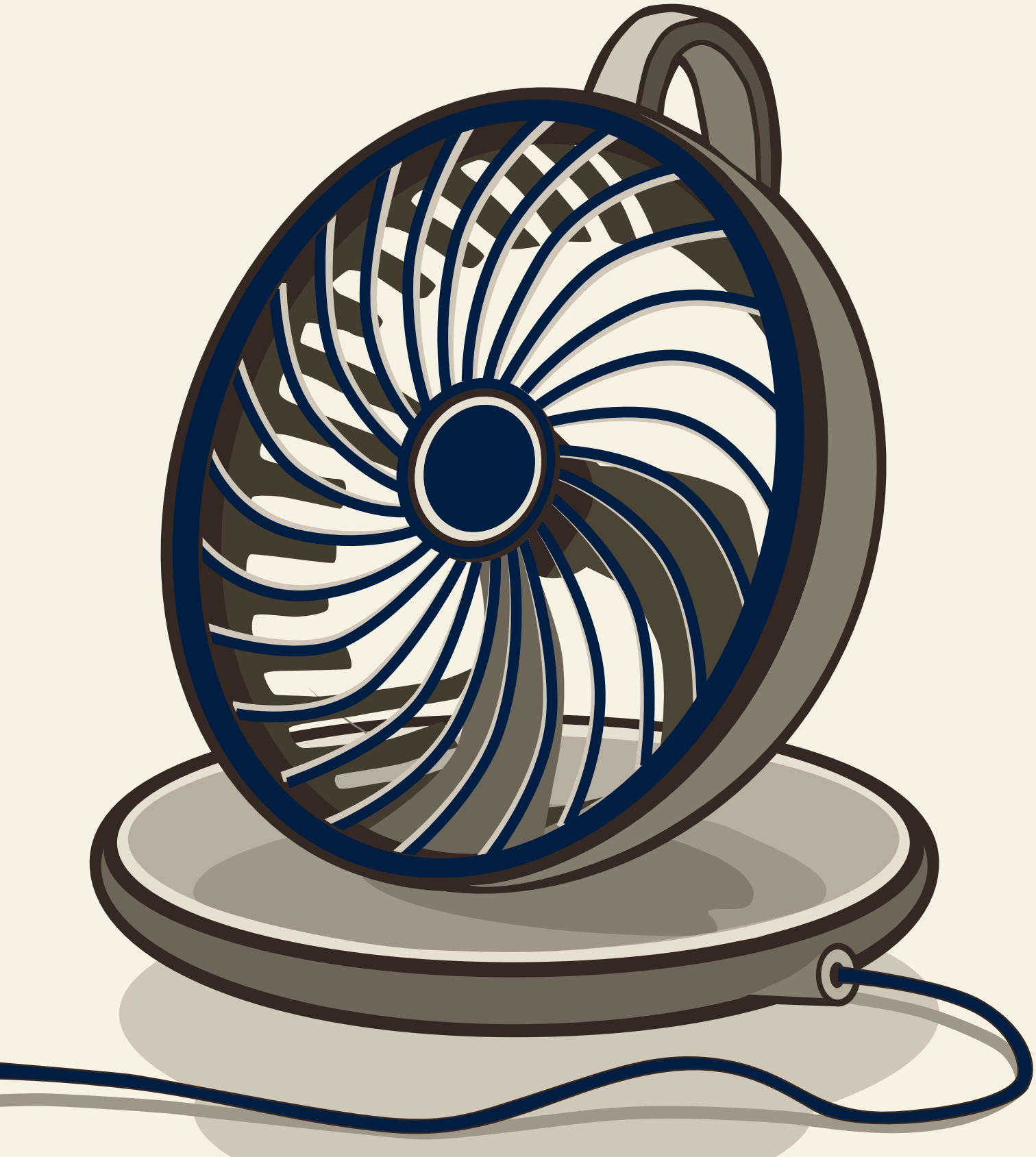
KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

07/2016

Jahrgang 9



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

Transparente Kommunikation

Rattfratz, Glutexo, Traumato: Sind Ihnen diese Begriffe geläufig? Wenn ja, dann gehören Sie wahrscheinlich auch zu der Gattung der Pokemon-Go-Spieler. Falls nicht, dann werden Sie aber womöglich schon einer Personengruppe begegnet sein, die völlig versunken auf Ihr Smartphone starrt und versucht diese digitalen Wesen einzufangen. Das Augmented-Reality-Spiel Pokemon Go hat weltweit ein enormes Spielfieber entfacht, aber den Datenschutz dagegen nicht allzu ernst genommen bzw. die Nutzer nicht wirklich in Kenntnis gesetzt. Verbraucherschützer beanstandeten bislang eine Vielzahl an Datenschutzlücken. So kann das Entwicklerstudio zum Beispiel personenbezogene Daten sammeln und an private Dritte zur Analyse und Auswertung weiterleiten. Den Pokemon-Spieler scheint das jedoch per se nicht zu stören.

Warum ich dieses Beispiel anbringe, ist im Folgenden begründet: Wir möchten unsere Maßnahmen transparent kommunizieren. In den vergangenen Wochen haben wir uns mit der Software von Online-Newsletter-Anbietern auseinandergesetzt. Denn künftig möchten wir den KaffeeKlatsch-Newsletter über solch einen Anbieter versenden und dabei ist der Datenschutz natürlich ein zentrales Thema. Die Gründe für einen Wechsel liegen für uns auf der Hand:

1. automatisierte Aktualisierung der Newsletter-Daten,
2. globaler Zugriff zu jeder Zeit,
3. Whitelisting bei allen großen, deutschen Mail-Anbietern,
4. Double-Opt-In-Verfahren.

Wir möchten dadurch die Erstellung und den Versand der Newsletter so einfach, aber auch so sicher wie möglich umsetzen und müssen im Zuge eines Wechsels Ihre vorhandenen Daten, Name und E-Mail-Adresse, bei einem Newsletter-Anbieter in einer Online-Liste anlegen. Wir haben uns für das Unternehmen rapidmail entschieden, da es unter anderem über ein zertifiziertes Versandsystem verfügt, sich dessen Server in Deutschland befinden und Ihre Daten damit in Deutschland bleiben.

Wir denken, dass es auch in Ihrem Interesse ist, Sie darüber zu informieren. Sollten Sie dennoch diesem Vorhaben widersprechen wollen, was wir sehr bedauern würden, da wir Sie gerne als Abonnenten behalten möchten, so senden Sie uns bis zum 10. August 2016 eine E-Mail mit Ihrem Widerspruch an *redaktion@bookware.de*.

Kommen wir nun aber zum Inhalt dieser Ausgabe, den wir für Sie vorbereitet haben. Tim Bourguignon lüftet „*Das Geheimnis des Einhorn-Entwicklers*“ (Seite 6), Robert Bräutigam schreibt über den *Code-Smell der Setter-/Getter-Methoden in Java* (Seite 12) und in der Kolumne „*Des Programmierers kleine Vergnügen*“ (Seite 15) lesen Sie dieses Mal etwas über die Umwandlung von ASCII-Ziffern.

