KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

07/2014

Jahrgang 7

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

Wissen interaktiv

Wenn jemand etwas zu Papier bringt, muss er sich nicht nur viele Gedanken darüber machen, was er sagen möchte, sondern auch darüber, wie er es sagen möchte. Das gilt natürlich grundsätzlich für jede Art von Kommunikation, aber bei dem geschriebenen Wort scheint es dann doch immer etwas aufwändiger.

Es ist ja auch wirklich einfacher, jemandem etwas persönlich zu erklären, bekommt man doch durch den Blick in das Gesicht des Gegenübers sofort Rückmeldung darüber, wie erfolgreich man seine Aufgabe erledigt. Schließlich erlaubt einem das unmittelbare Feedback ja auch noch die Richtung zu ändern, auf unausgesprochene Fragen zu reagieren und nicht zuletzt die Art des Erklärens zu überdenken.

Ein Vortrag bietet, was ein größeres Publikum anbetrifft, einen guten Kompromiss. Natürlich muss man sich hier genauso sorgfältig vorbereiten wie bei einem Artikel. Aber man hat während des Vortrags noch die Möglichkeit ein bisschen an der einen oder anderen Schraube zu drehen. Wird man nicht durch schlechte Beleuchtung daran gehindert, kann man sogar direktes Feedback sehen. Und es gibt selbstverständlich die Möglichkeit auf konkrete Fragen zu reagieren.

Für den Konsumenten ist es nicht immer leicht, Texte zu verstehen. Der Weg, den der Autor geht, kann zu kompliziert sein, einfach nur zu unübersichtlich oder unerwartungsgemäß. Der Stil kann ablenken, der rote Faden zu blass oder das Ziel nicht klar erkennbar sein und womöglich ist der Text mit Fremdwörtern durchsetzt. Mit einem Vortrag kann einem das zwar grundsätzlich auch passieren, aber da hat man immer die Möglichkeit, sich anschließend etwas erklären zu lassen, Fragen zu stellen oder Meinungen zu diskutieren.

Wer also nach einer interaktiveren Art der Wissensvermittlung sucht, der kann auf dem Herbstcampus fündig werden, denn der ist so eine Plattform für Vorträge, in denen man sich nicht nur Ideen und Anleitungen holen, sondern auch direkt mit den Referenten und anderen Teilnehmern sprechen kann.

Darüber hinaus ist eine Unterhaltung mit dem Referenten ein einmaliges 3D-Ereignis, das man ganz ohne zusätzliche Brille räumlich genießen kann; in einer deutlich besseren Auflösung, als diejenige, die uns 4K-Kino bietet, mit einer unschlagbaren Einzelbildfrequenz; mit einer Farbauswahl, die weit mehr als 16,7 Mio. Farben bietet, bei perfektem Kontrast; und das Ganze mit einer Tonqualität, die jedes Dolby-Surround-System in den Schatten stellt. Und – wenn der Referent nichts dagegen hat – mit haptischen Extras.

Mit anderen Worten: Lebendiger als jedes Papier und interaktiver als jede App.

Also freue ich mich darauf, Sie persönlich auf dem Herbstcampus begrüßen zu dürfen.

Ihr Michael Wiedeking
Herausgeber

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

Das Copyright des KaffeeKlatschs liegt vollständig bei der Bookware. Wir gestatten die Übernahme des KaffeeKlatschs 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 7, Nummer 7, Juli 2014
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

Editorial 3

Beitragsinfo 4

Inhalt 5

User Groups 21

Werbung 23

Das Allerletzte 25

Artikel

Konfigurationssicher

Einfache typsichere Konfiguration mit Java 6

Angemerkt

Android-Entwicklung mit AndroidAnnotations 9

Lost in Translation

Reguläre Ausdrücke als englische Sätze 14

Kolumnen

Qual der Wahl

Des Programmierers kleine Vergnügen 18

Sie haben ihr Ziel erreicht

Kaffeesatz 20

Konfigurationssicher

Einfache typsichere Konfiguration mit Java 6
von Róbert Bräutigam

Konfigurationsmanagement für Java Enterprise- und Web-Projekte ist kein neues Thema, da jede Applikation irgendwelche Einstellungen oder Einstellungsmöglichkeiten braucht. Selten wird aber bedacht, wie man als Entwickler Zugriff auf diese Werte hat, und es gibt kaum eine Lösung die mehr als eine (nicht typsichere) *Map*-ähnliche Schnittstelle zur Verfügung stellt.

Angemerkt

Android-Entwicklung mit AndroidAnnotations
von Zeeshan Ali 9

Fast jede native *Android*-Anwendung, unabhängig von der Größe und Funktionalität, enthält eine Menge Boilerplate-Code. *AndroidAnnotations* ist ein Framework, dass dieses Problem durch Reduzierung des Boilerplate-Codes löst und gleichzeitig die Android-Entwicklung intuitiver macht.

