KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

07/2013

Jahrgang 6

KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

Sie können die elektronische Form des KaffeeKlatschs
monatlich, kostenlos und unverbindlich
durch eine E-Mail an

abo@bookware.de

abonnieren.

Ihre E-Mail-Adresse wird ausschließlich für den Versand
des KaffeeKlatschs verwendet.

Editorial

DRY, SOLID und GESPÖNZ

Unsere Branche bringt durchaus einige Merkwürdigkeiten mit sich. So zum Beispiel die, dass bei jeder passenden und unpassenden Gelegenheit ein neues Akronym das Licht der Welt erblickt. Dass etwa „What You See Is What You Get“ zu WYSIWYG abgekürzt wird, mag ja noch nachvollziehbar sein. Aber „Don’t Repeat Yourself“ heißt jetzt plötzlich nur noch DRY oder statt „You Ain’t Gonna Need It“ ist nur noch von YAGNI die Rede.

Vielleicht gibt ja eine Abkürzung, die wie ein Adjektiv klingt, einfach mehr her. Aber nicht immer eignet sich ein Ausspruch oder Prinzip zu einer derartigen Verkürzung. Nichtsdestoweniger wird, was nicht passt, passend gemacht. So wird das in der *Perl*-Gemeinde bekannte „There Is More Than One Way To Do It“ nicht nur zu TIMTOWTDI komprimiert, sondern gleich noch die passende Aussprache geliefert: „Tim Toady“.

Immerhin mag eine solche Abkürzung als Gedächtnisstütze dienen. Aus den Anfangsbuchstaben der fünf bekanntesten Design-Prinzipien etwa (*Single-Responsibility*-Prinzip, *Open-Closed*-Prinzip, Liskovsches Substitutionsprinzip, *Interface-Segregation*-Prinzip und *Dependency-Inversion*-Prinzip) lässt sich das Wort „SOLID“ bilden – ein sehr schönes Wort für etwas, dass die Qualität von Software signifikant beeinflussen kann.

Solch passende Zufälle ergeben sich allerdings nicht besonders häufig. Will man etwa brauchbare Software entwickeln, so sollte diese bestimmt bezüglich der folgenden Punkte ausgereift sein:

- *Sicherheit*: Daten wie Passwörter und persönliche Daten sollten vor fremden Zugriffen geschützt sein, Kommunikation möglichst abhörsicher vonstatten gehen und fälschungssicher sein.
- *Performanz*: Die Software sollte schnell (genug) sein und in angemessenem Umfang skalieren.
- *Zuverlässigkeit*: Die Funktionstüchtigkeit sollte auch bei unsachgemäßer Bedienung (ungewollt oder gewollt) immer gewährleistet sein.
- *Nutzbarkeit*: Sowohl für die Anwender als auch für die Entwickler müssen die Ziele einer Anwendung und auch die damit zusammenhängenden Prozesse klar und dokumentiert sein. Eine einfache Bedienung ist dabei genau so wichtig wie eine brauchbare, erweiterbare Architektur.
- *Gemeinschaft*: Möglicherweise braucht es zu der Anwendung Raum, damit sich die Anwender oder Entwickler austauschen, Wünsche und Fehler dokumentieren, diskutieren und bewerten können.
- *Effizienz*: Neben der Effektivität sollte ein sorgsamer, ausgewogener Umgang mit den zur Verfügung stehenden Ressourcen gewährleistet sein.
- *Ökonomie*: Die mit einem Einsatz verbundenen Kosten müssen transparent sein – vom Lizenzmodell über die Wartungskosten bis hin zu den Betriebskosten.

Aus deren Anfangsbuchstaben lässt sich gerade noch das Wort GESPÖNZ bilden, weil unsere Hauptwörter all zu oft mit einem Konsonanten beginnen.

Wenigstens kann man so relativ einfach vom DRY-, SOLID- oder womöglich dem GESPÖNZ-Prinzip sprechen ohne in einem Wald von Bindestrichen zu versinken.

Übrigens wusste schon unser „Blödelbarde“ Otto völlig fachfremd von dieser Problematik zu berichten und sang: „G“ das steht für Güte, die ich stets in Dir seh’; „N“ ist Deine Nähe, wenn ich nah bei Dir steh’; „L“, das steht für Leidenschaft, die Du in mir entfachst; „P“, das ist die Poesie, wenn Du so zärtlich lachst; „F“, das steht für Freundschaft, die Du mir ständig gibst; „T“, das ist die Treue, weil Du mich wirklich liebst; „H“, das ist das Herz, das für mich schlägt bei Tag und Nacht; tu alles zusammen, ja dann heißt es: GNLPFTH.

In diesem Sinne: VSBDL[1],
Ihr Michael Wiedeking

[1] VSBDL = Viel Spaß bei der Lektüre.

Beitragsinformation

Der KaffeeKlatsch dient Entwicklern, Architekten, Projektleitern und Entscheidern als Kommunikationsplattform. Er soll neben dem Know-how-Transfer von Technologien (insbesondere Java und .NET) auch auf einfache Weise die Publikation von Projekt- und Erfahrungsberichten ermöglichen.

Beiträge

Um einen Beitrag im KaffeeKlatsch veröffentlichen zu können, müssen Sie prüfen, ob Ihr Beitrag den folgenden Mindestanforderungen genügt:

- Ist das Thema von Interesse für Entwickler, Architekten, Projektleiter oder Entscheider, speziell wenn sich diese mit der Java- oder .NET-Technologie beschäftigen?
- Ist der Artikel für diese Zielgruppe bei der Arbeit mit Java oder .NET relevant oder hilfreich?
- Genügt die Arbeit den üblichen professionellen Standards für Artikel in Bezug auf Sprache und Erscheinungsbild?

Wenn Sie uns einen solchen Artikel, um ihn in diesem Medium zu veröffentlichen, zukommen lassen, dann übertragen Sie Bookware unwiderruflich das nicht exklusive, weltweit geltende Recht

- diesen Artikel bei Annahme durch die Redaktion im KaffeeKlatsch zu veröffentlichen
- diesen Artikel nach Belieben in elektronischer oder gedruckter Form zu verbreiten
- diesen Artikel in der Bookware-Bibliothek zu veröffentlichen
- den Nutzern zu erlauben diesen Artikel für nicht-kommerzielle Zwecke, insbesondere für Weiterbildung und Forschung, zu kopieren und zu verteilen.

Wir möchten deshalb keine Artikel veröffentlichen, die bereits in anderen Print- oder Online-Medien veröffentlicht worden sind.