Ich wünsche Ihnen wieder viel Spaß mit dem KaffeeKlatsch

Ihr Oliver Klosa
Chefredakteur

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 Herrn Oliver Klosa via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-0 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 Oliver Klosa via E-Mail unter redaktion@bookware.de oder telefonisch unter 09131/8903-0.

Copyright

Das Copyright des KaffeeKlatsch liegt vollständig bei der Bookware. Wir gestatten die Übernahme des KaffeeKlatsch in Datenbestände, wenn sie ausschließlich privaten Zwecken dienen. Das auszugsweise Kopieren und Archivieren zu gewerblichen Zwecken ohne unsere schriftliche Genehmigung ist nicht gestattet.

Sie dürfen jedoch die unveränderte PDF-Datei gelegentlich und unentgeltlich zu Bildungs- und Forschungszwecken an Interessenten verschicken. Sollten diese allerdings ein dauerhaftes Interesse am KaffeeKlatsch haben, so möchten wir diese herzlich dazu einladen, das Magazin direkt von uns zu beziehen. Ein regelmäßiger Versand soll nur über uns erfolgen.

Bei entsprechenden Fragen wenden Sie sich bitte per E-Mail an copyright@bookware.de.

Impressum

KaffeeKlatsch Jahrgang 9, Nummer 7, Juli 2016
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 03-99
E-Mail: redaktion@bookware.de
Internet: www.bookware.de

Herausgeber/Redakteur: Oliver Klosa
Anzeigen: Oliver Klosa
Grafik: Nicole Delong-Buchanan

Inhalt

Editorial 3

Beitragsinfo 4

Inhalt 5

User Groups 16

Werbung 18

Das Allerletzte 19

Artikel

Das Geheimnis des Einhorn-Entwicklers

Eine Schatzkarte für die eigene Entwicklerreise 6

Hin und Zurück

Setters/Getters vs. OOP 12

Kolumnen

Mutabor

Des Programmierers kleine Vergnügen 15

Das Geheimnis des Einhorn-Entwicklers

Eine Schatzkarte für die eigene Entwicklerreise 6
von Tim Bourguignon

Wie sieht die nächste Generation an Software-Entwicklern aus? Wenn wir den Gerüchten, den Geschichten, dem Grundrauschen im Internet und den Eigenschaften, wonach Kunden suchen, glauben sollten, dann ist der Neuling exzellent, billig, „T-Shaped“, agil im Kopf und bereit, sowohl für große als auch für kleine Firmen zu arbeiten. Diese Fabeltiere existieren! Aber die Mehrheit unserer künftigen Ex-Studenten weiß nicht, was wir wirklich brauchen. Haben sie schuld? Haben wir unsere Erwartungen klar ausgedrückt? Haben wir festgehalten, wie wir sie in ihrer Entwicklung unterstützen wollen? In den meisten Fällen behaupte ich: nein!

Hin und Zurück

Setters/Getters vs. OOP
von Róbert Bräutigam 12

Setter-/Getter-Methoden in Java sind so verbreitet, dass man sie in fast jeder Klasse wiederfindet. Manche Entwickler generieren sie gar automatisch. Wenn man es jedoch genauer betrachtet, widersprechen solche Methoden grundsätzlich der objektorientierten Programmierung und sind nichts anderes als ein „Code-Smell“.

Mutabor

Des Programmierers kleine Vergnügen 15
von Michael Wiedeking

Was nutzt es einem, eine hexadezimale ASCII-Ziffer in seinen Wert umwandeln zu können, wenn man (wie etwa der Kalif Storch) nicht weiß, wie man von dort wieder zurück kommt.

Das Geheimnis des Einhorn-Entwicklers

Eine Schatzkarte für die eigene Entwicklerreise

von Tim Bourguignon

Wie sieht die nächste Generation an Software-Entwicklern aus? Wenn wir den Gerüchten, den Geschichten, dem Grundrauschen im Internet und den Eigenschaften, wonach Kunden suchen, glauben sollten, dann ist der Neuling exzellent, billig, „T-Shaped", agil im Kopf und bereit, sowohl für große als auch für kleine Firmen zu arbeiten. Diese Fabeltiere existieren! Aber die Mehrheit unserer künftigen Ex-Studenten weiß nicht, was wir wirklich brauchen. Haben sie schuld? Haben wir unsere Erwartungen klar ausgedrückt? Haben wir festgehalten, wie wir sie in ihrer Entwicklung unterstützen wollen? In den meisten Fällen behaupte ich: nein!

Das typische Entwicklerprofil reicht nicht.

Darf diese Hypothese überhaupt formuliert werden? In den letzten Jahren habe ich in meinem Berufsalltag mehrere hunderte Kandidaten-Profile durchgesehen und Vorstellungsgespräche geführt. Immer wieder bin ich zu der Erkenntnis gekommen, dass sehr wenige Kandidaten meine Erwartungen erfüllen. Liegt es an mir und ich erwarte zu viel? Es könnte sein. Aber zwei unterschiedliche Quellen scheinen mir recht zu geben.

In einem Blogeintrag [1] erklärt Chad Fowler, Autor des Buches *The passionate programmer*, folgendes:

> *There's a person I want to work with. I can't find this person. I've literally searched the world, and I can hardly find a trace. I'm not talking about someone specific. In fact, that's the problem. I'm talking about a set of traits and an attitude which is more scarce than I realized until recently...*

Danach listet er Skills auf, die er als wünschenswert betrachtet und meistens nicht findet.

In einer Studie der „Association of American Colleges and Universities" (AACU) von 2015 [2] liegen Studenten und Firmen bei 17 Fragen bezüglich der „Vorbereitung auf das professionelle Leben" ziemlich weit auseinander. Jeder zweite „College Graduate" geht davon aus, gut vorbereitet zu sein (bei 11 von den 17 Fragen antworteten mehr als 55 Prozent der Befragten positiv). Jedoch nur jede dritte Firma bestätigte das (über alle Fragen hinaus wurden lediglich 37 Prozent der Fragen positiv beantwortet). Da es sich dabei zum einen um eine US-Amerikanische Studie und zum anderen um keinen software-spezifischen Hintergrund handelt, lassen sich die Ergebnisse nicht ohne Weiteres auf die EU bzw. die Software-Entwickler übertragen. Dennoch zeigt sie entsprechende Tendenzen auf.

Kann das „Uni-Bashing" jetzt beginnen? Nein, wie schon erwähnt, liegt es wohl eher daran, dass die Software-Unternehmen die Erwartungen und Wünsche nicht klar genug ausgedrückt haben. Software-Entwickler sind anscheinend immer noch introvertierte Menschen.

Was ist aber nun unter einem Einhorn-Entwickler zu verstehen?

Der Einhorn-Entwickler ist jemand, für den die technische Qualität nötig, aber nicht ausreichend ist. Er ist

jemand, der pragmatisch, zielorientiert, kommunikativ, neugierig und experimentierfreudig ist und der es liebt zu forschen. Er ist jemand, der die Professionalität mit Löffeln gegessen hat und der weiß, wo er hin will. Und das ist nur die Spitze des Einhorns.

