session 407, einen Vortrag von Rory Lydon. Wir sind hier, weil wir uns beruflich mit Apples Applikationsserver WebObjects beschäftigen, einem der ersten Applikationsserver überhaupt. Ursprünglich noch von NeXT für und mit Objective-C entwickelt, wanderte er in der Übernahme Ende 1996 mit zu Apple und wurde Java-kompatibel gemacht.

Rory wird uns erzählen, was sich in der Java-Welt so tut. Sein Vortrag trägt den Titel „WebObjects: EJB – Making the Best of a Bad Thing“. Nach einer Einführung geht es zu der Beschreibung wesentlicher Features der EJB-Spezifikation, und viele der Anwesenden trauen ihren Ohren nicht. Rory zitiert aus der Spezifikation, vergleicht mit den eleganten, über Jahre gereiften Lösungen in WebObjects. Die Zuhörer machen sich besorgte Gedanken, wie ihre Anwendungen mit dieser EJB-Technologie umgesetzt werden können. Die Spezifikation selbst steht ebenfalls im Visier und Rory weist gnadenlos auf Lücken und Schwächen hin. Nach weiteren zitierten Passagen aus der Spezifikation entspannt sich die Stimmung im Saal und spätestens bei Bean- und Container-Managed Persistence gibt es dann auch erste Lacher. „Das kann doch so niemals funktionieren“, denken sich viele der Anwesenden.

Wenige Jahre später wird klar, dass die Anwesenden recht behalten sollten. Die EJB-Spezifikation stellt sich als eine der schwächeren in der Java-Welt heraus, und nach anfänglicher Akzeptanz im Markt wechseln viele Projektteams zu alternativen Technologien, wie Spring, Hibernate und SOAP.

Was diese Geschichte für mich allerdings am deutlichsten zeigt, ist die Stärke der Java-Community. Schlüsseltechnologien sind nicht heilig. Standards sind nicht alles. Praktiker finden praktikable Lösungen und diese erreichen als Open-Source-Software weite Verbreitung. Gute Standards bestehen, für weniger gute wird Ersatz gefunden. So schreiben wir auch heute moderne Java-Anwendungen, die Daten in relationalen Datenbanken speichern. Von WebObjects spricht eigentlich niemand mehr.

Kaffee mit Pizza

Dr. Thomas Wieland, Hochschule Coburg

Das erste Mal kam ich 1997 mit Java in Berührung. Ich hatte gerade meine Dissertation abgeschlossen, für die ich alle Software in C++ geschrieben hatte. Nebenbei hatte ich für ein Industrieunternehmen eine Anzeige- und Steuerungssoftware in Turbo Pascal entwickelt, wo auch eine Menge Code entstanden war. Ich mochte Pascal, weil es klare Programmstrukturen erlaubt und vor allem unheimlich schnell kompilierte. Mein damaliger 486er PC mit atemberaubenden 100 MHz Taktgeschwindigkeit brauchte nämlich für C++-Programme schon eine ganze Weile. Hatte man mal einen Fehler gemacht und das Programm stürzte ab, blieb in der Regel das Windows 95 gleich mit hängen. Das war einer der Gründe, weshalb ich auf IBM OS/2 umgestiegen war – ein damals vielversprechendes PC-Betriebssystem, da es vor allem deutlich stabiler als Windows war. Mit dem dortigen Borland C++ konnte man auch schon recht komfortabel programmieren.

Im Jahr 1997 kam ich dann zum Deutschen Zentrum für Luft- und Raumfahrt, wo wir meist auf Unix-Workstations arbeiteten. Bei einer Abteilungsbesprechung wurde eines Tages eine neue Programmiersprache namens „Java“ vorgestellt. Es sollte auf allen Plattformen verfügbar sein, ja noch besser: Java-Programme sollten unverändert auf allen Plattformen laufen! Die Syntax erinnerte stark an C++, wobei man die unschönen Dinge wie Speicherverwaltung und Mehrfachvererbung weggelassen hatte. Es klang alles sehr vielversprechend, sodass wir uns beim nächsten kleineren Entwicklungsprojekt mal auf Java einließen und begannen, damit Software zu schreiben.

Die ersten Erfahrungen waren ermutigend: Java-Code ließ sich schnell erstellen, es war elegant, unterstützte alle modernen Aspekte der Objektorientierung und sorgte so für einen gut lesbaren und klar gegliederten Code. Nur fehlte mir ein Feature, das ich an C++ besonders geschätzt hatte: die Templates. Da kam ich eines Tages zu einem Kollegen ins Büro, der mir begeistert von einer Java-Erweiterung namens „Pizza“ erzählte. Cooler Name, dachte ich. Wenn man schon das Klischee rauskehrte, dass Programmierer nur von starkem Kaffee (im Englischen eben auch „Java“ genannt, den ich damals überhaupt nicht mochte) und kalter Pizza lebten (iiihh!), war der Name gar nicht mal schlecht gewählt. Später erfuhr ich dann, dass die Entwickler ursprünglich ihren Pizza-Compiler völlig unabhängig von Java entwickelt und benannt hatten und diese Häufung von Stereotypen eigentlich nur Zufall war.