Selbstverständlich bleibt das Copyright auch bei Ihnen und Bookware wird jede Anfrage für eine kommerzielle Nutzung direkt an Sie weiterleiten.

Die Beiträge sollten in elektronischer Form via E-Mail an redaktion@bookware.de geschickt werden.

Auf Wunsch stellen wir dem Autor seinen Artikel als unveränderlichen PDF-Nachdruck in der kanonischen KaffeeKlatsch-Form zur Verfügung, für den er ein unwiderrufliches, nicht-exklusives Nutzungsrecht erhält.

Leserbriefe

Leserbriefe werden nur dann akzeptiert, wenn sie mit vollständigem Namen, Anschrift und E-Mail-Adresse versehen sind. Die Redaktion behält sich vor, Leserbriefe – auch gekürzt – zu veröffentlichen, wenn dem nicht explizit widersprochen wurde.

Sobald ein Leserbrief (oder auch Artikel) als direkte Kritik zu einem bereits veröffentlichten Beitrag aufgefasst werden kann, behält sich die Redaktion vor, die Veröffentlichung jener Beiträge zu verzögern, so dass der Kritisierte die Möglichkeit hat, auf die Kritik in der selben Ausgabe zu reagieren.

Leserbriefe schicken Sie bitte an leserbrief@bookware.de. Für Fragen und Wünsche zu Nachdrucken, Kopien von Berichten oder Referenzen wenden Sie sich bitte direkt an die Autoren.

Werbung ist Information

Firmen haben die Möglichkeit Werbung im KaffeeKlatsch unterzubringen. Der Werbeteil ist in drei Teile gegliedert:

- Stellenanzeigen
- Seminaranzeigen
- Produktinformation und -werbung

Die Werbeflächen werden als Vielfaches von Sechsteln und Vierteln einer DIN-A4-Seite zur Verfügung gestellt.

Der Werbeplatz kann bei Frau Natalia Wilhelm via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-16 gebucht werden.

Abonnement

Der KaffeeKlatsch erscheint zur Zeit monatlich. Die jeweils aktuelle Version wird nur via E-Mail als PDF-Dokument versandt. Sie können den KaffeeKlatsch via E-Mail an abo@bookware.de oder über das Internet unter www.bookware.de/abo bestellen. Selbstverständlich können Sie das Abo jederzeit und ohne Angabe von Gründen sowohl via E-Mail als auch übers Internet kündigen.

Ältere Versionen können einfach über das Internet als Download unter www.bookware.de/archiv bezogen werden.

Auf Wunsch schicken wir Ihnen auch ein gedrucktes Exemplar. Da es sich dabei um einzelne Exemplare handelt, erkundigen Sie sich bitte wegen der Preise und Versandkosten bei Natalia Wilhelm via E-Mail unter natalia.wilhelm@bookware.de oder telefonisch unter 09131/8903-16.

Copyright

Impressum

KaffeeKlatsch Jahrgang 6, Nummer 7, Juli 2013
ISSN 1865-682X
Bookware – eine Initiative der
MATHEMA Verwaltungs- und Service-Gesellschaft mbH

Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de
Internet: www.bookware.de

Herausgeber/Redakteur: Michael Wiedeking
Anzeigen: Natalia Wilhelm
Grafik: Nicole Delong-Buchanan

Inhalt

Artikel

Kolumnen

Unbeabsichtigt monadisch

Funktionale Entwicklung ist wie die dunkle Seite der Macht. Begibt man sich einmal auf diesen Pfad, kontrolliert sie die Gedanken des Entwicklers für immer, scheinbar auch wenn es um *Java*-Code geht. Folgendes beruht auf einer wahren Begebenheit.

Weiter so!

Gelegentlich hat man es mit Daten zu tun die sowohl paarweise als auch allein auftreten. Speichert man diese etwa in einem *Array* ab, so ergibt sich das Problem, dass man zwar die Stelle *i* bearbeitet, aber das nächste Element an einer geraden oder ungeraden Stelle liegen muss. Wie aber findet man effizient zum nächsten geraden oder ungeraden Index?

me <3 #omnom

Geeks und Nerds haben eine eigene Sprache, eigenen Humor – letztlich eine ganz eigene Subkultur. Diese ist von Wortspielereien, Katzenfotos und anderen auf Außenstehende seltsam anmutenden Dingen geprägt. Dies ist der Versuch einer Erklärung, und ein Plädoyer für mehr Toleranz.

Unbeabsichtigt monadisch

Monaden entwickeln in Java

von Róbert Bräutigam

Funktionale Entwicklung ist wie die dunkle Seite der Macht. Begibt man sich einmal auf diesen Pfad, kontrolliert sie die Gedanken des Entwicklers für immer, scheinbar auch wenn es um *Java*-Code geht. Folgendes beruht auf einer wahren Begebenheit.

Kapitel 3: Die Falle

Es war einmal vor nicht so langer Zeit in unserer Galaxie, in einem durchschnittlichen Büro, in dem an einer *JEE*-Software gearbeitet wurde...

Diese *Middleware*-Software hatte etwa 100 *Web-Services* Richtung *Frontend*, und benutzte Datenbankzugriffe, Web-Service-Schnittstellen und auch andere Verbindungen nach „unten". Die verschiedenen *Backend*-Aufrufe lieferten verschiedene Antwortobjekte zurück, mit eigenen Vererbungshierarchien teilweise unabhängig von einander.

An einem schönen Tag ist eine neue Anforderung eingetroffen: Die meisten Backend-Schnittstellen würden gerne zusätzliche Nachrichten zum Antwortobjekt zurückliefern, was auch die Frontends interessieren würde. Das heißt die Middleware muss die Nachrichten von den Backend-Aufrufen bis zu den Web-Service-Antwortobjekten behalten und immer weiterkopieren durch die verschiedenen Schichten. Die Nachrichten ähneln *HTTP Response Codes*: erfolgreich durchgeführt, Änderung wird später durchgeführt, und so weiter.

Ansatz

Die meisten Service-Methoden haben folgendes Schema:

```
public ÜberweisungsAntwort getTransfers(...) {
  return transformAntwort(backend.getTransfers(...));
}
private ÜberweisungsAntwort transformAntwort(
  BackendÜberweisung backendÜberweisung
) {
  ÜberweisungsAntwort antwort =
    new ÜberweisungsAntwort();
  antwort.setBetrag(backendÜberweisung.getBetrag());
  ...
  return antwort;
}
```

Also ein oder mehrere Backend-Aufrufe werden gemacht, weil die Anfrage- und Antwortobjekte immer umgewandelt werden zwischen den zwei Schichten.