Wenn man die Möglichkeit hätte, Neulinge zu unterstützen, welche Karrieretipps und -tricks sollte man ihnen geben? Wie ließe sich den künftigen Ex-Studenten am besten helfen? Was sind die wichtigen Aspekte eines Entwicklerprofils?

Die Ideen und Erwartungen des Autors werden in die folgenden drei Kategorien eingeordnet:

- Technik
- Zusammenarbeit
- Die eigene Persönlichkeit

Alle Facetten anzugehen, würde den Umfang dieses Artikels sprengen. Deshalb werden nur die wesentlichsten Aspekte – die eigenen Erkenntnisse des Autors – beschrieben.

Technik

Technisch exzellent zu sein, ist nach wie vor die wichtigste Eigenschaft eines Entwickler und muss insofern die erste Priorität bleiben. Klar ist aber auch, aufgrund des aktuellen Mangels an fähigen Arbeitskräften in der IT-Branche, dass es jedem erlaubt, der eine *for*-Schleife schreiben und *StackOverflow.com* halbwegs richtig bedienen kann, seine Miete zu zahlen. Scott Hanselmann hat für diese Entwickler den Begriff „*Dunkle-Materie-Entwickler*“ [3] geprägt. Wir haben zwar bewiesen, dass es Dunkle Materie geben muss, aber wir sehen sie nie. Dunkle-Materie-Entwickler publizieren nichts im Internet, sind vielleicht über VB6 oder COBOL nicht hinausgekommen und halten auch keine Vorträge auf Konferenzen. Rein vom Arbeitsaufwand her muss es diese Entwickler dennoch geben. Sie schaffen Arbeit weg und halten Teile unserer Gesellschaft am Leben. In diesem Personenkreis sind aber nicht diejenigen Mitarbeiter, die für die eigenen Kunden geeignet sind.

Wenn man eine Bewerbungsmappe in der Hand hält, mit einer ausführlichen Beschreibung des Technologie-Stacks – von der Domäne bis zur ausgewählten Methodik –, welche Art von Entwickler wird dann gesucht? Sicherlich zuerst diejenigen, deren Profile perfekt passen, oder? Das Buzzword-Bingo sollte stimmen. Dann wird man auf die Einstellungsjagd gehen und die Kriterien, die gegebenenfalls übergreifend sein könnten, analysieren. Diese „Soft Skills“ sind die wichtigsten Merkmale, die zu einem „Deal-Maker“ oder „Deal-Breaker“ führen. Natürlich nur solange der Technologie-Stack passt. Im Bereich der Technologie gibt es drei wesentliche Aspekte:

- Clean Code
- Qualitätsfokus
- Werkzeuge meistern

Clean Code, wie im Namensgeber-Buch von Robert „Uncle Bob“ Martin (2009) beschrieben [4], ist die vielleicht aktuell beste gemeinsame Definition von Code-Qualität, die wir haben. Uncle Bob legt einen klaren Fokus auf Lesbarkeit, organisches Wachstum und Einfachheit. Es ist ein lesenswertes Buch, egal welches Niveau man besitzt, da es immer interessante Aspekte zu entdecken gibt. Denn ohne Clean Code ist die Weiterentwicklungsgeschwindigkeit eines Projekts gefährdet. Des Weiteren ist ohne Clean Code die Zusammenarbeit schwerer und dadurch die Harmonie des Teams in Gefahr. Stellt man im Vorstellungsgespräch die Frage, was die Kandidaten mit dem Begriff Clean-Code verbinden, werden die Antworten ab und an ernüchternd sein und sind manchmal ein K.-o.-Kriterium.

Ein Prinzip von *eXtreme*-Programming (XP) ist der massive Einsatz von (automatisierten) Tests. Ob dabei von TDD (Test Driven Development), BDD (Behavior Driven Development) oder sonst irgendwelchen xDD-Varianten geredet wird, spielt zunächst einmal eine untergeordnete Rolle, solange die „Qualitätsbesorgnis“ vorhanden ist. „Test First“ hat sicher Vorteile gegenüber „Debug After“, aber es bringt auch eine gewisse Kritik-Historie mit sich. Es genügt nur eine extrem hohe Qualität, um ausreichend Flexibilität für ein Projekt einzukaufen. Diese Qualität muss natürlich bewiesen (also testbasiert) und leicht wiederholbar (also automatisiert) sein. Dieses „Mindset“ ist unersetzlich, wenn der Druck im Projekt steigt. Denn dann kehrt der Mensch zu seinen alten Routinen zurück. Und beinhaltet diese Routine kein Qualitätsbewusstsein, ist das Projekt sofort gefährdet.

Entwicklerwerkzeuge zu beherrschen, rundet den technischen Aspekt ab. Wer Mann-Monate oder sogar Mann-Jahre mit einem Werkzeug verbracht hat, muss in diesem Bereich eine Expertise aufgebaut haben und das Werkzeug zu jeder Zeit meistern. Was sind aber diese Werkzeuge?

Das sind erst einmal die Programmiersprachen. Darunter fällt auch die Beherrschung von mehreren Sprach-Paradigmen: prozedural, objektorientiert und funktional. Dabei stellen sich die Fragen, wie gut die Kandidaten Code lesen können, ob sie nur die Programmiersprache

verstehen oder auch gleich die dahinterliegende Philosophie. Erkennen es die Kandidaten in einer Code-Basis, wenn die Idiome einer Sprache nicht richtig verstanden wurden? Können Sie es zum Beispiel erfassen, wenn man objektorientierte Muster in einer funktionalen Sprache verwendet? Diese Aspekte gehören zu dem Punkt, Programmiersprachen zu meistern.

Der nächste Bereich der Werkzeuge ist der Technologie-Stack. Wer sich als Java-EE- oder Unity-3D-Entwickler bezeichnet, muss sich geschickt darin bewegen und sogar andere beraten können: Warum wird X üblicherweise so gemacht? Warum verwendet Stack S Technologie T und nicht eine konkurrierende Technologie Z? Wann ist dieser Stack von Vorteil und – noch wichtiger – wann nicht?

Zuletzt kommt noch die Fähigkeit, Applikationen zu beherrschen. Sie müssen IDEs wie aus dem Schlaf kennen: 90 Prozent der Tastaturabkürzungen, Tipps & Tricks, Scaffoldings etc. Der Vergleich mag etwas hinken, aber wie bewertet man einen Autofahrer, der nur die Hälfte der Gänge verwendet und jedesmal im Benutzerhandbuch sucht, wie die Scheibenwischer eingestellt werden? Gleiches gilt für den Source-Code, dessen Verwaltungstools, Continuous-Integrationsstrecken, Release-Management-Tools etc. All diese Werkzeuge geben dem Entwickler Rückendeckung, vorausgesetzt er nimmt deren Mächtigkeit wahr.

Wenn technische Exzellenz ein notwendiger Aspekt des Entwicklerprofils ist, reicht es dennoch nicht aus, um sich deutlich von der Entwicklermasse abzuheben. Dafür müssen weitere Aspekte herangezogen werden.