Lost in Translation

Reguläre Ausdrücke als englische Sätze 14
von Tim Bourguignon

Habt ihr schon einmal eine *Domain Specific Language* (*DSL*) geschrieben? „Wofür?“, fragt ihr? Wie wäre es um den Aufbau von regulären Ausdrücken zu vereinfachen? Unter uns gesagt, reguläre Ausdrücke schreiben zu müssen, ist genau der Grund warum ich deren Verwendung vermeide. So entstand die Idee, einen funktionalen Programmierstil zu verwenden, um eine *Fluent-API* zu definieren, die reguläre Ausdrücke generiert. Denn eine derartige „sprechende Schnittstelle“ kann man dann idealerweise auch drei Monate später immer noch einfach verstehen.

Konfigurationssicher

Einfache typsichere Konfiguration mit Java

von Róbert Bräutigam

Konfigurationsmanagement für Java Enterprise- und Web-Projekte ist kein neues Thema, da jede Applikation irgendwelche Einstellungen oder Einstellungsmöglichkeiten braucht. Selten wird aber bedacht, wie man als Entwickler Zugriff auf diese Werte hat, und es gibt kaum eine Lösung die mehr als eine (nicht typsichere) *Map*-ähnliche Schnittstelle zur Verfügung stellt.

Properties-Dateien

Die einfachste Möglichkeit Einstellungen in *Java*-Projekten bereitzustellen ist wohl durch *Properties*-Dateien, da das von Java bereits unterstützt wird [1].

Die Schnittstelle ist einfach genug: man braucht einen *String* zur Identifikation des Wertes, und man erhält einen *String* als Wert zurück:

```
String value = props.getProperty("mysetting");
```

Es gibt jede Menge Probleme mit dieser Lösung (abgesehen davon, dass *Properties*-Dateien nicht UTF-kodiert sind). Das erste Problem ist: woher wissen die Entwickler das eine Einstellung namens *mysetting* überhaupt existieren sollte? Was passiert bei einem Schreibfehler? Das zweite Problem ist natürlich, dass der Wert immer ein *String* ist. Was ist wenn man einen *Integer* oder ein *Date* braucht, oder sogar was komplexeres?

Commons Configuration

Es gibt ein *Apache*-Projekt für genau dieses Thema: *Commons Configuration* [2]. Es befasst sich hauptsächlich damit, wie man verschiedene *Backends* zum Konfigurieren benutzen und kombinieren kann, von *Properties*- und *XML*-Dateien bis zu Datenbanken und *JNDI*-Abfragen.

Die Schnittstelle zum Entwickler hat sich aber nicht viel geändert:

```
boolean valueBool = props.getBoolean("mysetting");
int valueInt = props.getInt("mysetting");
```

Wie man sieht, ist das erste Problem nicht gelöst, man muss noch immer magische Schlüssel kennen, die nur *Strings* sind. Das zweite Problem ist zumindest adressiert, aber nicht gelöst. Jetzt muss der Entwickler nicht nur den Namen der Einstellung kennen sondern auch den Typ. Kann die Einstellung wirklich mal *Boolean* und mal *Integer* sein? Auch wenn man das alles korrekt schreibt, was passiert wenn sich der Einstellungstyp ändert? Dann gibt es wieder zur *Runtime* eine *Exception*, falls überhaupt und nicht schon beim Kompilieren.

Mögliche Workarounds

Man könnte eigentlich das erste Problem lösen, indem man Enumerationen oder statische *Strings* als Schlüssel verwendet:

```
public enum SettingKeys {
  MySetting(„mysetting“),
  ...
};
```

Oder:

```
static final String MY_SETTING = "mysetting";
```

Diese Lösung muss aber von der Entwicklung kommen, und entbindet das Konfigurations-Framework nicht von der Verantwortlichkeit die Schnittstelle richtig zu definieren, um Fehler nach Möglichkeit auszuschließen.

Lösung

Wie könnte man eine Schnittstelle (unabhängig davon, woher die Einstellungen kommen) so definieren, dass es beide Probleme (Schlüssel und Wert-Typen) löst?

Die erste Beobachtung ist, dass die Einstellungen einen Typ haben. Dieser Typ sollte bei der Verwendung nicht immer festgelegt werden (also *getString()* oder *getInt()*), sondern ist eigentlich gegeben. Also:

```
public interface Setting<T> {
  Class<T> getType();
}
```

Diese Einstellungen kann man abfragen und ändern durch eine Konfiguration:

```
public interface Configuration {
  <T> T get(Setting<T> setting);
  <T> void set(Setting<T> setting, T value);
}
```

Das erste Problem ist damit gelöst, die Schlüssel für die einzelnen Einstellungen sind komplexe Objekte und fördern damit die richtige Verwendung.

Das zweite Problem ist auch gelöst, die Werte haben immer einen Typ, schon beim Kompilieren, das fördert auch die korrekte Verwendung. Wenn man eine Einstellung ändert, hilft der *Compiler* dabei wo und was man dazu alles ändern muss.

Speichern

Die Schnittstelle, wie oben definiert, ist unabhängig davon wie oder ob man die Einstellungen auch speichern will. Nichtsdestoweniger kann man die Implementation so bauen, dass man die Einstellungen in einfache *Properties*-Dateien oder auch durch *Commons Configuration* speichern kann.

Man braucht dazu nur sogenannte *Serializer*-Klassen, die gegebene Typen in *Strings* und zurück transformieren können:

```
public interface Serializer<T> {
  T deserialize(String value);
  String serialize(T value);
}
```

Die einzelnen Einstellungen könnten dann ihre eigenen *Serializer* zurückgeben:

```
public interface Setting<T> {
  ...
  Serializer<T> getSerializer();
}
```

Zeit und konvertierbare Einheiten

Oft findet man Einstellungen, die in mehrere Einheiten definiert werden können, beispielsweise ein Zeitintervall.

