KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X
12/2010

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

Über die allmähliche Verfertigung der Programme beim Tippen

> Wenn du etwas wissen willst und es durch Meditation nicht finden kannst, so rate ich dir, mein lieber, sinnreicher Freund, mit dem nächsten Bekannten, der dir aufstößt, darüber zu sprechen. Es braucht nicht eben ein scharfdenkender Kopf zu sein, auch meine ich es nicht so, als ob du ihn darum befragen solltest: nein! Vielmehr sollst du es ihm selber allererst erzählen.

So wusste schon Heinrich von Kleist in seiner geistreichen und sprachlich so wunderbaren Abhandlung *Über die allmähliche Verfertigung der Gedanken beim Reden* [1] zu empfehlen. Sicher ist es jedem von uns schon einmal so gegangen, über einem Problem verzweifelnd brütend endlich jemanden zu Rate zu ziehen, um beim Leid klagenden Sprechen festzustellen, dass sich das gerade eben noch Unverständliche langsam in Luft auflöst, um der gesuchten Lösung Platz zu machen.

Bei uns war es oft die Sekretärin, die erkannte, dass etwas nicht wunschgemäß fortschritt; und wohlwissend, dass sie von der Programmierung wahrlich keinen blassen Schimmer hat, freundlich fordernd fragte, wo denn genau das Problem sei. Und so sprudelte es aus dem Gefragten hervor, gleichfalls des Umstandes bewusst, dass das Gegenüber nichts von dem verstehen würde, was da kommt; wenngleich dieses, mit Empathie gesegnet, im rechten Moment mit „Aha!“ oder „Wirklich?“ oder vergleichbaren Ausrufen aufwartend, jenem die nötige Spannung brachte – ohne ihn unter Druck zu setzen –, die nötig war, um den Gedankenknoten entwirren zu können. So lernte man sich in verzweifelten Lagen an die gute Seele zu wenden und tat gut daran, diese nicht gegen einen Entwickler einzutauschen, stellte dieser doch gelegentlich Fragen, die einen im Redefluss, war man endlich in Fahrt gekommen, bremsten und so dem eigentlichen Ziel, die Lösung zu finden, eher kontraproduktiv im Wege standen.

Dieses Phänomen macht sich letztendlich jeder Entwickler gelegentlich zu Nutze, wenn er mit der Analyse eines Problems nicht weiter kommt; nur anstatt mit einem Menschen zu reden, bedient er sich dabei der Maschine, dem Rechner, mit der er auf eigentümliche Weise Konversation betreibt. Die Maschine, gleichermaßen geduldig, erlaubt das Schreiben von Programmen, die mit geeigneten Ausgaben gespickt, dem ratlosen Programmierer Schritt für Schritt dabei helfen, das Problem stückchenweise zu verstehen und sich der Lösung langsam anzunähern. Dabei bedient sie sich gelegentlich ganz dezent einer *NullPointerException* oder einer vergleichbaren Ausnahme, bis schließlich der Programmierer begriffen hat, was er eigentlich machen muss und wie er es am besten durchsetzen kann.

Denken Sie aber bitte nicht, ich glaubte mir den Schluss erlauben zu dürfen, die Maschine könne den Menschen ersetzen. Ganz im Gegenteil: Der Rechner soll zwar so viel machen, wie er kann, aber nur, damit der Mensch mehr Zeit für sich und seine Mitmenschen haben kann. Das wird nichts daran ändern, dass der Mensch die Maschinen programmieren muss, aber es wird – hoffentlich – dabei helfen, zum Beispiel die Weihnachtszeit in Ruhe und ohne Arbeit und Hektik verbringen zu können.

So wünsche ich Ihnen in diesem Sinne ein gesegnetes Weihnachtsfest und ein gesundes neues Jahr,

Ihr Michael Wiedeking

Referenzen

[1] Kleist, Heinrich von
Über die allmähliche Verfertigung der Gedanken beim Reden
http://gutenberg.spiegel.de/?id=5&xid=1467&kapitel=1#gb_found

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 Alexandra Specht via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-14 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 Alexandra Specht via E-Mail unter alexandra.specht@bookware.de oder telefonisch unter 09131/8903-14.

Copyright

Impressum

KaffeeKlatsch Jahrgang 3, Nummer 12, Dezember 2010
ISSN 1865-682X
Bookware – eine Initiative der MATHEMA Software GmbH

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: Alexandra Specht
Grafik: Nicole Delong-Buchanan

Inhalt

Artikel

Kolumnen

Schweizer Java

von Rüdiger Keller

In den letzten Jahren sind viele Sprachen für die Java Virtual Machine entstanden, von denen einige mehr, andere weniger populär geworden sind. Eine Sprache, die momentan viel Aufmerksamkeit auf sich zieht, ist Scala. Sie basiert auf den Ideen, ein besseres Java zu sein, die Konzepte objektorientierter und funktionaler Programmierung in sich zu vereinen und dabei an statischer Typisierung festzuhalten. Dies ist der erste Artikel der Serie und gibt einen Überblick zur Sprache.