Was wenn die Antwort jetzt zusätzliche Nachrichten enthalten muss? Die einfache Variante wäre alle Backend-Antwortobjekte und alle Middleware-Antwortobjekte noch mit einer Liste von Nachrichten zu erweitern, und die einfach immer in *transformAntwort()* durch zu kopieren. Leider ist diese Methode in jedem Service spezifisch implementiert, da jeder Service einen eigenen Antworttyp definiert. Also müsste man jeden Service anfassen und *transformAntwort()* mit dieser Logik erweitern. Etwa so:

```
private ÜberweisungsAntwort transformAntwort(
  BackendÜberweisung backendÜberweisung
) {
  ÜberweisungsAntwort antwort =
    new ÜberweisungsAntwort();
  antwort.setBetrag(backendÜberweisung.getBetrag());
  ...
  antwort.setNachrichten(
    backendÜberweisung.getNachrichten()
  ); // Nachrichten kopieren
  return antwort;
}
```

Diese Lösung hat aber ein großes Problem: Der *Compiler* kann nicht garantieren, dass alle Stellen wo Backend-Services aufgerufen werden auch modifiziert werden. Und wenn man das doch irgendwie schaffen würde, kann es in neuen Services leicht vergessen werden. Die Frage ist: Wie kann man diese Logik so sicher machen, dass es unmöglich ist es zu vergessen.

Typsicherheit

Anstatt die Antwortobjekte fürs Backend mit einer Liste von Nachrichten zu erweitern, scheint es sicherer zu sein, einen eigenen Typ dafür zu schreiben:

```
public class NachrichtenContainer<T> {
  private T wert;
  private List<Nachricht> nachrichten;

  public NachrichtenContainer(
    T wert, List<Nachricht> nachrichten
  ) {
    ...
  }
}
```

Dieses Objekt kann einen beliebigen Wert (das Antwortobjekt) und eine Liste von Nachrichten beinhalten. Allerdings statt normaler *setter*- und *getter*-Methoden, wird dieses Objekt unveränderbar sein, und wird den Wert nicht so leicht hergeben. Um an den Wert zu kommen, muss der Benutzer dieses Objekts entweder explizit alle Nachrichten wegschmeißen oder sie verarbeiten:

```
public class NachrichtenContainer<T> {
  ...
  public T discardMessages() {
    return wert;
  }
  public T handleMessages(
    Function<List<Nachricht>, Void> verarbeiter
  ) {
    verarbeiter.apply(nachrichten);
    return wert;
  }
}
```

Anstatt ein *interface* für die Verarbeitung von Nachrichten zu definieren, wird hier einfach eine generische Funktion erwartet, die eine Liste von Nachrichten als Parameter nimmt, und nichts (*Void*) zurückliefert. Die Klasse *Function* sieht so aus:

```
public interface Function<A, B> {
  public B apply(A a);
}
```

Die *transformAntwort()*-Methoden, die dieses Objekt verarbeiten müssen, würden dann so aussehen:

```
private ÜberweisungsAntwort transformAntwort(
  NachrichtenContainer<BackendÜberweisung>
  backendÜberweisungsContainer
) {
  ÜberweisungsAntwort antwort =
    new ÜberweisungsAntwort();
  BackendÜberweisung backendÜberweisung =
  backendÜberweisungsContainer.handleMessages(
    antwort.getNachrichtenHandler()
  );
  antwort.setBetrag(backendÜberweisung.getBetrag());
  ...
  return antwort;
}
```

Die ÜberweisungsAntwort *getNachrichtenHandler()*-Methode kann dann die Nachrichten so verarbeiten, dass sie für das Frontend sichtbar werden, z. B. kann es auch einfach die Nachrichten ins ÜberweisungsAntwort-Objekt kopieren wie vorher:

```
public Function<List<Nachricht>,
Void> getNachrichtenHandler() {
  return new Function<List<Nachricht>, Void> {
    public Void apply(List<Nachricht> nachrichten) {
      setNachrichten(nachrichten);
      return null;
    }
  };
}
```

Wenn also alle Backend-Aufrufe die Antwortobjekte in NachrichtenContainer-Objekte verpacken, müssen alle Services explizit die Nachrichten verarbeiten (oder wegwerfen). In jedem Fall wird es nie vergessen, da es sonst einen *Compiler*-Fehler gibt.

Abbildungen

Nicht jede Backend-Antwort kann direkt in Service-Antwort kopiert werden. Was passiert wenn der Aufruf indirekt, durch andere Methoden gemacht wird? Die Nachrichten müssen durch alle Methoden erhalten bleiben. Ein Einfaches Beispiel:

```
public BigDecimal getÜberweisungsLimit(...) {
  return backend.getLimits(...).getÜberweisungsLimit();
}
```

Das Limit wird hier durch einen Backend-Aufruf bestimmt, aber damit gehen alle Nachrichten des Aufrufs verloren (da der Rückgabewert *BigDecimal* ist). Falls die *getLimits()*-Methode einen NachrichtenContainer liefert, kann man *getÜberweisungsLimit()* nicht direkt aufrufen ohne die Nachrichten im Container wegzuwerfen. Es muss daher möglich sein den Wert im Container zu bearbeiten ohne die Nachrichten verarbeiten zu müssen. Dazu dient die *map()*-Methode:

```
public<U> NachrichtenContainer<U> map(
  Function<T, U> mapper
) {
  return new ResultsContainer(
    mapper.apply(wert), nachrichten
);
}
```

Die *map()*-Methode macht einfach ein neues Container-Objekt mit denselben Nachrichten, aber mit einem transformierten Wert. Die *getÜberweisungsLimit()*-Methode kann jetzt die Nachrichten weitergeben:

```
public NachrichtenContainer<BigDecimal>
getÜberweisungsLimit(...) {
  return backend.getLimits(...).map(
    new Function<LimitsAntwort, BigDecimal>() {
      public BigDecimal apply(
        LimitsAntwort antwort
      ) {
        return antwort.getÜberweisungsLimit();
      }
    }
  );
}
```

An dieser Stelle würden *Lambda-Expressions* helfen dies lesbarer zu gestalten (siehe das Java 8-Kapitel).

Mehrere Aufrufe zum Backend

Wenn ein Service mehrere Aufrufe zum Backend macht, müssen standardmäßig alle Nachrichten erhalten werden. Nur mit Abbildungen ist das nicht zu schaffen, wie das nächste Beispiel demonstriert:

```
public NachrichtenContainer
<ÜberweisungsAnlegenAntwort> anlegen(...) {
  NachrichtenContainer<VorprüfungsAntwort>
  prüfungsAntwort = backend.prüfen(...);
  NachrichtenContainer
  <ÜberweisungsAnlegenAntwort>
  anlegenAntwort = backend.anlegen(???);
  return ???;
}
```

Es gibt hier zwei Probleme: Wie wird die Prüfungsantwort ausgepackt, und wie werden die Nachrichten von dieser Prüfung erhalten?