Normalerweise werden die Einheiten in der Konfigurationsdatei als Kommentar hinterlegt, z. B. das *Timeout* in Sekunden. Das heißt der Code muss an allen Stellen, wo das eingelesen wird, implizit wissen, welche Einheit die Konfiguration verwendet und z. B. mit 1000 multiplizieren, damit es Millisekunden werden. Ein Fehler wird so schwierig zu erkennen sein, und wenn man die Einheit irgendwo ändert, vergisst man garantiert einige Stellen.

Das Problem stammt eigentlich von der Entscheidung, solche Einstellungen als *long* zu typisieren, weil Konfigurationswerkzeuge nur primitive Typen zur Verfügung stellen (falls überhaupt). Aber die Lösung oben ermöglicht es andere, komplexere Typen zur verwenden.

Zeitintervalle sollten eine eigene einheitsunabhängige Klasse haben:

```
public interface Duration {
  long toMillis();
}
```

Die entsprechende Einstellung würde dann so aussehen:

```
public class DurationSetting
  implements Setting<Duration> {
  ...
  }
```

Es ist also genau umgekehrt wie mit den Typen. Bei den meisten Konfigurations-Frameworks definiert man Typen für Einstellungen dort wo man sie verwendet (*getLong()*, usw.), obwohl der Typ eine Eigenschaft der Einstellung ist, aber die Einheiten behandelt man als wären sie eine Eigenschaft der Einstellung, obwohl die genau umgekehrt nur bei der Nutzung festgelegt werden muss (*toMillis()*, usw.).

Mit der *Duration*-Klasse legt der Entwickler selbst fest, in welcher Einheit das Zeitintervall verwendet wird, unabhängig davon wie man es in der Konfiguration angegeben hat. Die Konfigurationsdatei könnte z. B. folgende Einträge enthalten:

mysetting = 6h
mysetting2 = 65s

Es liegt nur am *Serializer* für die Klasse *Duration*, wie man Ausdrücke wie „6h“ oder „64s“ parsen und generieren kann.

Zusätzliche Features

Die oben definierten Ansätze erlauben auch einige interessante zusätzliche Features.

Die Einstellungen könnten sich selbst validieren, damit man keine falschen Werte einstellen kann. Dazu braucht man nur eine zusätzliche Methode:

```
public interface Setting<T> {
  ...
  void validate(T value) throws ValidationException;
}
```

Und es ist selbstverständlich, dass die *Configuration Implementation* diese Methode auch in der *set*-Methode verwendet.

Ein anderes Feature wäre, alle Einstellungen überall durch CDI [3] einfach auflisten zu können, wenn man die einzelnen Einstellungen entsprechend annotiert:

```
@Produces
public static final MY_SETTING =
  new DurationSetting(...);
```

Dann können alle Einstellungen so injiziert werden:

```
@Inject
public Instance<Setting<?>> settings;
```

Fazit

Java-Anwendungen haben meistens eine *Map*-ähnliche Konfigurationsschnittstelle. Diese haben das Problem, dass sowohl die Schlüssel als auch die Werte nicht typsicher sind.

Dieser Artikel zeigt, dass es durch die Verwendung von *Java Generics* ganz leicht ist, ein typsicheres Framework zu schaffen, um einfach Einstellungen einlesen und schreiben zu können, mit vielen Möglichkeiten, auch zusätzliche Features einzubauen.

Referenzen

[1] Oracle *Class Properties*, http://docs.oracle.com/javase/8/docs/api/java/util/Properties.html

[2] Apache Commons *Configuration*, http://commons.apache.org/proper/commons-configuration

[3] Oracle *Introduction for Contexts And Dependency Injection*, http://docs.oracle.com/javaee/7/tutorial/doc/cdi-basic.htm

Kurzbiographie

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

Herbstcampus

Wissenstransfer par excellence

1. – 4. September 2014, Nürnberg

Moderne Zeiten
Architekturen für eine Next Generation IT
Uwe Friedrichsen

Bullseye
Ein Blick auf Dart
Florian Hirsch

Schemafrei
PostgeSQL: Die NoSQL-Datenbank, die keiner kennt
Stephan Hochdörfer

Risiko Datensicherheit
End-to-End-Verschlüsselung von Anwendungsdaten
Peter Kirchner

Aller klar? Von wegen!
Die Fallstricke des Denkens und wie man sich gegen Fehlentscheidungen wappnet
Dr. Karl Kollischan

Vorsicht: Schuldenfalle
Was die IT aus der Finanzwelt lernen kann
Tammo van Lessen et al.

Built to last
Nachhaltige Software-Entwicklung
Frank Pientka

Trau, schau wem!
Absicherung gegenüber externen Diensten mit Hystrix
Alexander Schwartz

Meine Friseurin und die Agilität
Requirement Engineering beim Haareschneiden
Udo Wiegärtner

und vieles mehr...