Die Zusammenarbeit mit anderen

Beispiele für den Mythos des einsamen Hackers existieren zur Genüge, aber die meisten Entwickler sind dann doch politische Tiere. Sie arbeiten in Teams, sind sehr kommunikativ und erledigen zusammen deutlich mehr, als man als Einzelner schaffen könnte. Welche positiven Effekte können wir von dieser Zusammenarbeit erwarten? Unter anderem, dass man zusammen mehr erledigt, sich gegenseitig unterstützt und die üblichen Fehler nicht immer wieder selbst erleben muss. Aber man lernt auch schneller und kommt zügiger voran. Deshalb werden die folgenden drei Punkte vordergründig betrachtet:

- Pair Programming
- Mentoring
- Sich bewusst sein, ein Lern-Vektor zu sein

Der erste Schritt ist also, die Einsamkeit zu bekämpfen, zu zweit (oder zu mehreren) zu lernen und effizient zu arbeiten. Pair Programming ist eine der stärksten Austauschmethoden, die man einsetzen kann. Zwei Entwickler arbeiten dabei paarweise zusammen an der selben Aufgabe und an einem Rechner. Diese Technik wird oftmals sehr kritisiert, da zwei Entwickler die Arbeit von einem erledigen. Es stellt sich die Frage, wie sich diese Arbeitsweise rentieren kann?

Dazu gibt es bereits mehrere Studien (beispielsweise „*Strengthening the Case for Pair-Programming*“ von Ward Cunningham, Ron Jeffries, Laurie Williams und Robert R. Kessler [5]), aber das Fazit ist immer wieder dasselbe: die Grundgeschwindigkeit von Paaren ist bei einfachen Aufgaben oft niedriger als die von Entwicklern, die allein arbeiten. Dies wird sicherlich am Kommunikationsaufwand liegen, der da entsteht.

Sobald die Komplexität oder Ungewissheit eines Problems zunimmt, relativiert sich dieser Geschwindigkeitsverlust durch einen drastischen Qualitätsgewinn. Der Einsatz von Paaren rechnet sich dann sofort, da die Fehleranalyse, die sonst vermutlich nötig wäre und sehr zeitaufwändig ist, reduziert wird.

Des Weiteren sollte man den Know-how-Transfer berücksichtigen. Egal ob Junior, Senior oder gemischt, Paare bringen sich gegenseitig viel bei und wachsen gemeinsam schneller als allein. Es ist eine sehr gute Methode, um neue Mitarbeiter schnell zu integrieren.

Und zu guter letzt wird der Spaß der Mitarbeiter und das Teamverhalten immer wieder gelobt.

Aber eine kleine Warnung seitens des Autors. Wer sich mit Pair Programming intensiv beschäftigt, sollte streng auf die Zeit achten, da diese Methode ziemlich anstrengend sein kann. Man sollte zudem keine Angst haben, die Mitarbeiter nach 6 Stunden einfach mal nach Hause zu schicken. Pair Programming ist dennoch nicht mit jedem möglich, obwohl der Erfolg auch bei Entwicklern eintrat, die vorher mehr als skeptisch waren. Es lohnt sich also, dieser Methode eine Chance zu geben und sie auszuprobieren.

Das Nächste, was man unbedingt machen sollte, ist einen Mentor zu finden. Unter Mentoring wird hier eine aktive, gezielte Kollaboration zwischen Menschen mit Synergien und Emulationen ihrer Gedanken verstanden. Während der typische Kung-Fu-Meister seinem Lehrling die Weisheit lehrt, ist der Mentor ganz im Gegensatz dazu ein Partner auf Augenhöhe. Der Mentor ist zuerst einmal dazu da, um Feedback zu geben. Deswegen muss der Mentor nicht ein erfahrenerer Mensch sein, jedoch ist das oft einfacher. Wenn man von Feedback spricht, muss man auch die andere Seite betrachten: Feedback einholen. Der Mentee kann und darf nicht passiv blei-

ben. Es ist seine Rolle und in seinem Interesse, Fragen zu stellen und zu versuchen, dem Mentor das Wissen abzusaugen oder Feedback einzuholen.

Welche Aufgaben können im Rahmen einer Mentor-Mentee-Beziehung erledigt werden? Über Side-Projekte reden, zusammen entwickeln, sich über Bücher austauschen etc. Das ist alles ist möglich.

Ein paar Punkte müssen dabei aber vorab geklärt werden:

- Beide Parteien müssen sich darüber einig sein, was Sie von dieser Beziehung erwarten.
- Eine Zeit-Investition sollte vereinbart werden, an die sich jede Partei hält.
- Eine Exit-Strategie sollte vorab diskutiert werden, um den Partner zu respektieren. Nichts ist schlimmer als zu versuchen, diese Beziehung passiv sterben zu lassen.

Anfang 2015 sagte Uncle Bob in einem Podcast von DotNet Rocks:

> *The number of programmers in the world doubles every 5 years, which means that half the programmers in the world have less than 5 years experience [...] it leaves this industry in a state of perpetual inexperience. [6]*

Uncle Bob gibt im Podcast selbst zu, dass die Zahlen nicht ganz stimmen mögen. Aber die Tendenz könnte schon in diese Richtung gehen. Es heißt aber auch, dass jeder, der mehr als 5 Jahre Erfahrung besitzt, sich als Mentor berufen fühlen sollte. Ein Mentor zu sein ist vorteilhafter als man denkt. Der Networking-Aspekt und der Ruf, den man damit erhalten kann, werden die eigene Reputation erheblich stärken.

Oben wurde sowohl im Pair-Programming- als auch im Mentoring-Absatz vom Wissensaustausch und von den Lerneffekten geschrieben. Der kommende Punkt ist aber so wichtig, dass er einen eigenen Absatz verdient. Denn es ist essenziell ein Verbesserungs-Vektor für den anderen zu sein! Diese altruistische Sicht kann auch egoistisch begründet sein: Ich wage zu behaupten, dass wir ein Thema erst wirklich meistern können, wenn wir es gelehrt haben. Einige haben sicherlich schon folgendes erlebt: In der Architektur und im Design scheint alles perfekt zu sein, aber sobald man versucht, es anderen Leuten zu erklären, fallen einem die Problemen auf. Dieser Effekt ist auch als „Rubber-Duck-Problem-Solving" bekannt. Um den Problemen vorzugreifen, kann man folgendes Szenario durchspielen: Man stellt eine Plastikente auf den eigenen Bildschirm und erklärt dieser Ente die vorhandenen Probleme. Dies ist fast so gut wie eine Psychotherapie.

Neben Pair Programming und Mentoring gibt es unendlich viele Möglichkeiten den Entwickler-Kollegen zu helfen. Die Amerikaner reden z. B. vom „Brown Bag Lunch", einem Mittagessen inklusive eines Vortrag und einer Diskussion unter Kollegen. Hier gibt es verschiedene Formen. Egal in welcher Stadt man lebt, es gibt dort sicher User-Groups, Meetups oder Konferenzen in der Nähe, zu denen man hingehen sollte. Die Teilnehmerzahl ist dabei unwichtig – ob 2 oder 200 Teilnehmer –, die Veranstalter solcher Treffen sind immer für ehrliche Diskussionen offen. Auch wenn man klein anfängt, die Hauptsache ist, man fängt an.