Dazu braucht man noch eine Methode *flatMap()* im NachrichtenContainer:

```
public <U> NachrichtenContainer<U> flatMap(
  Function<T, NachrichtenContainer<U>> mapper
) {
  NachrichtenContainer<U> resultat =
  mapper.apply(wert);
  return new NachrichtenContainer(
    resultat.wert, merge(resultat.nachrichten, nachrichten)
  );
}
```

Sie funktioniert also fast wie *map()*, nur liefert der *mapper* selbst auch einen NachrichtenContainer. Die Nachrichten von beiden Containern werden dann zusammengemischt. Das Resultat wird daher alle Nachrichten beinhalten. Mehrere Backend-Aufrufe kann man dann so machen:

```
public NachrichtenContainer
<ÜberweisungsAnlegenAntwort> anlegen(...) {
  return backend.prüfen(...).flatMap(
    new Function<VorprüfungsAntwort,
    NachrichtenContainer
    <ÜberweisungsAnlegenAntwort>>() {
      public NachrichtenContainer
      <ÜberweisungsAnlegenAntwort>
      apply(VorprüfungsAntwort prüfungsAntwort) {
        return backend.anlegen(prüfungsAntwort);
      }
    }
  );
}
```

KeinException Exception

Es gibt noch ein kleines Problem mit *Exceptions*. Backend-Aufrufe werfen spezielle (aber unterschiedliche) *Exceptions*, aber die *apply()*-Methode in *Function* hat keine *Exception* deklariert, deshalb wird die oben definierte *anlegen()*-Methode so noch nicht kompiliert.

Die *Function*-Klasse muss deshalb noch einen Klassenparameter für *Exceptions* haben:

```
public interface Function<A, B, E extends Exception> {
  public B apply(A a) throws E;
}
```

Die *map()*- und *flatMap()*-Methoden müssen auch angepasst werden (hier nur *map()* aufgeführt):

```
public<U, E extends Exception>
NachrichtenContainer<U> map(
  Function<T, U, E> mapper
) throws E {
  return new NachrichtenContainer(
    mapper.apply(wert), nachrichten
  );
}
```

Wenn eine Methode mit einem *Function*-Parameter keine *Exception* werfen will, kann es *RuntimeException* angeben (das ja nicht deklariert sein muss). Schöner ist es allerdings dafür ein spezielles *KeinException* zu deklarieren:

```
public final class KeinException extends
RuntimeException {
  private KeinException() {
  }
}
```

Da diese Klasse einen privaten Konstruktor hat, gibt es garantiert keine Instanzen die man werfen könnte.

Java 8, The „Ocho“

Es ist umständlich immer eine *Anonyme Function*-Klasse zu schreiben, aber Java 8 eilt zur Rettung. Mit *Lambda*-Ausdrücken kann man die oben erwähnten Beispiele wesentlich einfacher schreiben, Abbildungen z. B. so:

```
public NachrichtenContainer<BigDecimal>
getÜberweisungsLimit(...) {
  return backend.getLimits(...).map(
    antwort -> antwort.getÜberweisungsLimit()
  );
}
```

Fazit

Lösungen aus der funktionalen Welt haben auch in Java eine Lebensberechtigung (wie die hier beschriebene NachrichtenContainer-*Monade*), obwohl sie noch sehr umständlich zu schreiben sind. Java 8 wird diesen Aspekt vielleicht ändern, bis dann werden die in diesem Artikel beschriebenen Lösungsansätze eher ein Kuriosum in Java sein.

Des Programmierers kleine Vergnügen

Weiter so!

von Michael Wiedeking

Gelegentlich hat man es mit Daten zu tun die sowohl paarweise als auch allein auftreten. Speichert man diese etwa in einem *Array* ab, so ergibt sich das Problem, dass man zwar die Stelle *i* bearbeitet, aber das nächste Element an einer geraden oder ungeraden Stelle liegen muss. Wie aber findet man effizient zum nächsten geraden oder ungeraden Index?

Will man exemplarisch die zu *i* nächsthöhere gerade Zahl finden, so sagt die naheliegende Lösung: wenn *i* ungerade ist, dann zähl nur 1 dazu andernfalls 2. Ob eine Zahl ungerade ist, lässt sich dadurch herausfinden, ob das niederwertigste Bit gesetzt ist oder nicht. Ob das Bit gesetzt ist, lässt sich dadurch herausfinden, indem man die Zahl durch ein Bit-weises Und „&“ mit 1 verknüpft und prüft, ob das Ergebnis ungleich 0 ist.

```
boolean ungerade(int i) {
  return (i & 1) != 0;
}
```

Damit lässt sich die nächste Zahl recht einfach finden:

```
int nächsteGeradeZahlGrößerAls(int i) {
  if (ungerade(i)) {
    return i + 1;
  } else {
    return i + 2;
  }
}
```

Man ahnt es schon: Diese Implementierung ist für uns an dieser Stelle nicht akzeptabel. Denn wir können natürlich anstatt der Boole'schen Eigenschaft direkt die ganzzahlige Eigenschaft unserer ungerade-Funktion nutzen. Wir wissen nämlich, dass $(i \,\&\, 1)$ nur dann 1 ist, wenn die Zahl ungerade ist und andernfalls 0. Also können wir damit ohne eine Prüfung die gewünschte Korrektur vornehmen.

```
int nächsteGeradeZahlGrößerAls(int i) {
  return i + 2 – (i & 1);
}
```

Auf diese Weise lassen sich analog auch der gerade Vorgänger, der ungerade Nachfolger und ungerade Vorgänger mit $(i - 2 + (i \,\&\, 1))$, $(i + 1 + (i \,\&\, 1))$ beziehungsweise $(i - 1 - (i \,\&\, 1))$ finden.

Falls übrigens die zu *i* nächste gerade oder ungerade Zahl gefunden werden soll, die größer oder gleich bzw. kleiner oder gleich dem gegebenen *i* ist, so kann das durch eine geeignete Umstellung erreicht werden:

```
int nächsteGeradeZahlGrößerOderGleich(int i) {
  return i + (i & 1);
}
```

Auch hier lassen sich die anderen Varianten leicht finden. Dabei ist nur erwähnenswert, dass für ungerade Zahlen das Ergebnis von $(i \,\&\, 1)$ „negiert“ werden muss, was man natürlich mit $(1 - (i \,\&\, 1))$ erreicht. Damit lassen sich also der gerade „Vorgänger“, der ungerade „Nachfolger“ und ungerade „Vorgänger“ mit $(i - (i \,\&\, 1))$, $(i + 1 - (i \,\&\, 1))$ bzw. $(i - 1 + (i \,\&\, 1))$ bestimmen.