Angemerkt

Android-Entwicklung mit AndroidAnnotations

von Zeeshan Ali

Fast jede native *Android*-Anwendung, unabhängig von der Größe und Funktionalität, enthält eine Menge Boilerplate-Code. *AndroidAnnotations* ist ein Framework, dass dieses Problem durch Reduzierung des Boilerplate-Codes löst und gleichzeitig die Android-Entwicklung intuitiver macht.

Genauso wie andere Frameworks und Programmiersprachen, hat auch Android seine Eigenheiten. Ein Problem während der Entwicklung ist, dass bestimmte Code-Blöcke zwingend geschrieben werden müssen, der sogenannte Boilerplate-Code: Zusätzlich viele Methodenaufrufe, Variablen-Deklarationen und *Event-Handler*, die weitschweifig sind. So muss jede *Android Activity* [1] z. B. zwingend die folgenden Code-Zeilen enthalten:

```
Public class MainActivity extends Activity {
  @Override
  Public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

Wir können dieses Problem mit *AndroidAnnotations* [2] lösen. AndroidAnnotations ist ein kleines Framework, das uns ermöglicht

- Boilerplate-Code zu reduzieren und
- weitschweifige Code-Zeilen mit Annotationen zu ersetzen.

Dadurch ist der Code besser lesbar, intuitiv und spätere Pflege fällt leichter.

Das Prinzip von AndroidAnnotations

Bei der Entwicklung mit AndroidAnnotations sind Android-Klassen, Methoden und Variablen usw. mit Annotationen markiert. Sie werden später von einem Annotation-Prozessor, der vom Framework zur Verfügung gestellt wird, bearbeitet und der Code automatisch generiert. Das Framework fügt auch *Views, Ressourcen* usw. durch *Dependency Injection* [3] in eine Activity oder Service-Klasse [1] ein.

Eine mit Annotationen markierte Klasse hat eine entsprechende generierte Klasse, die den Unterstrich ('_') am Ende des Namens hat. Diese von dem Framework generierten Klassen müssen in der Anwendung statt den annotierten Klassen erwähnt werden, z. B. im Android-Manifest [4] oder bei der Erzeugung einer Activity mit *Intent* [1].

Folgend wird mit einem Beispiel erklärt, wie man dieses Framework verwenden kann. Dieses Beispiel ist eine Android-Anwendung, die nur eine Activity enthält.

Android-Entwicklungsumgebung einrichten

Dieses Code-Beispiel setzt voraus, dass eine funktionierende Android-Entwicklungsumgebung mit der *Eclipse IDE* bereits vorhanden ist. Es gibt zahlreiche Quellen [5], die diesen Prozess detailliert beschreiben.

Die Einrichtung einer solchen Entwicklungsumgebung basiert grob auf folgenden Schritten.

1. Installation des Java Development Kits (JDK)
2. Installation des Android Software Development Kits (SDK)
3. Installation einer Variante der Eclipse IDE, die Android-Entwicklung unterstützt
4. Zuletzt die Installation des *ADT-Plugins* für Eclipse

Android-Beispiel-Anwendung ohne AndroidAnnotations (normale Entwicklung)

Der erste Schritt ist die Erzeugung eines neuen Android-Projektes im Eclipse mit einem passenden Namen, Package-Namen und einer leeren Activity-Klasse. Wenn die Anwendung läuft, erscheint diese Activity zuerst.

Layout-Erstellung

Der nächste Schritt ist die Erstellung eines *Layout-Files* im XML-Format. Dieses File beschreibt welche Benutzeroberflächen-Komponenten unsere Activity verwendet und wie sie dargestellt werden sollen.

```
<?xmlversion="1.0"encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <EditText
    android:id="@+id/myInput"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
  <Button
    android:id="@+id/myButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click me!"
  />
  <TextView
    android:id="@+id/myTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
</LinearLayout>
```

Dieses Layout erzeugt ein Eingabefeld und einen Button. In diesem Feld kann ein Benutzer Text wie z. B. seinen Namen eingeben. Beim Klick auf den Button wird eine Willkommensnachricht angezeigt.

Die Anwendungsfunktionalität in der Activity-Klasse

Der nächste Schritt ist die Anwendungsfunktionalität in der Activity zu schreiben. Wie bereits oben in der Layout-Beschreibung erwähnt, basiert diese Anwendung auf nur einer Activity, die ein Eingabefeld und einen Button enthält. Der Benutzer gibt einen Text ein und klickt auf den Button. Ein Event-Handler ist mit diesem Button verknüpft, der nach dem Klick die Eingabe liest und eine Willkommensnachricht ausgibt.

```
public class MainActivity extends Activity {
  private EditText myInput;
  private Button clickButton;
  private TextView textView;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //Initialize all the variables
    myInput = (EditText)findViewById(R.id.myInput);
    clickButton = (Button)findViewById(R.id.myButton);
    textView =
      (TextView)findViewById(R.id.myTextView);
    clickButton.setOnClickListener(
      new View.OnClickListener() {
        @Override
        public void onClick(View v) {
          String name= myInput.getText().toString();
          textView.setText("Hello "+name+"!");
        }
      }
    );
  }
}
```

Die Activity-Klasse im AndroidManifest eintragen

Der letzte Schritt ist die Eintragung der Activity-Klasse im *AndroidManifest-File*. Wenn eine Anwendung mit dem *Project Creation Wirzard* im Eclipse erzeugt wurde und die Activity durch diesen Wizard generiert, wird dieser Eintrag automatisch vom Framework gemacht und die Activity ist zusätzlich als Eingangspunkt der Anwendung markiert.