Des Programmierers kleine Vergnügen

von Michael Wiedeking

Haben Sie das auch schon erlebt, dass ein Lehrer oder Professor ein Thema mit den Worten *der Rest sei evident* oder *trivial* beendet hat. Mir ist das des öfteren passiert. Und wahrscheinlich habe ich mich deswegen beim letzten Vergnügen dafür gerächt und es ungerechter Weise an Ihnen ausgelassen. Habe ich doch zum Vertauschen der Bits einen Verweis angegeben, der sich im Nachhinein als suboptimal herausstellte.

Deutsch für Informatiker

von Alexandra Specht

Ja, das Fest der Liebe naht; was liegt da näher, als etwas über die Sprache der Liebe zu schreiben! Und natürlich liegt es in der Natur dieser Kolumne, dass es sich auch um Informatiker drehen wird.

Schweizer Java

Eine Einführung in Scala, Teil 1

von Rüdiger Keller

In den letzten Jahren sind viele Sprachen für die Java Virtual Machine entstanden, von denen einige mehr, andere weniger populär geworden sind. Eine Sprache, die momentan viel Aufmerksamkeit auf sich zieht, ist Scala. Sie basiert auf den Ideen, ein besseres Java zu sein, die Konzepte objektorientierter und funktionaler Programmierung in sich zu vereinen und dabei an statischer Typisierung festzuhalten. Dies ist der erste Artikel der Serie und gibt einen Überblick zur Sprache.

Hintergrund

Scala wird unter der Regie von Martin Odersky am Ecole Polytechnique Fédérale de Lausanne in der Schweiz entwickelt. Die ersten Designentwürfe wurden bereits 2001 gemacht und Version 1.0 wurde 2003 veröffentlicht. Mit Version 2.0 wurden 2006 viele tiefgreifende Änderungen vorgenommen. Die Sprache entwickelt sich stetig weiter und ist mittlerweile bei Version 2.8.1 angekommen.

Scala steht für *Scalable Language*. Diese Namensgebung drückt aus, dass die Schöpfer der Sprache sich das Ziel gesetzt haben, sie zum Schreiben von Programmen beliebiger Größe und Komplexität geeignet zu machen.

Scala läuft auch unter .Net, wenngleich die Unterstützung weit unter der von Java herhinkt.

Grundideen

Die Grundideen der Sprache sind ein besseres Java zu sein und sowohl objektorientierte als auch funktionale Paradigmen zu unterstützen. Durch die Nähe zu Java soll der Umstieg erleichtert werden.

Ein Grundsatz der Sprache ist das Streben nach Kürze und Prägnanz und hat zwei Ausprägungen. Einerseits soll der in ihr geschriebene Code prägnant und ohne überflüssige Syntax-Elemente sein, andererseits soll die Sprache selbst, beziehungsweise ihre Spezifikation, möglichst kurz sein. Dies bewirkt, dass es nur sehr wenige Ausnahmen und Sonderfälle gibt. Im Gegensatz zu Java sind die Sprachkonstrukte nicht auf spezielle Anwendungsfälle zugeschnitten, sondern möglichst allgemein und zueinander orthogonal gehalten.[1] Dadurch stehen dem Entwickler in Scala sehr mächtige Ausdrucksmöglichkeiten zur Verfügung und viele Features lassen sich als Bibliotheksfunktionen realisieren, die normalerweise als Sprachkonstrukte bekannt sind.

Hier noch exemplarisch einige Verallgemeinerungen und Vereinfachungen gegenüber Java. Es gibt keine primitiven Typen, sondern nur Objekte. Es gibt keine Operatoren, sondern stattdessen Methoden in Infixnotation. Die *import*-Anweisung kann praktisch überall stehen und auch Methoden und Felder aus unveränderlichen Werten statt nur aus Typen importieren. Und der Vergleich mit „==“ wird mit Hilfe der *equals*-Methode realisiert – möchte man tatsächlich Referenzen vergleichen, kann man dies mit *eq* und *ne* tun.

Konkrete Beispiele

Um einen Vorgeschmack zu vermitteln, wie Scala-Code im Vergleich zu Java aussieht und welche Vorteile es bringen kann, folgen nun einige Beispiele.

Eine *Value*-Klasse, die einen Angestellten repräsentiert, komplett mit Zugriffsmethoden[2] und *equals*, *hash*-Code und *toString*:

```
case class Angesteller(
  name: String,
  abteilung: String,
  gehalt: BigDecimal
)
```

Eine Methode, welche eine Sequenz[3] von Angestellten nach ihrem Gehalt sortiert und die zehn Bestverdiener zurückliefert:

```
def topTen(angestellte: Seq[Angestellter]) = {
  angestellte.sortBy(_.gehalt).takeRight(10)
}
```

Eine Methode, die aus einer Sequenz von Angestellten eine Sequenz von Strings macht, die jeweils aus Name und Abteilung eines Angestellten bestehen:

```
def namen(angestellte: Seq[Angestellter]) = {
  angestellte.map(a => a.name + ": " + a.abteilung)
}
```

[1] Siehe zu dem Thema auch [1] und [2].