Pizza war ein Präcompiler. Die Programme mit der Endung „.pizza“ wurden durch den Pizza-Compiler in „.java“-Dateien übersetzt, die wiederum vom normalen Java-Compiler verarbeitet wurden. Pizza erlaubte Zeiger auf Funktionen, algebraische Datentypen und vor allem Templates – heute als „Generics“ bekannt. Damals diskutierte die Community noch heftig darüber, warum Java keine Templates braucht und man alles durch Polymorphismus über die Hierarchie eines gemeinsamen Objektobjekts „Object“ auch hinbekommen könne.

Ein Beispiel für eine solche Templateklasse ist (aus dem Pizza-Tutorial):

```
class StoreSomething<A> {
  A something;
  StoreSomething(A something) {
    this.something = something;
  }
  void set(A something) {
    this.something = something;
  }
  A get() {
    return something;
  }
}
```

Also eine Syntax, die man auch aus C++ kennt, aber mit strengerer Typprüfung. In gewohnter Weise wird erst bei der Instanziierung einer konkreten Klasse die entsprechende Variante kompiliert:

```
StoreSomething<String> a = new StoreSomething("I'm a string!");
StoreSomething<int> b = new StoreSomething(17+4);

b.set(9);

int i = b.get();
String s = a.get();
```

Unser Fazit nach einigen Tests war dann, dass Pizza zwar tolle Features und Möglichkeiten bot, es für ein längerfristig angelegtes Projekt wie unseres aber damals zu unsicher war, wie und in welcher Form Pizza auf Dauer bestehen würde. Später wurde Pizza dann auf SourceForge gehostet (http://pizzacompiler.sourceforge.net/), Ende 2001 aber eingestellt. Erst 2004 hielten die Generics dann in ähnlicher Form (aber immer noch etwas anders) in Java 1.5 Einzug. Das Pizza-Team kam dagegen nicht weiter als bis Java 1.4, eine Übernahme in den Java-Hauptstrang fand nie statt. Schade eigentlich, dabei passt Pizza doch gut zu Kaffee, oder?

Oh Gott, ein in Java implementiertes RDBMS

Klaus Rohe, Flughafen München GmbH

Als Java 1995 offiziell das Licht der Welt erblickte, war ich als Consultant bei der Informix Software GmbH beschäftigt. Das Aufgabenspektrum reichte von der Beratung von Kunden, wie man performante Datenbankanwendungen entwickelt, über Datenbank-Performance-Tuning bis zur Portierung der Informix-Datenbankserver auf verschiedene Unix-Derivate. Die für mich wichtigen Programmiersprachen waren C, Embedded SQL for C (ESQL/C) und Informix 4GL. Die Informix-Datenbankserver selbst waren in C und besonders laufzeitkritische Teile in Assembler implementiert.

Ich habe natürlich in der Zeit von 1995 bis 1997 mit Java sporadisch experimentiert, auch mit Visual J++ von Microsoft. J++ war eine proprietäre Java-Implementierung für Windows, die das Motto von Java „Write once run anywhere“ total unterlief. Dies führte zu einem langen Rechtsstreit zwischen Sun und Microsoft, der möglicherweise einige Rechtsanwälte fast so reich gemacht hat wie Bill Gates. Die Performance der Java-Programme, die ich geschrieben habe, war nicht berauschend und der Schwerpunkt schien mir auf der Entwicklung von Applets zu liegen. Das war nicht mein Thema, und so habe ich Java erst einmal ad acta gelegt.

1999 gab es für mich dann einen Kulturschock, als Informix die Firma Cloudscape aus Oakland (Kalifornien) kaufte, deren gleichnamiges Produkt ein komplett in Java implementiertes relationales Datenbankmanagementsystem (RDBMS) war. Zu dieser Zeit waren meine Schwerpunktthemen Datenbank-Performance-Tuning und Hochverfügbarkeit. Ein in Java programmiertes RDBMS war für mich daher unter diesen Gesichtspunkten ein Sakrileg.

Cloudscape hat seitdem einiges durchgemacht. IBM kam 2001 durch die Übernahme von Informix in den Besitz von Cloudscape und übergab es 2004 unter dem Namen „Derby“ an die Apache Software Foundation. Seit 2006 ist „Derby“ als Java DB im JDK enthalten, das mittlerweile zusammen mit Sun zu Oracle migriert ist. Es sind also nicht nur ehemalige Mitarbeiter von Informix bei Oracle gelandet, sondern auch ein Teil des Quellcodes.

Den oben genannten Kulturschock habe ich übrigens schon lange verwunden.