Insgesamt benötigen wir etwa drei Instruktionen für diese Funktionen. Es geht aber noch ein bisschen besser, wenn man sich vor Augen hält, wie gerade bzw. ungerade Zahlen aussehen. Eingangs wurde festgestellt, dass zu einer ungeraden Zahl nur 1, aber zu einer geraden 2 addiert werden muss. Die Lösung nutzte eine Addition mit zwei und einer anschließenden Korrektur. Die Frage ist aber, ob man diese Addition mit 2 nicht eliminieren kann.

Ist *i* ungerade, so genügt es 1 dazuzuzählen: $i + 1$. Was muss man machen, damit *i* ungerade wird? Man muss nur das niederwertigste Bit mit Hilfe des Bit-weisen Oder „|“ setzen. $(i \mid 1)$ ist also ungerade und immer größer oder gleich *i*. Ist *i* ungerade ändert sich nichts, ist *i* gerade wird die Zahl um 1 größer und ungerade. Damit erhalten wir eine Lösung, die nur zwei statt drei Instruktionen benötigt:

```
int nächsteGeradeZahlGrößerAls(int i) {
  return (i | 1) + 1;
}
```

Man sieht gleich, dass man das auch für die nächste ungerade Zahl größer oder gleich i einsetzen kann, denn $(i \mid 1)$ löst ja schon genau dieses Problem.

```
int nächsteUngeradeZahlGrößerOderGleich(int i) {
  return i | 1;
}
```

Der „Vorgänger“ von i, der kleiner oder gleich ist, kann darauf basierend ermittelt werden:

```
int vorherigeUngeradeZahlKleinerOderGleich(int i) {
  return (i – 1) | 1;
}
```

Auch die nächste ungerade Zahl, die sicher größer ist, kann darauf aufbauend implementiert werden.

```
int nächsteUngeradeZahlGrößerAls(int i) {
  return (i + 1) | 1;
}
```

Bei der „Negation“ dieser Fälle für die verbleibenden vier gestaltet es sich ein bisschen schwieriger, weil zu diesem Zweck das niederwertigste Bit gelöscht werden muss. Dazu benötigt man eine Bit-Maske, bei der nur das niederwertigste Bit nicht gesetzt ist und „verundet“ diese mit der gewünschten Zahl. Dieses Bit-Muster erhält man indem man sämtliche Bits der Zahl 1 mit „~“ invertiert. Damit erhält man eine gerade Zahl, die kleiner oder gleich der gegebenen ist.

```
int vorherigeGeradeZahlKleinerOderGleich(int i) {
  return i & ~1;
}
```

Und damit hat man die Basis, um die fehlenden drei Funktionen zu implementieren. Der gerade „Nachfolger“ etwa, der größer oder gleich der gegebenen Zahl ist, kann dadurch bestimmt werden, dass die Zahl zunächst erhöht und anschließend „begradigt“ wird.

```
int nächsteGeradeZahlGrößerOderGleich(int i) {
  return (i + 1) & ~1;
}
```

Der kleinere gerade „Vorgänger“ und der kleinere ungerade „Vorgänger“ lassen sich dementsprechend mit $(i – 1)$ & ~1 respektive $(x$ & ~1$) – 1$ bestimmen.

Damit lässt sich zwar nicht immer die Anzahl der Instruktionen gegenüber dem alten Verfahren reduzieren, aber insgesamt ist ja schon eine Verbesserung eingetreten. Übrigens sieht das ganz anders aus, wenn der Prozessor eine Und-Nicht-Instruktion hat. Dann kann nämlich $(x$ & ~1$)$ in nur einer Instruktion abgehandelt werden, was die maximale Anzahl der Instruktionen tatsächlich auf 2 minimiert.

Deutsch für Informatiker

me <3 #omnom

von Golo Roden

Geeks und Nerds haben eine eigene Sprache, eigenen Humor – letztlich eine ganz eigene Subkultur. Diese ist von Wortspielereien, Katzenfotos und anderen auf Außenstehende seltsam anmutenden Dingen geprägt. Dies ist der Versuch einer Erklärung, und ein Plädoyer für mehr Toleranz.

Kennen Sie Omnom? Nein? Omnom ist ein kleines, grünes, süßes und äußerst bezauberndes Monster, das nur einen einzigen Wunsch hat: Omnom möchte mit Süßigkeiten gefüttert werden – und genau das ist die Aufgabe, die man als Spieler in dem äußerst niedlichen Spiel „Cut the Rope" [1] und dessen Nachfolgern zu erfüllen hat.

Um seinem Wunsch Nachdruck zu verleihen, weist Omnom gelegentlich mit Gesten und großem Augengeklimper darauf hin, dass er wartet. In Verbindung mit der fröhlichen Hintergrundmusik entführt Omnom den Spieler in „Cut the Rope" in eine heile, bonbonfarbene Welt, die man äußerst treffend mit dem Wort „niedlich" beschreiben kann.

Nun klingt das Wort „niedlich" unter Umständen abwertend, weshalb die *Geeks* und *Nerds* unter den Informatikern gerne auf ein anderes, in diesem Fall englischsprachiges Wort ausweichen, das eindeutig positiv belegt ist: Omnom ist „cute".

Om-nom-nom-nom!

Sein Name geht dabei auf die Figur des Krümelmonsters aus der Sesamstraße zurück, das das Auffressen seiner Kekse akustisch stets mit einem herzhaften „om-nom-nom-nom" unterlegt.

Auf YouTube gibt es ein entsprechendes Video, das ebendiesen Begriff und seinen Ursprung thematisiert [2]. In diesem Video fällt jedoch eine Besonderheit auf: Das Krümelmonster spricht grammatikalisch nicht ganz korrekt. Auf die Frage, ob es wahr sei, dass es seinen Kekskonsum eingeschränkt habe, antwortet es beispielsweise: „That not entirely true [...], me love cookies!"

Diese verkürzte und vereinfachte Form zu sprechen, wurde im Internet anderweitig aufgegriffen: Der Satz „I can has cheezburger" ist die vermutlich bekannteste Variante [3]. Jeder Informatiker, der sich als Geek und Nerd sieht, kennt – und schätzt – diesen Satz.