[2] Scala generiert Getter und Setter, aber nicht nach dem Bean-Standard.

[3] Eine Sequenz in Scala ist, vereinfacht gesagt, das Pendant zu *java.util.Collection*.

Eine Methode, die von einer Sequenz von Mitarbeitern alle externen Mitarbeiter zurückliefert:

```
def externe(mitarbeiter: Seq[Mitarbeiter]) = {
  mitarbeiter.filter(_.istExtern)
}
```

Eine Methode, die beliebige Objekte[4] in einen String umwandelt:

```
def asString(x: Any) = x match {
  case d: Date => dateFormat.format(d)
  case i: Int if(i < 100) => "Eine kleine Zahl"
  case "" => "Der leere String"
  case null => "null"
  case _ => x.toString
}
```

Eine Methode, welche die Fakultät einer Zahl berechnet, wobei BigInt[5] verwendet wird, um Überläufe zu vermeiden:

```
def fakultaet(n: BigInt): BigInt = {
  if (n == 0) 1 else n * fakultaet(n – 1)
}
```

Augenscheinliche Unterschiede

Es folgt ein Hello-World-Programm, um einige Besonderheiten der Sprache zu erläutern:

```
object HelloWorld {
  def main(args: Array[String]) {
    hello("World")
  }

  def hello(s: String) = {
    println("Hello " + s)
  }
}
```

Man sieht, dass die *main*-Methode nicht statisch ist. Statische Methoden oder Felder gibt es generell nicht. Stattdessen gibt es die Möglichkeit Objekte mit Hilfe des Schlüsselworts *object* zu definieren. Sie sind benannte Singleton-Instanzen, die unter anderem als Ersatz für alle statischen Konstrukte Javas dienen.

Weiter fällt auf, dass die Methodendefinitionssyntax ungewohnt ist. Eine Methodendefinition wird stets mit dem Schlüsselwort *def* eingeleitet und die Typen der Parameter werden durch einen Doppelpunkt getrennt hinter dem Namen des Parameters notiert.

Die Methoden sind ohne Sichtbarkeitsmodifikator definiert. In Java entspräche dies der Sichtbarkeit *package-private*, in Scala ist die Standardsichtbarkeit *public*.

Außerdem enthält der Code keine Semikolons, da diese in Scala optional sind. Der Compiler versucht sie bei deren Fehlen nach bestimmten Regeln selbst zu setzen. Dies nennt man auch Semikoloninferenz.

Definitionssyntax

Im Folgenden sollen nun nach und nach die wichtigsten Sprachkonstrukte vorgestellt werden, beginnend mit der Syntax für Definitionen. Zuerst eine Gegenüberstellung der Syntax von Java auf der linken Seite und Scala auf der rechten Seite:

`int i;`	`var i: Int`
`final int i;`	`val i: Int`
`void method() { … }`	`def method {…}`
`int num() { return res; }`	`def num: Int = res`
`int add(int a) { ... }`	`def add(a: Int): Int = {…}`
`class Foo<T> { ... }`	`class Foo[T] { ... }`

Man sieht, dass Definitionen in Scala stets mit einem Schlüsselwort, wie *val*, *def* und *class* eingeleitet werden und Typangaben durch einen Doppelpunkt getrennt nachgestellt werden. Weiterhin können die geschweiften Klammern um die Rümpfe von Methoden weggelassen werden, wenn diese aus nur einem Ausdruck bestehen. Zu beachten ist, dass Methoden implizit den Rückgabetyp *Unit*, das Äquivalent zu Javas *void*, haben, wenn nach der Methodensignatur kein Gleichheitszeichen gesetzt ist. Außerdem ist das *return* optional – wird es weggelassen, gibt die Methode den Wert des letzten Ausdrucks zurück.

Typinferenz

Einer der Gründe für die geänderte Definitionssyntax, im Vergleich zu Java, ist die Typinferenz. Sie bewirkt, dass der Compiler den Typ einer Definition auch ohne explizite Angabe bestimmt und verwendet. Dazu ein Beispiel:

```
var i = 42
```

Dieser Ausdruck ist nicht untypisiert, sondern stattdessen wird der Typ vom Compiler bestimmt. Der Compiler wählt dabei immer den spezifischsten, passenden Typen.

[4] Any ist Supertyp für jeden anderen Typen und damit die Wurzel der Typhierarchie in Scala.

[5] BigInt ist ein Wrapper um java.math.BigInteger aus der Scala-Standardbibliothek.

In diesem Fall wird der Typ von *i* auf *Int* festgelegt und das Programm verhält sich genau so, als hätte man dies explizit angegeben. Würde man im Folgenden versuchen, der Variable *i* einen Wert vom Typ *String* zuzuweisen, würde es beim Compilieren zu einem Fehler kommen. Auch mit Typinferenz ist somit die statische Typprüfung gewährleistet.