Zum Schluss noch ein paar Sätze über das Bloggen und Schreiben von Artikeln. Dies sind sehr gute Marketing- und Vorbereitungsmaßnahmen. Wer sich und seinen Namen voranbringen will, sollte diese Möglichkeiten nutzen, da sich Menge und Inhalt auf Dauer am meisten rechnen. Außerdem dient es auch – wie beim Rubber-Ducking – der eigenen Weiterbildung. Der Austausch und die Diskussion stehen dabei im Vordergrund, und diese Wege bieten ein sehr geringes Feedback/Aufwand-Verhältnis.

Sich selbst

Am härtesten ist es wahrscheinlich, die Soft Skills zu meistern. Und was Kandidaten künftig mitbringen müssen, sind folgende Eigenschaften:

- Sich-selbst-kennen
- Professionalismus, Zweck und Ethik
- Neugier, Experimentierfreudigkeit

Der wichtigste Punkt ist dabei, dass man zuerst einmal sich selbst kennt. Es ist das A und O, um seinen Weg im Berufsleben zu finden und dabei glücklich zu sein. Dies ist wiederum nicht unbedingt altruistisch zu sehen, da ein glückliches Team-Mitglied sein Bestes geben kann, wobei das Gegenteil aber eher fragwürdig wäre.

Was möchte ich machen? Was möchte ich überhaupt nicht machen? Es kann sowohl um die Domäne, die Technologie oder auch die Aufgabe gehen. Das Schlüsselwort ist hier Proaktivität. Wer weiß, in welche Richtung er gerne gehen will, der kann sein Leben dahin lenken und Einfluss darauf nehmen. Man kommt natürlich auch so durchs Leben. Das ist aber alles eine Einstellungssache oder ein Glückstreffer.

Was für ein Mensch ist man? Welche Rolle übernimmt man im Team? Welches psychologische Profil nach Briggs-Myers [7] hat man? Welche Farbe ist es? Kann man dieses Profil auch bei anderen Personen erkennen?

Wenn man sich ins Team einordnen und die Konflikte vorhersehen kann, ist man deutlich schneller in der Lage, Einfluss darauf zu nehmen.

Welche Rahmenbedingungen braucht man, um am effizientesten zu sein? Braucht man die Möglichkeit des Open-Space oder eher Ruhe? Kann man effizient „remote" arbeiten oder wird direktes Feedback benötigt?

Alle diese Aspekte muss man an sich selbst beobachten und kennen, um den Kunden wiederum sein Bestes zu zeigen.

Wenn wir von unseren Kunden sprechen, sind diese in der Regel das Wichtigste, oder? Ja und nein. Ärzte und Anwälte legen z. B. den hippokratischen Eid ab, um den Werten und der Ethik ihrer Berufsbranche zu folgen. Im Jahr 2013 hat Edward Snowden von der NSA gezeigt, dass ethische Grundsätze in der IT-Branche noch keine weit verbreiteten Eigenschaften sind. Eigentlich kann sich jeder Software-Entwickler nennen. Ein paar Online-Kurse reichen schon aus, um Kundenwünsche erfüllen zu können. Dies ist jedoch anscheinend der kleinste gemeinsame Nenner, um Entwickler zu sein. Was ist mit der Datenverletzung, wo speichert der Online-Vermittlungsdienst für Fahrdienstleistungen GPS-Koordinaten, Bewegungen und private Information von Prominenten? Es gibt leider noch deutlich mehr Beispiele. Auch bei uns in Deutschland, wie wir 2015 mit Hinblick auf die VW-Affäre erfahren haben.

Datensparsamkeit ist hier der übergeordnete Begriff; und als Entwickler gehört man zu denjenigen, die als einzige an der Quelle von solchen Problemen sitzen. An dieser Stelle behaupte ich, dass nur die Entwickler die Hand heben können, da ansonsten keiner mehr den Überblick hat. Das ist ein Aspekt, der zum Professionalismus zählt. Ein anderer davon ist natürlich, sich zu bemühen, Probleme selbst bis ins Ziel zu tragen und Kundenwünsche bestmöglich erfüllen zu wollen. Es heißt aber auch, bereit zu sein und in Konfrontation mit diesen Kunden zu gehen, um ihnen die wichtigen Aspekte der Probleme aufzuzeigen – unter anderem die ethischen.

Zuletzt ist unter Professionalismus auch zu verstehen, sich mit dem Zweck der eigenen Arbeit auseinanderzusetzen. Deshalb ist an dieser Stelle der Tweet von Scott Hanselmann [8] empfehlenswert:

> *Whatever software you're making, if it's not saving babies, you need to CHILL OUT.*

Hier sind wir wieder im Bereich „Sich-selbst-kennen" angelangt. Was ist der Zweck der eigenen Arbeit? Wofür arbeitet man so lang am Abend?

Die letzten Puzzlestücke, die bei Einhorn-Entwickler zu suchen sind, sind Neugier und Experimentierfreudigkeit. Niemand weiß alles. Da kann das Buzzword-Bingo zwischen Stellenanzeige und Lebenslauf noch so gut sein, man muss immer mit Lücken rechnen. Wenn ein Kandidat neugierig ist, wird er seine Lücken selbst ausfüllen, selbstständig untersuchen und sich eigenständig weiterbilden. Neugierige Mitarbeiter übernehmen Verantwortung, stellen die unangenehmen Fragen und akzeptieren keine leeren Antworten. Ohne Pragmatismus kann diese Persönlichkeitseigenschaft natürlich sehr anstrengend sein, daher muss es ein Gleichgewicht zum Professionalismus geben. Aber ohne die Neugier werden Profile schnell uninteressant.

Und nun?

Nehmen Sie diese Aspekte mit und lassen Sie sie sinken. Kehren Sie regelmäßig zurück und lesen Sie dies hier wie einen Leitfaden. Diese Aspekte nur abzuhaken, ist nicht das Ziel, sondern man soll sich bewusst machen, dass es sie gibt und sich damit – wenn auch kurz – auseinandersetzen.

Wer beruflich als „Hüter" von Einhorn-Entwicklern fungiert, sollte auch darüber nachdenken, inwiefern seine Aktionen die Weiterentwicklung hindern oder erlauben. Wird genug Zeit zum Erforschen geschaffen? Werden Diskussionen und Austausch, idealerweise in der Arbeitszeit, ermöglicht? Werden externe Referenten eingeladen? Wird eine Lernkultur in der Firma gepflegt? Nur so wird eine Aufzucht von Einhorn-Entwicklern möglich und die eigene Firma bleibt dann auch für diejenigen attraktiv. Daher gibt es nur zwei Optionen, wie dieser Witz zeigt:

CTO: "Wir müssen unbedingt mehr in die Weiterbildung unserer Leute investieren."
CEO: "Ja, und dann sind sie im Markt besser aufgehoben und verlassen die Firma... super."
CTO: "Was ist die Alternative? Wir bilden sie nicht weiter und sie bleiben bei uns?"

Geheimnis gelöst? Sicher nicht!