Das hat jedoch nichts mit der vielbeschworenen Verkümmerung der Sprache zu tun, sondern ist lediglich Ausdruck einer Subkultur: Natürlich sind Informatiker von ihrer Intelligenz her problemlos in der Lage, den Satz „May I have a cheeseburger?" grammatikalisch, orthographisch und auch in Bezug auf die enthaltene Höflichkeit korrekt zu formulieren – nur geht dann der Witz verloren.

Dies kann man ablehnen oder befürworten, dies kann man „cute" finden oder nicht. Beides ändert aber nichts daran, dass sich diese Art zu sprechen – genauso wie bestimmte Begriffe – im Web etabliert hat, und von Millionen von Menschen tagtäglich genutzt wird. Jede Szene hat ihre eigene Sprache, warum also sollte es bei den Informatikern anders sein?

Memes sind die Gene der Web-Kultur

Bei all diesen im Web geborenen Dingen spricht man von *Memes* [4], wobei ein Meme laut Wikipedia zunächst nichts anderes ist als ein „Internet-Phänomen", das sich in Form eines Links, eines Bildes, einer Audio- oder Videodatei viral über das Web verbreitet.

Das erklärt jedoch nicht die Herkunft des Begriffs Meme. Hierzu muss man gezielt unter „Mem" [5] nachschlagen. Dort erfährt man dann, dass es sich um ein Kunstwort handelt, das einen einzelnen Bewusst-

seinsinhalt beschreibt, den man durch Kommunikation weitergeben und verbreiten kann. Dies entspricht der genetischen Vererbung, übertragen auf das Konzept von Gedanken und Ideen: Während Gene der biologischen Evolution dienen, bewirken Memes eine soziokulturelle Entwicklung und beeinflussen nicht den Einzelnen sondern die Gesellschaft.

Neben den bereits genannten gibt es zahlreiche weitere Beispiele für Memes: Katzenfotos, die mit grammatikalisch nicht ganz korrekten Texten versehen wurden, zählen beispielsweise dazu. Da diese Fotos die Katzen in meist mehr oder minder lustigen Szenen zeigen, hat sich in Anlehnung an den Chat-Ausdruck *LOL* (Laughing out loud) der Begriff *Lolcat* eingebürgert [6].

„I can has cheezburger" zählt hier ebenso dazu, wie beispielsweise Grumpy Cat, eine stets mürrisch guckende Katze [7], die es zu zahllosen Bildern gebracht hat [8].

Wie verbreitet und auch wichtig Katzen im Web sind, zeigt sich daran, dass als Argument für das iPhone 5 im Vergleich zu dessen Vorgängern verschiedentlich genannt wurde, dass Katzenbilder nun auf Grund des größeren Displays deutlich besser betrachtet werden könnten [9].

Der Einfluss sozialer Medien

Das Medium, um über all diese Dinge zu diskutieren, ist naturgemäß das Web. Eine besondere Rolle spielen hierbei die sozialen Netzwerke wie beispielsweise Twitter, wo man Tweets mit Hilfe von sogenannten *Hashtags* verschlagworten kann. Auch hierbei entstehen gewisse Begriffe, die ihrerseits zum Meme werden.

Als Beispiel sei „#epicfail" genannt, was einen im negativen Sinne besonders eindrucksvollen Fehler beschreibt, während ein positives Ergebnis mit verhältnismäßig geringem Aufwand problemlos zu erreichen gewesen wäre. Für Beispiele genügt es, die Twitter-Suche zu bemühen [10]: Obgleich diese Suche lediglich in den aktuellen Tweets sucht, wird man stets fündig.

Fazit

Memes schrecken oftmals zunächst ab. Zum einen, weil sie von Außenstehenden häufig nicht verstanden werden, zum anderen, weil der Humor durchaus ein wenig eigen ist. Sheldon Cooper [11], eine der Hauptfiguren aus der Fernsehserie The Big Bang Theory, beweist dies auf überspitzte Art. Von vielen nicht verstanden und als Sonderling abgetan, wird er von vielen Geeks und Nerds geliebt, weil er intelligent, wortgewandt und einer von ihnen ist.

Wie so oft mit Subkulturen kann man die Faszination, die von ihnen ausgeht, nicht erklären – man muss sie erleben. Dazu muss man sich jedoch auf sie einlassen und sich vorurteilsfrei mit ihnen befassen.

Das größte Hindernis hierbei ist nicht das (zunächst) fehlende Verständnis für Omnom, Grumpy Cat und Sheldon Cooper, sondern das eigene Ego. Es kostet den Sprung über den eigenen Schatten, sich als *Newbie* zu outen und zuzugeben, dass man nicht versteht, worum es geht. Doch wer dies offen und ehrlich macht, wird auch offen und ehrlich empfangen. Eigentlich ist es nicht schwer.

Wer das jedoch nicht macht, und den bequemen Weg geht und bei seinen Vorurteilen bleibt, dem sei gesagt: *#epicfail*.

PS: Der Begriff <3 im Titel dieses Artikels ist ein *Emoticon*, wie beispielsweise auch die verschiedenen Smileys. Dreht man den Kopf um 90° nach rechts, erkennt man ein Herz. Der Titel drückt also aus, dass der Autor dieses Artikels zu jenen gehört, die Omnom „cute" finden.

Referenzen

[1] cuttherope
http://www.cuttherope.ie

[2] YouTube *Sesame Street & The Origin of Om nom nom nom*
http://www.youtube.com/watch?v=Cqz9ZXUoUcE

[3] Wikipedia *I can has cheesburger*
https://en.wikipedia.org/wiki/I_Can_Has_Cheezburger%3F

[4] Wikipedia *Internet-Phänomen*
http://de.wikipedia.org/wiki/Internet-Ph%C3%A4nomen

[5] Wikipedia *Mem*
http://de.wikipedia.org/wiki/Mem

[6] Wikipedia *Lolcat*
http://de.wikipedia.org/wiki/Lolcat

[7] Wikipedia *Grumpy Cat*
http://de.wikipedia.org/wiki/Grumpy_Cat

[8] *I don't like morning people…*
http://24.media.tumblr.com/tumblr_meixnsPX1k1rha7cbo1_1280.jpg

[9] itler *Der einzige Grund warum man ein iPhone 5 benötigt*
http://itler.net/der-einzige-grund-warum-man-ein-iphone-5-benoetigt

[10] Twitter *Ergebnisse für #epicfail*
https://twitter.com/search/realtime?q=%23epicfail&src=typd

[11] Wikipedia *Sheldon Cooper*
http://en.wikipedia.org/wiki/Sheldon_Cooper

Golo Roden (www.thenativeweb.io) ist Gründer und technologischer Visionär der „the native web UG" (haftungsbeschränkt), eines auf native Web-Technologien spezialisierten Unternehmens. Für die Entwicklung moderner Web-Anwendungen bevorzugt er *JavaScript* und *Node.js* und hat mit „Node.js & Co." das erste deutschsprachige Buch zu diesem Thema geschrieben. Darüber hinaus ist er journalistisch für verschiedene Fachmagazine und als Referent und Content Manager für Konferenzen im In- und Ausland tätig. Für sein qualitativ hochwertiges Engagement in der Community wurde Golo von Microsoft zweifach als Most Valuable Professional (MVP) für C# ausgezeichnet.