```
<?xmlversion="1.0"encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.androidannotationstest"
  android:versionCode="1"
  android:versionName="1.0">

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="21"
  />

    <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme"
    >
    <activity
      android:name=".MainActivity"
      android:label="@string/app_name">
      <intent-filter>
        <actionandroid:name="android.intent.action.MAIN"/>

        <categoryandroid:name=
          "android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
    </application>
  </manifest>
```

Die Anwendung ausführen

Die Anwendung kann jetzt mit Eclipse ausgeführt werden und sie läuft entweder auf einem Emulator-Gerät

oder einem Android-Gerät, das mit dem PC über USB verbunden ist. Das Emulator-Gerät kann durch den Android-SDK-Manager erzeugt werden. Folgendes Bild zeigt die Ausgabe:

Abbildung 1: Ausgabe ohne AndroidAnnotations

Entwicklung mit AndroidAnnotations

Jetzt entwickeln wir die gleiche Anwendung mit dem AndroidAnnotations-Framework. Damit werden die Vorteile dieses Frameworks gezeigt und auch die Unterschiede, die beachtet werden müssen.

Installation von AndroidAnnotations

Die AndroidAnnotations-Installation [6] enthält die folgenden Schritte.

1. Das Package *androidannotations-bundle-X.X.X.zip* [7] herunterladen und extrahieren.
2. Das File *androidannotations-api-3.0.1.jar* im *lib*-Verzeichnis des Projektes ablegen.
3. Aktivierung der Annotationen-Bearbeitung in den Eclipse-Projekteinstellungen wie folgt:
   - Das Projekt im Project Explorer auswählen und mit Rechtsklick das Context-Menü öffnen
   - *Properties* auswählen
   - Die Option *JavaCompiler* → *Annotation Processing* → *Enable Project Specific Settings* → *EnableAnnotation Processing* aktivieren
4. In gleicher Weise die *Factory Path*-Option unter dem Annotation Processing auswählen, die *Enable project specific settings*-Option aktivieren, auf *Add External JARs-Option* klicken und das File *android-annotations-X.X.X.jar* hinzufügen.
5. Auf *Apply* klicken und die Projekteinstellungen aktualisieren lassen.

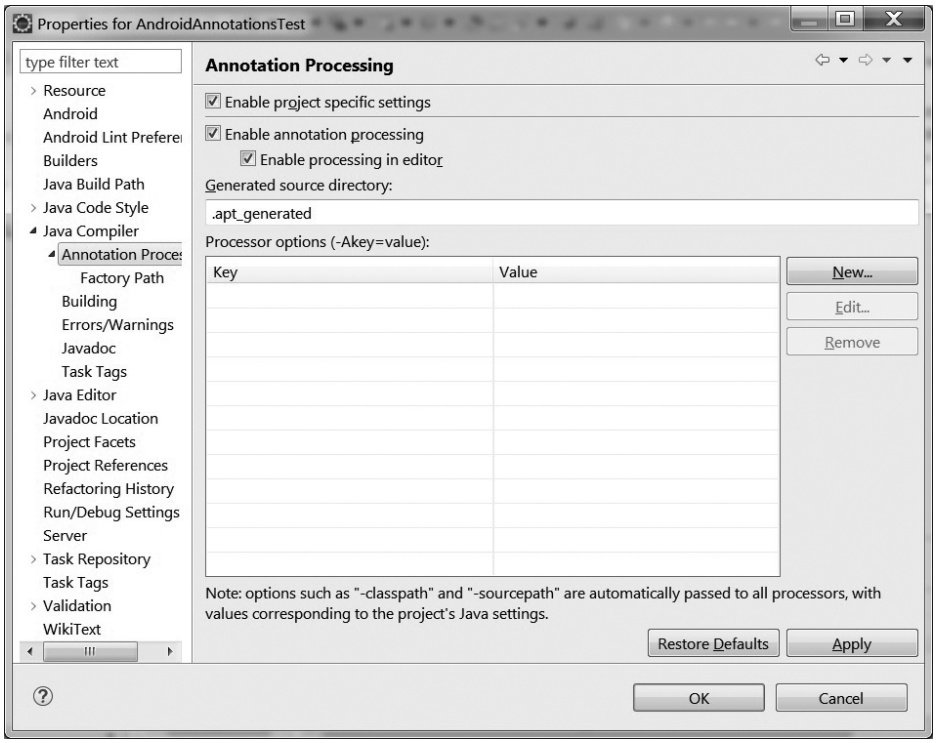

Abbildung 2a: Annotation-Processing im Eclipse aktivieren

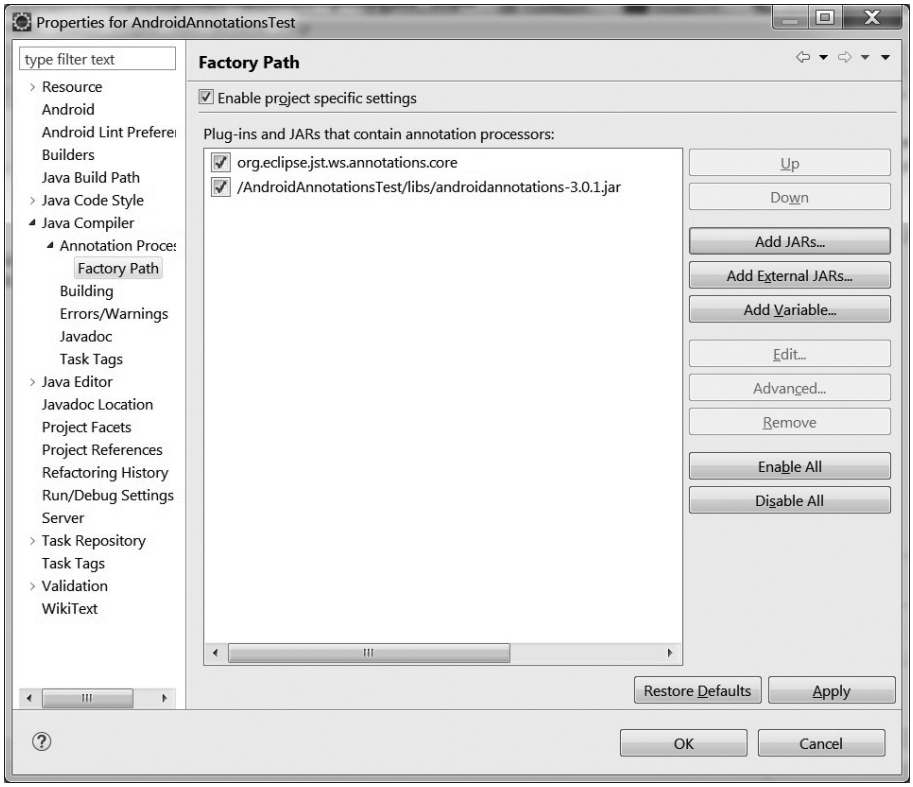

Abbildung 2b: Annotation-Processing im Eclipse aktivieren

Bei der Entwicklung mit AndroidAnnotations sind die Unterschiede nur in der Activity-Klasse und im AndroidManifest-File. Im Layout-File sind keine Änderungen erforderlich.

Activity mit AndroidAnnotations

Zuerst ist die Activity-Klasse mit Annotationen umzuschreiben.