Ausdrücke

Betrachten wir, wie die Syntax für Ausdrücke aussieht. Prinzipiell ist sie Javas sehr ähnlich, allerdings gibt es einige Besonderheiten, die zu beachten sind. Methodenaufrufe können nicht nur wie in Java geschrieben werden:

obj.methode(*arg*)

sondern auch in Infixnotation, wie man es sonst nur von Operatoren, wie +, kennt:

obj methode *arg*

Dies ist möglich, weil Scala generell nicht zwischen Methoden und Operatoren unterscheidet. Daher sind beispielsweise auch folgende Ausdrücke äquivalent[6]:

1 + 2 und 1.+(2)

Mancher Leser wird sich nun fragen, wie die mathematischen Grundregeln, wie Punkt vor Strich, realisiert werden. Dafür wird das erste Zeichen des Methodennamens zur Präzedenzbestimmung herangezogen. Eine Methode, die mit dem Zeichen + beginnt, hat niedrigere Präzedenz, als eine Methode, die mit dem Zeichen * beginnt. Somit ist gewährleistet, dass sich die Operatorähnlichen Methoden von der Präzedenz her so verhalten, wie man es von den Operatoren aus Java gewohnt ist. Die Präzedenz ist aber nur relevant, wenn man die Infixnotation nutzt. Herkömmliche Methodenaufrufe mit Punkt werden, wie gewohnt, von links nach rechts ausgewertet.

Bedingte Ausdrücke werden mit *if* notiert. Allerdings ist das *if*, im Gegensatz zu Java, ein Ausdruck und keine Anweisung, was den Fragezeichenoperator überflüssig macht:

if (*cond*) *expr1* **else** *expr2*
aber auch
val *v* = **if** (*cond*) *expr1* **else** *expr2*

Schleifen werden mit *while* oder mit *for* geschrieben, wobei sich die *for*-Schleife von Javas unterscheidet und in der Ausdrucksstärke weit darüber hinausgeht. Daher wird sie in einem späteren Artikel noch einmal im Detail betrachtet und hier nur in ihrer einfachsten Form vorgestellt. Die *while*-Schleife hingegen gleicht der von Java:

for (*element* <- *collection*) { *expr* }
while (*cond*) { *expr* }

Do it youself

Wer selbst mit der Sprache experimentieren will, dem sei empfohlen sich unter [3] das *Current Stable Release* herunterzuladen und den interaktiven Modus, genannt REPL[7], auszuprobieren. Den REPL-Modus startet man, indem man mit einer Kommandozeile in das Unterverzeichnis *bin* der Scala-Distribution wechselt und dort das Script *scala* startet.

Wer gleich ein Projekt in seiner Lieblings-IDE aufsetzen will, der hat die Wahl zwischen den Plugins für Eclipse [4], Netbeans [5] und IntelliJ IDEA [6].

Abschließend

Damit ist der erste Einblick in die Sprache Scala abgeschlossen. Die elementaren Unterschiede zu Java wurden vorgestellt und es ist hoffentlich gelungen, die Neugierde zu wecken und Lust auf mehr zu machen. Der nächste Artikel wird sich mit Klassen, Traits und Objekten beschäftigen.

Referenzen

[1] Neal Gafter's blog *Failure of Imagination in Language Design*
http://gafter.blogspot.com/2006/09/failure-of-imagination-in-language_17.html

[2] Stephen Colebourne's Weblog *Java language design by use case*
http://www.jroller.com/scolebourne/entry/java_language_design_by_use

[3] Scala *Current Stable Release*
http://www.scala-lang.org/downloads

[4] scala-ide
http://www.scala-ide.org

[5] NetBeans *scala*
http://wiki.netbeans.org/Scala

[6] JetBRAINS *Scala Plugin for IntelliJ IDEA*
http://confluence.jetbrains.net/display/SCA

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.

[6] Oder doch nicht ganz äquivalent, denn im zweiten Fall wird die Eins als Fließkommazahl interpretiert. Richtig müsste es also (1).+(2) heißen.

[7] REPL steht für *Read Evaluate Print Loop* und entspricht in etwa den von früher bekannten Interpretern.

Des Programmierers kleine Vergnügen

Kleine Verwechslungen

von Michael Wiedeking

Haben Sie das auch schon erlebt, dass ein Lehrer oder Professor ein Thema mit den Worten *der Rest sei evident* oder *trivial* beendet hat. Mir ist das des Öfteren passiert. Und wahrscheinlich habe ich mich deswegen beim letzten Vergnügen dafür gerächt und es ungerechter Weise an Ihnen ausgelassen. Habe ich doch zum Vertauschen der Bits einen Verweis angegeben, der sich im Nachhinein als suboptimal herausstellte.