Kaffeesatz

Entschleunigung

von MichaelWiedeking

Mit der Einführung des Internets sind bei der Kommunikation Geschwindigkeiten erreicht worden, die ihresgleichen suchen. Wobei sich natürlich die Frage stellt, ob dies wirklich immer für den Inhalt – quantitativ wie qualitativ – förderlich ist. Und, ob dabei nicht auch etwas auf der Strecke bleibt.

Am 25. Februar 1795 schrieb Heinrich von Kleist in einem Brief an seine Schwester Ulrike:

> *„Ein Geschenk mit so außerordentlichen Aufopferungen von Seiten der Geberin verknüpft, als Deine für mich gestrickte Weste, macht natürlich auf das Herz des Empfängers einen außerordentlichen Eindruck.“*

Eigentlich grenzt es an ein kleines Wunder, dass überhaupt Briefe von ihm erhalten geblieben sind, die er vor über 200 Jahren geschrieben hat. Wahrscheinlich ist es dem Umstand zu verdanken, dass Briefe etwas ganz Besonderes sind und es verdienen aufgehoben zu werden. Die Sammlung von Kleists Briefen an die verschiedensten Adressaten zeigt, dass dies tatsächlich von vielen Empfängern so gesehen wird.

Literarisch nicht ganz so wertvoll sind die Zettelchen, die ich in der Schule bekommen habe. Ich habe sie aufgehoben und so besitze ich eine Schachtel voll mit kleinen bekritzelten Papierstückchen, fast allen, die ich je bekommen habe. Für die Nachwelt werden sie – soweit ich das im Moment beurteilen kann – nicht besonders interessant sein, da sich unter meinen ehemaligen Mitschülern im Moment niemand befindet, dessen Ruhm eine Veröffentlichung rechtfertigen würde.

Sämtliche Briefe, die ich im Laufe der Zeit bekommen habe, finden auch in einer eigenen Kiste Platz. Sie sind literarisch auch schon deutlich wertvoller als die Zettelchen, enthalten sie doch auch vernünftige Sätze, beschreiben vollständig Ereignisse und erzählen Geschichten vergangener Zeiten. Auch das eine oder andere Fax ist dabei, von denen die ältesten allerdings kaum noch zu entziffern sind. Sie sind Opfer ihres Materials geworden: Thermopapier.

Mit dem Eintritt ins E-Zeitalter und den damit einhergehenden virtuellen Medien, hat sich die Situation nicht wirklich verbessert. Der schwerwiegendste Nachteil ist sicher das fehlende haptische Erlebnis. Natürlich kann man eine E-Mail auch ausdrucken, aber das ist irgendwie nicht dasselbe. Und verloren gehen können die virtuellen Briefe auch. So hat bei mir vor einigen Jahren ein Technologiewechsel stattgefunden, dem praktisch alle E-Mails von der Zeit vor Ende 2004 zum Opfer gefallen sind. Schade eigentlich.

Zugegeben, E-Mails können auch ein bisschen entschädigen, kann ich doch lesen, was ich geschrieben habe. Eine Schulfreundin von mir hat mir aber einmal erzählt, dass sie sich tatsächlich schon in der Prä-E-Ära Kopien der von ihr geschriebenen Briefe gemacht hat. Dabei darf man nicht vergessen, dass wir miterleben durften, wie das erschwingliche Kopieren überhaupt erst möglich wurde.

Was mich allerdings am meisten stört, ist der Umstand, dass ich eine E-Mail sofort zugestellt bekommen kann. Es mag gelegentlich ganz nützlich sein, dass eine Bestellung heute noch oder sogar gleich ankommt, egal wie weit weg man die E-Mail verschickt. Aber bei einem „echten“ Brief darf das einfach nicht so schnell gehen. Der braucht Zeit. Nicht nur beim Reisen, sondern auch beim Reifen.

Neulich habe ich mich wieder einmal gegen eine E-Mail entschieden und einen Brief verschickt. Vom Entschluss an, den Brief zu schreiben, bis zu dem Zeitpunkt, als er in den Briefkasten eingeworfen wurde, ist fast ein ganzer Tag vergangen. Das war auch gut so, denn Dinge, die gut sein sollen, brauchen ihre Zeit. Die Formulierungen können durchdachter sein und der Inhalt besser strukturiert. Insgesamt kann das Lesevergnügen schon alleine dadurch um ein Vielfaches gesteigert werden, dass man einen Brief erst einmal relativ umständlich aufmachen muss.

Ich habe übrigens auch einen Brief zurück bekommen. Es hat genau eine Woche gedauert, bis ich die Antwort auf meinem Tisch liegen hatte. Völlig losgelöst davon, worum es in dem Briefwechsel ging, kann ich Ihnen versichern, dass es eine tolle Woche war, eine mit Spannung und Vorfreude – wenn Sie wissen, was ich meine.

User Groups

Fehlt eine User Group? Sind Kontaktdaten falsch? Dann geben Sie uns doch bitte Bescheid.

Bookware
Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de

Java User Groups

Deutschland

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

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

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

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

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
http://www.ruhrjug.de
Kontakt: Herr Heiko Sippel (heiko.sippel@ruhrjug.de)

JUGF
Java User Group Frankfurt
http://www.jugf.de
Kontakt: Herr Alexander Culum
(alexander.culum@web.de)

JUG Deutschland e.V.
Java User Group Deutschland e.V.
c/o asc-Dienstleistungs GmbH
http://www.java.de (office@java.de)

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

JUG Karlsruhe
Java User Group Karlsruhe
Universität Karlsruhe, Gebäude 50.34
http://jug-karlsruhe.de
jug-karlsruhe@gmail.com

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

jugm
Java User Group München
http://www.jugm.de
Kontakt: Herr Andreas Haug (ah@jugm.de)

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

JUG MeNue
Java User Group der Metropolregion Nürnberg
c/o MATHEMA Software GmbH
Henkestraße 91, 91052 Erlangen
http://www.jug-n.de
Kontakt: Frau Alexandra Specht
(alexandra.specht@jug-n.de)