```
@EActivity(R.layout.activity_main)
public class MainActivity extends Activity{

  @ViewById(R.id.myInput)
  EditText myInput;

  @ViewById(R.id.myTextView)
  TextView textView;
    @Click
  void myButton(){
    String name= myInput.getText().toString();
    textView.setText("Hello "+name+"!");
  }
}
```

Man bemerkt schnell, dass die Zahl der Code-Zeilen reduziert ist. Hier eine kurze Beschreibung dieser Annotationen:

- *@EActivity* markiert eine Java-Klasse als eine Activity und diese Klasse wird von dem Annotation-Prozessor des Frameworks bearbeitet. Die *setContentView()*- und *onCreate()*-Methoden müssen nicht aufgerufen werden.
- *@ViewById* injiziert ein View-Objekt durch seine ID, die entweder der Name des View-Objektes ist oder explizit eingegeben werden kann.
- *@Click* verknüpft eine *Listener*-Methode zu einem *View*-Objekt. Die View-Objekt-ID kann entweder explizit als Argument zu dieser Funktion eingegeben werden oder im Default-Fall ist der Name der Funktion die ID des View-Objektes.

Die Änderungen im AndroidManifest-File

Das AndroidAnnotations-Framework bearbeitet eine mit Annotationen markierte Activity-Klasse, wobei die Annotationen mit Code ersetzt werden und eine neue Klasse generiert wird. Der Name dieser Klasse ist der Name der markierten Klasse mit einem extra Unterstrich ('_') am Ende. Die Klasse *MainActivity* heißt jetzt z. B. *MainActivity_*. Es ist zu beachten, dass die Klassen mit Unterstrich als Endsilbe im Klassennamen in der Anwendung erwähnt werden müssen, wenn AndroidAn-

notations verwendet werden. Beispielsweise im AndroidManifest-File oder bei der Erzeugung einer Activity mit *Intent*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/
    apk/res/android"
  package="com.example.androidannotationstest"
  android:versionCode="1"
  android:versionName="1.0"
>
  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="21"/
  >
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
      <activity
        android:name=".MainActivity_"
        android:label="@string/app_name">
        <intent-filter>
          <actionandroid:name=
              "android.intent.action.MAIN"/>

          <categoryandroid:name=
              "android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
  </application>
</manifest>
```

Die Ausgabe dieser Version mit Annotationen ist dieselbe wie im Fall ohne Annotationen.

Vorteile von AndroidAnnotations

Wie das obige Beispiel gezeigt hat, erhalten wir die gleiche Ausgabe mit kürzerem Code, der gleichzeitig intuitiv ist. Das ist der Hauptvorteil der AndroidAnnotations. Dazu müssen wir die notwendigen Bibliotheken herunterladen, unsere Entwicklungsumgebung damit einrichten, unsere Activity-Klasse mit Annotationen umschreiben und sie mit dem Unterstrich am Ende ihres Namens in unserer Anwendung erwähnen. Das ist alles. Dieses Beispiel war sehr einfach, die Vorteile dieses Frameworks werden erst so richtig klar, wenn wir aufwendigere Anwendungen schreiben.