Das beschriebene Profilbild eines Einhorn-Entwicklers ist natürlich nicht vollständig und auch „etwas" subjektiv geprägt. Bei den Recherchen wurden viele Entwickler interviewt, sodass deren Meinungen auch einfließen konnten. Sie können und dürfen mit diesen Meinungen nicht einverstanden sein.

Viele Aspekte mussten im Rahmen dieses Artikels außen vor gelassen werden. Diese werden womöglich in einem nachfolgenden Buch angegangen werden, wie z. B.

Open-Source, Üben, Minimalismus, Agilität, Schweigen können, Networking, Personal-Branding, Politik und noch vieles mehr. Dieses Buch ist als eine Art Sammlung von Aufsätzen zu sehen, die als Erinnerungsstütze oder Leitfaden für eine Entwicklerkarriere dienen können. Es sind eher gedankenauflösende Aufsätze und nicht unbedingt Lösungen, die festgehalten werden. Der Arbeitstitel lautet „Developer's Journey".

Wenn Sie sich, liebe Leserinnen und Leser, in diesem Artikel wiedererkennen, dann interessiert mich Ihr Feedback. Sind Sie mit diesem Profil des Einhorn-Entwicklers nicht einverstanden, ist es umso besser, solange ich Ihr Feedback erhalte. Bitte kontaktieren Sie mich unter: *info@developersjourney.info*. Falls Sie ein Beta-Leser werden wollen, melden Sie sich einfach für den Newsletter via *www.developersjourney.info* an. Ich freue mich schon auf Ihr Feedback.

Referenzen

[1] Fowler, Chad *Who I want to hire*, 2013
http://chadfowler.com/blog/2013/04/09/who-i-want-to-hire.html

[2] Hart Research Associates *Falling Short? College Learning and Career Success*, 2015
https://www.aacu.org/leap/public-opinion-research/2015-survey-results

[3] Hanselman, Scott *Dark Matter Developers: The Unseen 99%*, 2012
http://www.hanselman.com/blog/DarkMatterDevelopersTheUnseen99.aspx

[4] Wikipedia *Clean Code*,
https://de.wikipedia.org/wiki/Clean_Code

[5] Cunningham, Ward; Jeffries, Ron; Williams, Laurie; Kessler, Robert R. *Strengthening the Case for Pair-Programming*,
http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF,

[6] Martin, Robert C. *Beyond Software Craftsmanship with Uncle Bob*
https://www.dotnetrocks.com/default.aspx?showNum=1094, 2015

[7] Wikipedia *Myers-Briggs Typeindikator*,
https://de.wikipedia.org/wiki/Myers-Briggs-Typenindikator

[8] Hanselman, Scott *Whatever …*
https://twitter.com/shanselman/status/233077423888027648, 2012

Kurzbiografie

Tim ist ein Alltags-Geek, Agile Entwickler, Bullshit-Jäger, internationaler Speaker und vieles mehr. Er ist neugierig und kreativ; er liebt es zu forschen, lernen, lehren und reden – genau das hat er laut seiner Familie schon immer getan. Als gebürtiger Franzose lebt er in Deutschland und arbeitet als Berater, Entwickler, Scrum Master und Agile Coach. Wenn er nicht hinter einem Rechner oder in einen Scrum-Team sitzt, ist er hinter seiner Kamera, in seinen Laufschuhen oder bei Frau und Kindern zu finden – in immer wechselnder Reihenfolge!

Copyright © 2016 Bookware 1865-682x/16/07/001 Von diesem KaffeeKlatsch-Artikel dürfen nur dann gedruckte oder digitale Kopien im Ganzen oder in Teilen gemacht werden, wenn deren Nutzung ausschließlich privaten oder schulischen Zwecken dient. Des Weiteren dürfen jene nur dann für nicht-kommerzielle Zwecke kopiert, verteilt oder vertrieben werden, wenn diese Notiz und die vollständigen Artikelangaben der ersten Seite (Ausgabe, Autor, Titel, Untertitel) erhalten bleiben. Jede andere Art der Vervielfältigung – insbesondere die Publikation auf Servern und die Verteilung über Listen – erfordert eine spezielle Genehmigung und ist möglicherweise mit Gebühren verbunden.

Herbstcampus

Wissenstransfer par excellence

30. 8. – 1. 9. 2016
in Nürnberg

Hin und Zurück

Setters/Getters vs. OOP

von Róbert Bräutigam

Setter-/Getter-Methoden in Java sind so verbreitet, dass man sie in fast jeder Klasse wiederfindet. Manche Entwickler generieren sie gar automatisch. Wenn man es jedoch genauer betrachtet, widersprechen solche Methoden grundsätzlich der objektorientierten Programmierung und sind nichts anderes als ein „Code-Smell".

Objektorientierung

Manchmal erklärt man Objektorientierung, indem man Objekte mit früheren, prozeduralen Strukturen vergleicht. Man sagt, Objekte sind nichts weiter als Daten mit Funktionen, die mit diesen Daten arbeiten können oder dürfen.

Obwohl das korrekt ist und vielleicht reicht sich einen ersten Eindruck zu verschaffen, ist es keine ausreichend detaillierte Definition. Objekte sollten hauptsächlich Verantwortlichkeiten haben, unabhängig davon, welche Daten sie haben oder ob überhaupt Daten vorliegen. Die Daten sollten für die Außenwelt nicht wichtig sein.

In der Tat ist es „verboten" die Daten innerhalb eines Objektes nach außen zu geben. Diese Regel nennt man auch Datenkapselung und ist eine der wichtigsten Eigenschaften um wartbaren, wiederverwendbaren Code schreiben zu können.

Setter-/Getter-Methoden

Alle Java-Entwickler haben schon Setter-/Getter-Methoden geschrieben oder von einer IDE generieren lassen:

```
public class Person {
  private String name;
  private String prefix;
  ...
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }

  public String getPrefix() {
    return prefix;
  }

  public void setPrefix(String prefix) {
    this.prefix = prefix;
  }
}
```

Hier wird der Name und das Namenspräfix von der Person nach außen gegeben bzw. gesetzt. Das verletzt eindeutig die Datenkapselung, da die internen Daten nach außen gegeben werden.

Das Problem ist nicht nur rein theoretisch. Im Enterprise-Bereich kommt es beispielsweise häufig vor, dass man das Datenmodell ändern muss, um neue Features zu implementieren. Falls diese Modelle nicht objektorientiert geschrieben worden sind oder die Datenkapselung nicht richtig eingeplant worden ist, spürt man diese Änderung überall in der Applikation. Es fängt mit der Datenschicht an, geht dann weiter zum „Business Logik"-Code und früher oder später zum UI. Alles muss sich an die neue Datenstruktur anpassen und das steigert auch die Kosten, das Feature zu implementieren.

Wie denn sonst?

Anstatt interne Daten anzubieten, muss das Objekt in der Lage sein, den gewünschten Vorgang selbst auszuführen. Das nennt man „*Tell, don't ask*"-Prinzip, auf Deutsch: Sag, frag nicht!