JUG Ostfalen
Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
http://www.jug-ostfalen.de
Kontakt: Uwe Sauerbrei (info@jug-ostfalen.de)

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

Schweiz

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

.NET User Groups

Deutschland

.NET User Group OWL
http://www.gedoplan.de/cms/gedoplan/ak/ms_net
%o GEDOPLAN GmbH

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

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

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

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

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

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

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

.NET User Group Leipzig
http://www.dotnet-leipzig.de
Kontakt: Herr Alexander Groß (agross@dotnet-leipzig.de)
Herr Torsten Weber (tweber@dotnet-leipzig.de)

.NET Developers Group München
http://www.munichdot.net
Kontakt: Hardy Erlinger (hardy.erlinger@hotmail.com)

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

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

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

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

Österreich

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

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

Software Craftsmanship Communities

Deutschland

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

MATHEMA

meet the experts of enterprise infrastructure

Neues in Java 7

Die nächste Java Generation
19. – 20. August 2013, 6. – 7. November 2013,
835,– € (zzgl. 19 % MwSt.)

Programmierung mit Java

Einführung in die Java-Technologie und die Programmiersprache Java
26. – 30. August 2013, 11. – 15. November 2013,
1.645,– € (zzgl. 19 % MwSt.)

iPhone Programmierung

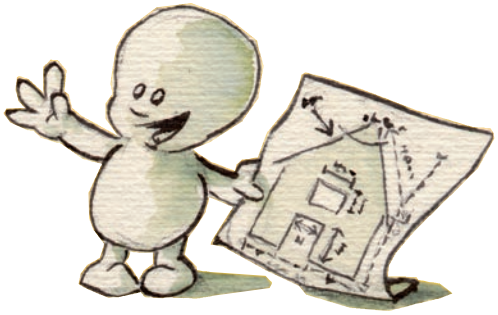
Mobile Anwendungen für das Apple iPhone
16. – 18. September 2013, 25. – 27. November 2013,
1.180,– € (zzgl. 19 % MwSt.)

Enterprise JavaBeans (EJB)

Enterprise JavaBeans im Detail
7. – 11. Oktober 2013, 1.870,– € (zzgl. 19 % MwSt.)

Spring Framework

JavaEE ganz ohne EJB
18. – 20. November 2013, 1.315,– € (zzgl. 19 % MwSt.)



Lesen bildet. **Training macht fit.**

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

MATHEMA

Software-Entwickler (m/w)
Software-Architekt (m/w)

Arbeiten Sie gerne selbstständig, motiviert und im Team? Haben Sie gesunden Ehrgeiz und Lust, Verantwortung zu übernehmen?

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

Wenn Sie ihr Know-how gerne auch als Trainer oder Coach weitergeben möchten, Sie über Berufserfahrung mit verteilten Systemen verfügen und Ihnen Komponenten- und Objektorientierung im .Net- oder JEE-Umfeld vertraut sind, dann lernen Sie uns doch kennen.

Wir freuen uns auf Ihre Bewerbung!

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

Herbstcampus

Wissenstransfer par excellence

2.– 5. September 2013 in Nürnberg

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

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

Offline - na und?
Strategien für offline-fähige Applikationen in HTML 5

JavaScript on Steroids
Eine Einführung in TypeScript

NoSQL hoch drei
3 NoSQL Datenbanken in 70 Minuten

Fundamental
Neuheiten in der Base Class Library unter .NET 4.5

Lambdas
Funktionale Programmierung in Java mit Lambdas

Schwarze Acht
Neuerungen im JDK 8

Ausgeswingt
Oberflächengestaltung mit JavaFX 2.0

Spielen mit Bauklötzen
Composite Component Tuning mit HTML 5 und JavaScript

Unter Beobachtung
Reaktive Programmierung auf der JVM

Weltumspannend
Einsatz des TFS in heterogenen Umgebungen

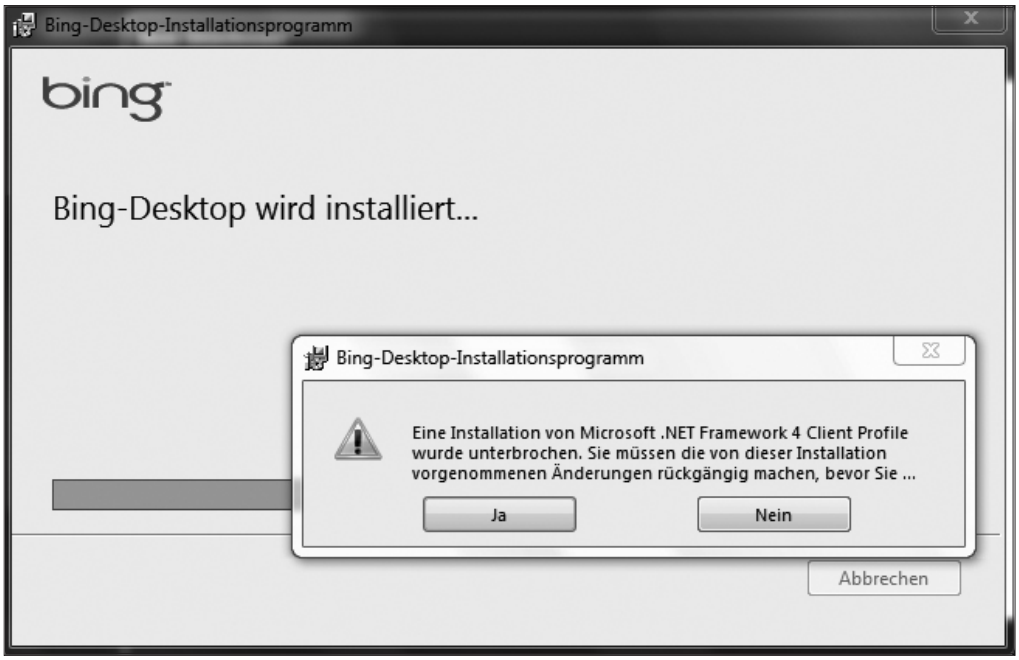
Willkommen in Babylon
Der polyglotte Architekt

u.v.m.

www.herbstcampus.de

veranstaltet von: KaffeeKlatsch Bookware
unterstützt durch: MATHEMA Capgemini
open source PRESS Galileo Computing O'REILLY dpunkt.verlag HANSER

Das Allerletzte



Dies ist kein Scherz!
Diese Fehlermeldung 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 im August.

Herbstcampus

Wissenstransfer par excellence

2.– 5. September 2013

in Nürnberg