Nachteile von AndroidAnnoations

AndroidAnnotations haben auch ihre Grenzen. Ein Entwickler kann mit diesem Framework kürzere und intuitive Activities, Services usw. schreiben, aber die Layout-Files müssen genauso wie vorher zusätzlich geschrieben werden.

Abbildung 3: Ausgabe mit AndroidAnnotations

Fazit

Abschließend kann man sagen, dass AndroidAnnotations die Android-Entwicklung einfacher und intuitiv machen mit Übernahme von Boilerplate-Code und anderen Vereinfachungen, aber dazu muss man die Android-Entwicklung mit diesem Framework lernen.

Referenzen

[1] DEVELOPERS *Application Fundamentals Android Application Components*
http://developer.android.com/guide/components/fundamentals.html

[2] ANDROIDANNOTATIONS *AndroidAnnotationsFramework*
http://androidannotations.org

[3] WIKIPEDIA *DependencyInjection*
http://de.wikipedia.org/wiki/Dependency_Injection

[4] DEVELOPERS *AndroidManifest File*
http://developer.android.com/guide/topics/manifest/manifest-intro.html

[5] DEVELOPERS *Installing the Android SDK*
http://developer.android.com/sdk/installing/index.html

[6] ANDROIDANNOTATIONS *Installation Instructions*
https://code.google.com/p/androidannotations/wiki/Configuring

[7] ANDROIDANNOTATIONS *Package Download*
https://github.com/excilys/androidannotations/wiki/Download

Kurzbiographie

ZEESHAN ALI (zeeshan.ali@mathema.de) ist als Software-Entwickler bei der MATHEMA Software GmbH tätig. Nach dem Informatik-Studium hat er erste Erfahrungen im Bereich OOP mit Android-Entwicklung gesammelt. Aktuell beschäftigt er sich mit der Entwicklung und dem Testen von Java-Enterprise-Anwendungen. Neben dem Beruf ist er daran interessiert andere Kulturen kennenzulernen und seine Weltanschauung zu erweitern.

Lost in Translation

Reguläre Ausdrücke als englische Sätze

von Tim Bourguignon

Habt ihr schon einmal eine *Domain Specific Language* (*DSL*) geschrieben? „Wofür?", fragt ihr? Wie wäre es um den Aufbau von regulären Ausdrücken zu vereinfachen? Unter uns gesagt, reguläre Ausdrücke schreiben zu müssen, ist genau der Grund warum ich deren Verwendung vermeide. So entstand die Idee, einen funktionalen Programmierstil zu verwenden, um eine *Fluent-API* zu definieren, die reguläre Ausdrücke generiert. Denn eine derartige „sprechende Schnittstelle" kann man dann idealerweise auch drei Monate später immer noch einfach verstehen.

So habe ich mit Kollegen darüber gesprochen, und ein paar Stunden später hatte ich den ersten Entwurf einer solchen DSL auf dem Papier und die ersten C#-Code-Zeilen geschrieben.

Verbal Expression

Zuallererst habe ich existierende Projekte gesucht und dabei die *VerbalExpressions* [1] gefunden. Dieses Projekt wurde ursprünglich in *JavaScript* geschrieben, aber auch für verschiedene Sprachen portiert (*Ruby, C#, Python, Java, Groovy, PHP, Haskell, C++* und *Objective-C*). Zum Glück finde ich die Syntax dieser *VerbalExpressions* zu komplex, schwer zu verstehen und viel zu beschränkt. Und dass verschiedene Funktionen, abhängig vom Kontext, denselben Zweck haben, sind für mich K.o.-Kriterien. *VerbalExpressions* sind für mich eine gute Idee, aber nicht reif genug. Es gibt also noch genügend Raum für einen anderen Ansatz: die *SimpleExpressions* und die *MagicExpressions*.

SimpleExpressions

SimpleExpressions [2] war mein erster Versuch diese DSL zu schreiben. Ich wollte ganz nah an der englischen Sprache bleiben, keine Klammern verwenden und so weit wie möglich eine Art Gedicht schreiben; also die Syntax so zu schreiben, dass es sich wunderbar lesen lässt.

Um *SimpleExpressions* zu verwenden, muss man zuerst ein *SimpleExpression*-Objekt instanziieren und es als *dynamic* deklarieren. An diesem Objekt kann man nun Funktionen verketten. Die letzte Funktion muss ein *Generate()* sein, um dem Parser Bescheid zu geben, dass der Ausdruck zu Ende ist. Das generierte Pattern selbst erreicht man via der *Expression*-Eigenschaft. Man kann auch die Helper-Methoden *IsMatch(string)* und *Matches(string)* verwenden, um den „gewrappten" regulären Ausdruck direkt zu nutzen.

```
//Declaration
dynamic se = new SimpleExpression();

//The command
SimpleExpressions result = se.Letters.Generate();

//The result
var pattern = result.Expression;

//Matching
var success = result.IsMatch("something");
var matchCollection = result.Matches("something");
```

Die DSL

Gehen wir davon aus, dass wir ein *SimpleExpression*-Objekt zur Hand haben. Hier ist ein Teil der Funktionen, die man daran verketten kann: *Letters, Numbers,*

Alphanumerics, Text(string), One(char), OneOf(string), EitherOf(string), Maybe(string), NumberInRange(string), And(string), Except(string) etc.