Generisch ohne einen genauen Anwendungsfall kann man die Getter-/Setter-Methoden nicht ersetzen. Es

muss eine Verantwortlichkeit geben, weshalb das Objekt gefragt werden kann, etwas zu tun. Nehmen wir beispielsweise folgenden Code:

```
String emailSalutation =
  person.getPrefix() + " " + person.getName();
```

Dieser Code basiert auf den Getter-Methoden, um die volle Anrede zusammenzustellen. Wie kann man aber sonst die Anrede erstellen, wenn das Objekt die Methoden nicht bereit stellt? Zum Beispiel so:

```
public class Person {
  private String prefix;
  private String name;
   ...
  public String getSalutation() {
    return prefix + „ „ + name;
  }
}
```

Auf diese Weise ist die Datenkapselung erhalten geblieben und es wurden keine internen Daten veröffentlicht. Obwohl die Methode auch mit „get“ anfängt, ist es keine Getter-Methode, da sie keine internen Daten liefert, sondern etwas kalkuliert, das Geschäftslogik darstellt.

Man könnte nun argumentieren, dass Personen manchmal doch neue Namen erhalten. Deshalb ist „setName()“ vollkommen berechtigt. Falls das tatsächlich im System abgebildet sein muss, dann gibt es auch einen Use-Case dazu. Der Use-Case wird wahrscheinlich nicht sagen, dass es möglich sein muss den Namen zu setzen, sondern die Möglichkeit beschrieben wird, dass Personen z. B. beim Heiraten den Namen wechseln können. Um das abzubilden ist diese Lösung vielleicht besser:

```
public void rename(String newName) {
  this.name = newName;
}
```

Der kleine aber sehr wichtige Unterschied zwischen Setter- und Business-Methode liegt letztlich in der Wiederverwendbarkeit und Wartbarkeit der Klasse. Das Ziel ist dabei, dass interne Änderungen an der Klasse innerhalb der Klasse bleiben. Werden z. B. mehrere Namenspräfixe gebraucht, muss die „getSalutation()“-Methode nur intern geändert werden. Andere Klassen, die diese Methode nutzen, müssen nichts ändern.

String oder Name?

Was passiert aber, wenn sich die Namen ändern? Zum Beispiel sollen die Namen jetzt nicht Strings sein, sondern auf Vorname und Nachname aufgeteilt werden. Es scheint, dass diese Änderung nicht auf die Person begrenzt sein wird.

Das Problem stammt aus der Entscheidung Namen mit Strings zu implementieren. Um solche Änderungen abbilden zu können, muss eigentlich der Name selbst als eine eigene Klasse geschrieben werden:

```
public class Name {
  private String name;
  ...
}
```

Diese Klasse muss natürlich selbst ein „richtiges“ Objekt sein, mit eigenen Verantwortlichkeiten. Zum Beispiel kann der Name wissen, wie man diesen Namen formell oder informell anspricht:

```
public class Name {
  private String name;
  ...
  public String getFormalAddressing() {
    return name;
  }
  public String getInformalAddressing() {
    return name;
  }
}
```

Die Person würde dann so aussehen:

```
public class Person {
  private Name name;
  private String prefix;
  ...
  public String getSalutation() {
    return prefix + " " + name.getFormalAddressing();
  }
}
```

Jetzt ist es möglich den Namen, auf Vorname und Nachname aufzuteilen, ohne dass die anderen Klassen geändert werden müssen:

```
public class Name {
  private String firstName;
  private String lastName;
  ...
  public String getFormalAddressing() {
    return firstName + " " + lastName;
  }
}
```

Persistenz und UI

„Herkömmliche“ Architekturen im Enterprise-Bereich bauen normalerweise Schichten, um die Persistenz, die Business-Logik und die UI zu separieren. Der Übergang zwischen diesen Schichten wird dabei häufig mit „Beans“ gelöst.

„Beans“ sind einfache Strukturen, die nur Daten und die dazugehörigen Setter-/Getter-Methoden definieren. Da sie keine Verantwortlichkeiten haben, sind diese Konstruktionen nicht wirkliche „Objekte“. Dieses Design ist wahrscheinlich deshalb so beliebt, weil es die maximale Flexibilität bietet, wie man die Daten in anderen Schichten behandelt.

Es ist also auch in Situationen anwendbar, bei denen es noch nicht ganz klar ist, was man mit den Daten machen möchte, wenn noch kein richtiges Modell zur Verfügung steht oder wenn das Modell noch nicht ganz verstanden wurde. Das alles führt dazu, dass diese Patterns einfach zu implementieren sind.

Leider wird die gesparte Zeit später vielfach zurückgezahlt, da die Schichten jetzt durch die Daten fest gekoppelt sind und deshalb fast jede Änderung alle Schichten betreffen wird.

Die objektorientierte Lösung erlaubt solche schichtbasierten Architekturen nicht, zumindest nicht in dieser Form. Da die Objekte Daten nicht nach „außen“ geben dürfen, müssen sie sich selbst darstellen oder auch sich selbst speichern. Wie sie das machen, hängt vom konkreten Anwendungsfall ab. Der Punkt ist aber, dass diese Belange grundsätzlich keine Ausnahmen bilden.

Fazit

Obwohl Setter-/Getter-Methoden ein sehr verbreitetes Muster sind, deuten diese eigentlich auf einen Kapselungsfehler und sind daher ein „Code-Smell“.

Falls man sich entscheidet die Kapselung richtig zu implementieren, kann man die Wartbarkeit von Code deutlich steigern. Richtige Objekte zu schreiben, benötigt allerdings manchmal eine komplett andere Denkweise, als man in Java-Enterprise vielleicht gewohnt ist. Dabei sind Schichtenarchitekturen, UI und Persistenz ebenfalls eine Herausforderung.

Kurzbiografie

Robert Bräutigam ist als Senior Consultant für MATHEMA Software GmbH tätig. Er ist seit 1999 als Entwickler und Architekt im Java Enterprise-Umfeld beschäftigt und interessiert sich für leichtgewichtige Lösungen im technischen Bereich sowie im Projektmanagement.

Herbstcampus

Wissenstransfer par excellence

30. 8. – 1. 9. 2016
in Nürnberg

Des Programmierers kleine Vergnügen

Mutabor

von Michael Wiedeking

Was nutzt es einem, eine hexadezimale ASCII-Ziffer in seinen Wert umwandeln zu können, wenn man (wie etwa der Kalif Storch) nicht weiß, wie man von dort wieder zurück kommt.

Um zu einer Hexadezimalzahl x mit $0 \leq x \leq 15$ die korrespondierende Ziffer zu liefern, gibt es eine sehr einfache Methode: Man definiert ein Array *digits* mit den möglichen 16 Zeichen und liefert über Indizierung einfach das entsprechende Zeichen zurück.

```
static char[] digits = {
   '0', '1', '2', '3', '4', '5', '6', '7',
   '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'
};

static char charOf(int x) {
   // assert 0 ≤ x ≤ 15;
   return digits[x];
}
```

Und das ist das Schnellste, was man machen kann.