Wer sich noch erinnert, weiß, dass es im letzten Vergnügen [1] darum ging, bei einem Bit-Muster zwei Bits zu invertieren und anschließend zu vertauschen. Sei $x = (x_{31}x_{30}\ldots x_1x_0)_2$ ein Wort mit – der Einfachheit halber – 32 Bit x_i. In diesem Wort sollen gemäß der gestellten Aufgabe, die beiden niederwertigsten Bits x_1 und x_0 vertauscht werden. Allgemein geht es zunächst also um das Problem, zwei Bits x_i und x_j zu vertauschen (wobei man dabei – ohne die Funktionstüchtigkeit einzuschränken – davon ausgehen kann, dass $i > j$ ist).

Das als Lösung ganz lapidar zitierte Vergnügen [2] beschäftigte sich zwar mit dem Austausch von gleich positionierten Bits verschiedener Wörter, ohne dabei eine Hilfsvariable zu Hilfe zu nehmen, aber das Prinzip funktioniert natürlich auch mit Bits im selben Register.

```
int swapBits(int x, int i, int j) {
  int delta = i – j;
  int y = (x ^ (x >>> delta)) & (0x1 << j);
  return x ^ y ^ (y << delta);
}
```

Die erste Anweisung schiebt sozusagen das eine Bit x_i über das andere Bit x_j, exklusiv-odert beide aus und isoliert das Ergebnis. Die nächste Anweisung schiebt dieses isolierte Bit sowohl über das eine Bit x_i als auch das andere Bit x_j und exklusiv-verodert wieder beide. Damit sind, wie in [2] ausführlicher erklärt wurde, durch drei Exklusiv-Oder die beiden Bit-Werte von x_i und x_j ausgetauscht worden.

Für das Tauschen benötigen wir insgesamt 8 Instruktionen. In unserem eigentlichen Anwendungsfall stehen allerdings die beiden Positionen mit $i = 1$ und $j = 0$ fest, und damit kann auf einige Operationen verzichtet werden.

```
int swapBits(int x) {
  int y = (x ^ (x >>> 1)) & 0x1;
  return x ^ y ^ (y << 1);
}
```

Damit sind nur noch 6 Instruktionen nötig. Wenn man aber schon mal dabei ist, die Besonderheiten zu berücksichtigen, so kann man auch ein anderes Verfahren nutzen, dass allerdings im allgemeinen Fall aufwändiger ist. Anstatt die betroffenen Bits durch ein Exklusiv-Oder zu tauschen, kann man nämlich die Bits einfach extrahieren, vertauschen und wieder einfügen.

```
int swapBits(int x, int i, int j) {
  int delta = i – j;
  int Mj =  0x1 << j;
  int Mij =  (0x1 << i) | Mj;
  int a =  (x >>> delta) & Mj;
  int b =  (x & Mj) << delta;
  return (x & ~Mij) | a | b;
}
```

Mit 12 Operationen benötigt man also deutlich mehr als die obigen 8 Instruktionen. Dafür kann man diese Variante aber auch bei mehr als einem Bit nutzen: Mit Hilfe geeigneter Masken lassen sich so auch größere, sich nicht überlappende Bit-Gruppen mit vergleichbaren Aufwand austauschen.

Alleine 4 der Operationen entfallen auf die Erstellung der benötigten Masken. Geht man nun wieder vom Spezialfall $i = 1$ und $j = 0$ aus, so vereinfacht sich der Code enorm.

```
int swapBits(int x) {
  int a = (x >>> 1) & 0x1;
  int b = (x & 0x1) << 1;
  return (x & 0xFFFFFFFC) | a | b;
}
```

Liegen alle Masken als Konstanten vor, reduziert sich der Aufwand auf 7 Instruktionen. Der Code lässt sich aber noch weiter reduzieren, wenn man sich zunutze macht,

dass außer x_1 und x_0 alle Bits auf 0 gesetzt sind. Dann darf das erste *Und* entfallen, da das nicht benötigte Bit ohnehin „rausgeschoben“ wird. Jetzt befinden sich in a und b die beiden interessanten Bits. Beim Zusammenbau stehen also alle Informationen zur Verfügung und ein weiterer Zugriff auf x ist gar nicht mehr nötig.

```
int swapBits(int x) {
  int a = x >>> 1;
  int b = (x & 0x1) << 1;
  return a | b;
}
```

Damit werden nur noch unschlagbare 4 Instruktionen benötigt.

Bezogen auf das Originalproblem kommt jetzt nur noch das Invertieren der Bits hinzu. Natürlich dürfen nicht alle Bits invertiert werden, sondern nur die beiden zu vertauschenden oder vertauschten Bits. Und das lässt sich diesmal wirklich trivial erledigen, indem man diese mit der Maske 0x3 exklusiv-verodert.

Den Zweiflern unter Ihnen, die meinen, Optimierungen dieser Art spielten doch keine Rolle und seien den Denkaufwand nicht wert, dem soll an dieser Stelle verdeutlicht werden, was die Reduktion der Instruktionen um die Hälfte bedeutet: Anstatt von morgens um acht Uhr bis nachmittags um fünf Uhr arbeiten zu müssen, können Sie in Zukunft schon vor dem Mittagessen um zwölf Uhr nach Hause gehen. Und für wen sich das nicht lohnt, dem kann natürlich auch hiermit nicht geholfen werden.