An dieser Stelle könnte die Kolumne eigentlich aufhören, wenn nicht die Ausgangsbedingungen all zu ideal wären. Sind die abzubildenden Werte nahezu lückenlos aufgereiht, ist eine derartige Array-Lösung fast immer die beste. Selbst bei größeren Lücken kann die Array-Lösung – auch bei vielen, niemals genutzter Einträgen – sehr attraktiv sein. Erst wenn die Größe des Arrays unüberschaubar oder unwirtschaftlich wird, muss man nach einer anderen Lösung suchen.

Bei dieser Aufgabe gibt es ja das Problem, dass die Werte von 0 – 9 auf einen anderen Bereich abgebildet werden müssen als die Werte von 10 – 15.

```
if (x < 10) {
   return (char) ('0' + x);
} else {
   return (char) ('A' + x – 10);
}
```

Alternativ kann auch wie folgt vorgegangen werden:

```
int c = '0' + x;
if (x >= 10) {
 c += 'A' – '0' – 10;
 // c += 7
}
return (char) c;
```

Im letzten Vergnügen [1] haben wir uns das Bit-Muster derart zunutze machen können, dass wir die Bedingung eliminieren konnten. Das geht hier aber nicht so gut, weil sich die gesuchte Grenze $10 = (1010)_2$ nicht so richtig in den Bits wieder findet; jene liegt nämlich bei $8 = (1000)_2$.

Es müsste also – in Anlehnung an die oben gegebene alternative Implementierung – eine Möglichkeit geben, nur dann die 7 hinzu zu addieren, wenn $x \geq 10$ ist. Wie immer benötigen wir also eine Maske, die entweder Null ist oder nur dann vollständig aus Eins-Bits besteht, wenn die Bedingung erfüllt. Dann lässt sich nämlich

```
int m = … ;
return (char) ('0' + x + (7 & m));
```

schreiben. Ist $x < 10$, dann ist $m = 0$ und die 7 wird eliminiert; andernfalls wird sie wunschgemäß hinzugezählt. Und wie muss m jetzt aussehen?

Na ja, ist $x \geq 10$ (oder anders geschrieben $x > 9$), dann und nur dann ist $9 - x < 0$. Betrachtet man nun das Vorzeichen-Bit der Differenz, so ist das nur bei erfüllter Bedingung 1. Also liefert

```
int m = (9 – x) >> 31;
```

die gewünschte Maske, die entweder 0 ist, oder für $x > 9$ nur aus Einsen besteht.

Aber diese Variante ist so deutlich langsamer als die Array-Lösung, dass sich das wirklich nur dann lohnt, wenn ein Array aus oben genannten Gründen nicht in Frage kommt.

Referenzen

[1] Wiedeking, M. *Des Programmierers kleine Vergnügen – Zauberhafte Verwandlung*, KaffeeKlatsch, Jahrgang 9, Nr. 6, S. 18f, Bookware, Juni 2016 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2016-06.pdf

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

Java UserGroup Bremen
http://www.jugbremen.de
Kontakt: Rabea Gransberger (rgransberger@gmx.de)

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

Java User Group Saxony
Java User Group Dresden
http://www.jugsaxony.de
Kontakt: Herr Falk Hartmann
(falk.hartmann@jugsaxony.org)

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

JUG Deutschland e.V.
Java User Group Deutschland e.V.
c/o Stefan Koospal
http://www.java.de (office@java.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 Görlitz
Java User Group Görlitz
http://www.jug-gr.de
kontakt@jug-gr.de

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

JUG Karlsruhe
Java User Group Karlsruhe
http://jug-karlsruhe.de
(jugkarlsruhe@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: (info@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),
Rainer Anglett (ra@jugs.org)

Schweiz

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

.NET User Groups

Deutschland

.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
(info@dodnedder.de)

.NET UserGroup Frankfurt
http://www.dotnet-usergroup.de

.NET User Group Friedrichshafen
http://www.dotnet-fn.de
Kontakt: Tobias Allweier
(info@dotnet-fn.de)

.NET User Group Hannover
http://www.dnug-hannover.de
Kontakt: (dnug@indisoftware.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 Developers Group Stuttgart
http://www.devgroup-stuttgart.net
(GroupLeader@devgroup-stuttgart.net)
Kontakt: Herr Michael Niethammer

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

Österreich

.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, Schweiz, Österreich

Softwerkskammer – Mehrere regionale Gruppen und Themengruppen unter einem Dach
http://www.softwerkskammer.org
Kontakt: Nicole Rauch (nicole.m@gmx.de)

Metropolregion Nürnberg
JUG

Die Java User Group
Metropolregion Nürnberg
trifft sich regelmäßig einmal im Monat.

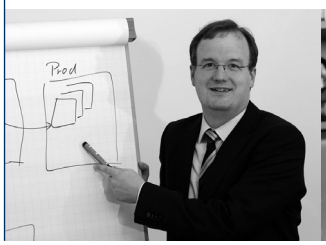
Thema und Ort werden über
www.jug-n.de
bekannt gegeben.

Weitere Informationen
finden Sie unter:
www.jug-n.de

MATHEMA

meet the experts of enterprise infrastructure

- **Entwicklung mobiler Anwendungen mit iOS**
  19. – 21. September 2016, 1.250,– € (zzgl. 19 % MwSt.)
- **Anwendungsentwicklung mit der Java Enterprise Edition**
  19. – 23. September 2016, 2.150,– € (zzgl. 19 % MwSt.)
- **Einführung in C# und .NET**
  Einstieg in C# und die .NET-Plattform für Programmieranfänger
  26. – 30. September 2016, 2.150,– € (zzgl. 19 % MwSt.)
- **Entwicklung mobiler Anwendungen mit Android**
  7. – 9. November 2016, 1.250,– € (zzgl. 19 % MwSt.)
- **AngularJS**
  28. – 29. November 2016, 950,– € (zzgl. 19 % MwSt.)
- **Scrum Basics**
  950,– € (zzgl. 19 % MwSt.)
- **Scrum im Großen** – Agiles Organisationsdesign nach LeSS, 950,– € (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

„An MATHEMA schätze ich unsere öffentliche Präsenz, z. B. bei User Groups und Konferenzen, sowie die Vielseitigkeit meiner Tätigkeit."

Thomas Haug, Principal Consultant

Wir sind ein Consulting-Unternehmen mit Schwerpunkt in der Entwicklung unternehmenskritischer, verteilter Systeme und Umsetzung von Service-orientierten Architekturen und Applikationen von Frontend bis Backend. Darüber hinaus ist uns der Wissenstransfer ein großes Anliegen: Wir verfügen über einen eigenen Trainingsbereich und unsere Consultants sind regelmäßig als Autoren in der Fachpresse sowie als Speaker auf zahlreichen Fachkonferenzen präsent.

join the experts of enterprise infrastructure

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

Das Allerletzte

```
// Dear future me. Please forgive me.
// I can't even begin to express how sorry I am.
```

Liebes zukünftiges Ich. Bitte vergib mir. Ich weiß gar nicht wie ich anfangen soll, zum Ausdruck zu bringen, wie leid es mir tut.

Dies ist kein Scherz!
Dieser Code-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 im August.

Herbstcampus

Wissenstransfer par excellence

30. August – 1. September 2016
in Nürnberg