Referenzen

[1] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Vielleicht, vielleicht auch nicht*, KaffeeKlatsch, Jahrgang 3, Nummer 11, Seite 14, November, 2010

[2] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Exklusivsubstitution*, KaffeeKlatsch, Jahrgang 1, Nummer 9, Seite 21, September, 2008

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

Die Sprache der Liebe

von Alexandra Specht

Ja, das Fest der Liebe naht; was liegt da näher, als etwas über die Sprache der Liebe zu schreiben! Und natürlich liegt es in der Natur dieser Kolumne, dass es sich auch um Informatiker drehen wird.

Ich habe in der letzten Zeit Änderungen in der Sprache meiner Kollegen beobachtet. Sie ist weicher geworden, persönlicher, netter, liebevoller. Ich weiß nicht, ob sich tatsächlich etwas geändert hat, oder ob meine Wahrnehmung anders geworden ist; ob mir einmal etwas aufgefallen ist und ich daraufhin einfach mehr darauf geachtet habe.

Ja, Sie wissen ja selber, dass gerade im Web-Oberflächen-Design einiges hübsch gemacht werden muss, damit alle, vor allem die Kunden, zufrieden sind. Dann sagen Sie schon mal etwas über das Look-and-Feel. Und dass es *sanft* wirken muss. Sanft! Wie nett! Sanfte GUIs. Warum nicht auch *zart*? Das klingt dann nicht nur lieb, sondern gleich nach Märchen, nach Feen und einem Zauberwald, in dem alle (IT-)Wünsche umgehend wahr werden. Oder vielleicht nicht alle, aber die obligatorischen drei Wünsche.

Ich habe auch noch andere Sachen gehört. Zum Beispiel, dass der Code *schön* gemacht werden muss. Schön wie eine Prinzessin zu Weihnachten? Schöner Code: auch sehr nett. Noch netter als *sauberer* Code, wobei ich das schon sehr beeindruckend finde, dass sich zunehmend Entwickler Gedanken darüber machen, dass der Code nicht nur funktioniert, sondern auch noch sauber sein soll, gut nachvollziehbar, gut wartbar und was noch alles.

Ein Entwickler meinte einmal, dass die von ihm gerade betrachtete Applikation *sexy* sei. Ist Software sexy? Offensichtlich kann sie für diejenigen, die das beurteilen können, schon sexy sein, wobei man das zwar erst noch definieren müsste; aber generell sind wir uns über die grundlegende Bedeutung der Aussage sicherlich einig – nehme ich an. Sexy heißt hier nicht mehr und nicht weniger, als dass die Software attraktiv, schön und gut gemacht ist.

Naja, und wenn dann doch irgendwelche Fehler auftauchen, dann wird zwar erst einmal geschimpft, aber nur heimlich im Stillen mit dem Rechner. Gerade mal, dass dann nicht die „Mädchen-Methode“ gewählt wird; ich nehme an, Sie wissen, was ich meine.[1]

Aber wenn darüber geredet wird, dann ist es kein Fehler, sondern das Programm ist eben noch ein bisschen *buggy*. Buggy, das klingt definitiv schöner als fehlerhaft. Das klingt liebevoll. Wie ein Kind, das nicht störrisch genannt wird, sondern eben ein kleiner Dickkopf ist. Da stört mich noch nicht einmal, dass *buggy* kein deutsches Wort ist; das ist einfach ein hübscher kleiner Euphemismus für etwas, dass eben noch nicht perfekt ist, aber geschätzt wird, weil so viel Zeit und Energie investiert wurde.

Nun ja. Das ist doch eigentlich auch die Hauptsache, dass man etwas (beruflich) macht, was man gerne macht. Das man etwas schafft, was einem am Herzen liegt, was man schön findet, sexy, die Fehler liebevoll wahrnimmt und versucht, es zu verbessern.

Man sagt ja auch, dass es Liebe ist, wenn man jemand (oder etwas) nicht trotz seiner Fehler liebt, sondern gerade deswegen.

Apropos Liebe: Ich wünsche Ihnen allen wunderschöne Weihnachten und ein Jahr voller Liebe, zu was und wem auch immer, Hauptsache viel davon!

Kurzbiographie

Alexandra Specht (alexandra.specht@mathema.de) arbeitet bei der MATHEMA Software GmbH als Account-Managerin. Sie kümmert sich um Vertrieb, Key-Account, Personalplanung, Konferenzen etc. Nebenbei beschäftigt sie sich mit Vergnügen und so viel es geht mit Literatur und Sprache. Kommentare und Feedback nimmt sie gerne per E-Mail entgegen.

[1] Wenn nicht, dann hier die Lösung: Draufhauen, natürlich. Hilft immer!

Kaffeesatz

Geplatzte Platzreservierung

von Michael Wiedeking

Man kann wohl davon ausgehen, dass das Gros der Anwender in der Regel überhaupt keine Ahnung hat, was für Aufwände dahinter stecken, ihm eine präsentable und bedienbare Web-Oberfläche zu bieten. Oftmals haben wir diesen Aufwand selbst verschuldet, weil wir Entwickler glauben, Vorgänge müssten so abgebildet werden, wie wir es gewohnt sind, aber oftmals ginge es viel einfacher.

Nehmen wir einmal die Platzreservierung. Geht man an den Schalter, dann gibt es in der Regel nur zwei Möglichkeiten, an Karten zu kommen. Bei der einen, fast schon ausgestorbenen Variante, fragt man nach Karten für ein bestimmtes Konzert und dann sagt der Vorverkäufer: „Da haben wir noch die und die Plätze," und sagt dazu deren Kategoriepreise. Jetzt kann man sich die benötigten Karten aussuchen oder man muss zu einem anderen Vorverkauf.

In Zeiten des Internets gibt es das fast nicht mehr. In der Regel dreht der Vorverkäufer den Bildschirm in Richtung des Kunden, tippt auf eines der gelben Felder und sagt: „Gelb ist noch frei," und erwähnt vielleicht noch, wo sich auf dem Plan überhaupt die Bühne befindet. Dann wählt man aus und bekommt tatsächlich die gewünschten Plätze.

Wenn nicht just in diesem Moment jemand anderes die Karten gekauft hat. Typischerweise sind die Terminals an einen Zentralrechner gekoppelt, und wenn nun Sitze ausgewählt werden, so werden diese für alle anderen momentan gesperrt. Solange es sich um Terminals aller Vorverkaufsstationen handelt ist das kein Problem: Die Wege sind kurz, die Anzahl der Teilnehmer limitiert und die Wahrscheinlichkeit eines Konflikts relativ gering.

Bringt man das Ganze ins Internet und hat man es plötzlich mit mehreren tausend (Web-)Terminals zu tun, dann ändert sich die Situation drastisch. Plötzlich merkt man, dass es lange Wege gibt, Verbindungen, die nicht den nötigen Durchsatz haben und Latenzzeiten, die zu Depressionen führen.

Jetzt stellt sich natürlich die Frage, ob das wirklich sein muss. Eine pragmatische Lösung ist hier, auf den Luxus zu verzichten, den Benutzer seine Plätze auswählen zu lassen. Auch wenn er den Platz auf dem Sitzplan sieht, so hat er doch keine Ahnung, ob er wirklich gut und dem Preis angemessen ist. Wäre es da nicht viel einfacher, ihm einfach die besten, noch zur Verfügung stehenden Plätze innerhalb einer Kategorie zur Verfügung zu stellen, ohne zu sagen wo die sich befinden?

Dann hat man es plötzlich ganz einfach: Keine Transaktionen mehr, keine Synchronisationsprobleme und keine der Probleme, die bei ständigem Wechsel der Verfügbarkeit und der daraus erforderlichen Bildschirmauffrischungen auftreten. Dafür kommen alle Anfragen schön seriell herein, können bequem der Reihe nach abgearbeitet werden und falls wirklich mal etwas schiefgehen sollte, bekommt der Kunde in seiner Bestätigungsmail halt mal eine schlechte Nachricht.

Es scheint also, dass es sich durchaus lohnen kann, alte Verfahren zu überdenken. Und wenn man Glück hat, kann man den Aufwand auf ein Minimum reduzieren. Ach wenn doch alles so einfach wäre, wie das Reservieren von Theaterkarten.

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.

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 Hamburg
Java User Group Hamburg
http://www.jughh.org

JUG Deutschland e.V.
Java User Group Deutschland e.V.
c/o asc-Dienstleistungs GmbH
Ehrengard-Schramm-Weg 11, 37085 Göttingen
http://www.java.de (office@java.de)

rheinjug e.V.
Java User Group Düsseldorf
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf
http://www.rheinjug.de
Kontakt: Herr Heiko Sippel (info@rheinjug.de)

Java User Group Saxony
Java User Group Dresden
http://www.jugsaxouy.de
Kontakt: Herr Torsten Rentsch (torsten@jugsaxony.de)
Herr Falk Hartmann (falk@jugsaxony.de)
Herr Kristian Rink (kristian@jugsaxony.de)

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
Universitätsstr. 12, 45127 Essen
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
(javausergroupfrankfurt@googlemail.com)

JUG Karlsruhe
Java User Group Karlsruhe
Universität Karlsruhe, Gebäude 50.34
Am Fasanengarten 4, 76131 Karlsruhe
http://jug-ka.de
Kontakt: David Linsin

JUGC
Java User Group Köln
http://www.jugcologne.org
Kontakt: Herr Michael Hüttermann
(michael@huettermann.net)

JUG Münster
Java User Group für Münster und das Münsterland
http://www.jug-muenster.de
Kontakt: Herr Thomas Kruse (tkjugi@sforce.org)

JUGS e.V.
Java User Group Stuttgart e.V.
c/o Dr. Michael Paus
Schönaicherstraße 3, 70597 Stuttgart
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)

JUG Berlin Brandenburg
http://www.jug-bb.de
Kontakt: Herr Ralph Bergmann (orga@jug-bb.de)

jugm
Java User Group München
Jupiterweg 8, 85586 Poing
http://www.jugm.de
Kontakt: Herr Andreas Haug (ah@jugm.de)

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)
Siekstraße 4, 38444 Wolfsburg
http://www.jug-ostfalen.de
Kontakt: Uwe Sauerbrei (info@jug-ostfalen.de)

Schweiz

JUGS
Java User Group Switzerland
Postfach 2322, 8033 Zürich
http://www.jugs.ch (info@jugs.ch)
Kontakt: Frau Ursula Burri

.Net User Groups

Deutschland

DNUG-Köln
DotNetUserGroup Köln
Goldammerweg 325, 50829 Köln
http://www.dnug-koeln.de
Kontakt: Herr Albert Weinert (info@der-albert.com)

.net Usergroup Frankfurt
c/o Thomas Sohnrey, Agile IService
Mendelssohnstrasse 80, 60325 Frankfurt
http://www.dotnet-ug-frankfurt.de
Kontakt: Herr Thomas 'Teddy' Sohnrey
(thomas.sohnrey@gmx.de)

.Net User Group Leipzig
Brockhausstraße 26, 04229 Leipzig
http://www.dotnet-leipzig.de
Kontakt: Herr Alexander Groß (agross@dotnet-leipzig.de)
Herr Torsten Weber (tweber@dotnet-leipzig.de)

.Net User Group Bonn
.NET User Group "Bonn-to-Code.Net"
Langwartweg 101, 53129 Bonn
http://www.bonn-to-code.net (mail@bonn-to-code.net)
Kontakt: Herr Roland Weigelt

Dodned
.NET User Group Franken
http://www.dodned.de
Kontakt: Herr Bernd Hengelein
Herr Thomas Müller (consulting@tom-mue.de)

.NET User Group Oldenburg
c/o Hilmar Bunjes und Yvette Teiken
Sachsenstr. 24, 26121 Oldenburg
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 OWL
http://www.gedoplan.de/cms/gedoplan/ak/ms_net
c/o GEDOPLAN GmbH
Stieghorster Str. 60, 33605 Bielefeld

.NET User Group Paderborn
c/o Net at Work Netzwerksysteme GmbH,
Am Hoppenhof 32, 33104 Paderborn
http://www.dotnet-paderborn.de
(raacke@dotnet-paderborn.de)
Kontakt: Herr Mathias Raacke

.net Developer-Group Ulm
c/o artiso solutions GmbH
Oberer Wiesenweg 25, 89134 Blaustein
http://www.dotnet-ulm.de
Kontakt: Herr Thomas Schissler (tschissler@artiso.com)

.Net Developers Group Stuttgart
Tieto Deutschland GmbH
Mittlerer Pfad 2, 70499 Stuttgart
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
Kontakt: Frau Catrin Busley

INdotNET
Ingolstädter .NET Developers Group
http://www.indot.net
Kontakt: Herr Gregor Biswanger
(gregor.biswanger@web-enliven.de)

.NET DGH
.NET Developers Group Hannover
Landwehrstraße 85, 30519 Hannover
http://www.dotnet-hannover.de
Kontakt: Herr Friedhelm Drecktrah (friedhelm@drecktrah.de)

Österreich

.NET Usergroup Rheintal
c/o Computer Studio Kogoj
Josefgasse 11, 6800 Feldkirch
http://usergroups.at/blogs/dotnetusergrouprheintal/default.aspx
Kontakt: Herr Thomas Kogoj (thomas@kogoj.com)

.NET User Group Austria
c/o Global Knowledge Network GmbH,
Gutheil Schoder Gasse 7a, 1101 Wien
http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx
Kontakt: Herr Christian Nagel (ug@christiannagel.com)

Herbstcampus

Wissenstransfer par excellence

Der **Herbstcampus** ist *die* Konferenz für Software-Entwickler und -Architekten mit den Technologieschwerpunkten Java und .Net

Der **Herbstcampus** bietet ein umfassendes und hochwertiges Vortragsprogramm mit namhaften Referenten, das den Teilnehmern wichtiges Know-how vermittelt und über sämtliche aktuellen Entwicklungen informiert

5. – 8. September 2011 in Nürnberg

www.herbstcampus.de

Das Allerletzte

…

// We trust that the [...] information* is in
// the same order as the rows in the table.
// If no one messes with the code,
// everything should be fine.

…

* information: Ergebnis einer *native query* aus der DB

Dies ist kein Scherz!
Dieser bemerkenswerte Kommentar wurde tatsächlich
in der freien Wildbahn angetroffen.

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 Mitte Januar 2011

Herbstcampus

Wissenstransfer par excellence

5. – 8. September 2011

in Nürnberg

http://www.herbstcampus.de