Elemente zu gruppieren ist für reguläre Ausdrücke sehr wichtig. Damit können wir „benannte" oder „unbenannte" Gruppen definieren. Dies wird mit *SimpleExpression* mit der folgenden Kombination gemacht:

```
.Group.X.Y.Z.Together.As(name)
```

Die *Group*- und *Together*-Schlüsselwörter dienen als Klammerung wie „(" und „)" und *As(string)* beschreibt den potenziellen Namen für diese Gruppe.
Als Zusammenfassung ist hier ein einfaches, aber „echtes" *SimpleExpression*-Beispiel:

```
se.Group
   .Text("http")
   .Maybe("s")
   .Text("://")
.Together
.As("protocol")
.Generate();
// se.Expression == „(?<protocol>http(s)?://)"
```

Bittere Erkenntnis

Die *SimpleExpressions*-Funktionen sind tatsächlich sehr einfach und produzieren sehr gut lesbare Ausdrücke. Aus Sicht der Grammatik sind die kürzeren Ausdrücke sogar elegant. Aber je länger die Ausdrücke werden, desto schlimmer wird es, diese Ausdrücke laut zu lesen und fließend zu schreiben. Hier ist ein komplexeres Beispiel – ein Ausdruck zum Prüfen des „zweiten Teils" einer E-Mail:

```
string allowedChars = @"!#$%&'*+/=?^_`{|}~-";
se.Group
   .Letters.And(allowedChars).AtLleast(1)
   .Group
      .One(".")
      .Alphanumerics.And(allowedChars).AtLeast(1)
   .Together.As("dotAndAfter")
.Together.As("afterAt")
.Generate();
```

Darüber hinaus sind *SimpleExpressions* für Experten zu ungenau. Wenn man reguläre Ausdrücke schon beherrscht, wundert man sich bestimmt, was die genaue Übersetzung von manchen Funktion wie z. B. *Group.Together* oder *Except* ist. Baut es eine Klasse „[]", eine Gruppe „()" oder eine *non-capturing*-Gruppe „(?:)" auf? Wie macht man eine Alternative? Wie baut man *Back-References*?

Reguläre Ausdrücke sind sehr genau, viel genauer als unsere Sprachen. Wir können eigentlich kein englisches Konzept eins-zu-eins umsetzen.

Was ist dann mit SimpleExpression?

SimpleExpressions sind zwar sehr „chic", es gibt aber leider so viele kleine Unschönheiten, sodass das ganze Projekt nicht ernst genommen werden kann.

Es gibt mehrere Möglichkeiten, um es besser zu machen. Klammern verwenden, manche Funktionen in eine Programmierungsreihenfolge statt in eine grammatische Reihenfolge „umdrehen". Kurz: weniger lesbar, aber dafür genauer.

Ein guter Freund, Ghusse [3], hat mir dabei geholfen die *SimpleExpressions*-DSL theoretisch abzubilden. Nach einiger Zeit hat er Lust bekommen einen anderen Ansatz auszuprobieren – so wurden die *MagicExpressions* geboren.

Zusammen haben wir seitdem die *Magex* weiterentwickelt. Dieses OpenSource-Projekt (MIT license) findet ihr auf *Github* [4].

MagicExpression

Kurz gesagt, die *MagicExpression* trennt sich von dem dynamischen Ansatz der *SimpleExpressions* und verwendet dafür eine vollständige *Fluent-API*. Die *Magex* ersetzen einen Teil der Funktionen mit weniger-lustigen-aber-genaueren funktionalen Paradigmen und bleiben damit deutlich einfacher als die *SimpleExpressions*. So können die *Magex* viel komplexere Ausdrücke abbilden.

Die Basis

MagicExpression verlinkt man am Besten über *NuGet* [5]:

```
Install-Package MagicExpression
```

Magex-Objekt instanziieren:

```
var magicWand = Magex.New();
```

Das war es schon. Keine unnötige *.Generate()*-Funktion zum Schluss, der Ausdruck wird direkt *on-the-fly* interpretiert und jede Funktion gibt ein *Magex* zurück, sodass man es direkt weiterverwenden kann.[1]

Anstatt alle Funktionen der *MagicExpressions* zu zeigen, schauen wir uns lieber ein paar Beispiele an. Mehr Informationen über die API findet man direkt auf der Homepage des *Github*-Projekts [4].

[1] "MagicExpression for muggles" (© Ghusse)

Beispiel 1: Floating Point Nummer

```
var magicWand = Magex.New();

magicWand
    .Character('-')
        .Repeat.AtMostOnce()
    .CharacterIn(Characters.Numeral)
        .Repeat.Any()
    .Character('.')
    .CharacterIn(Characters.Numeral)
        .Repeat.AtLeastOnce();

// liefert den regulären Ausdruck
// -?[0-9]*\.[0-9]+
var floatingPointNumberDetector =
   new Regex(magicWand.Expression);
// Akzeptiert "1.234", "-1.234", "0.0"
// Akzeptiert nicht "0" "1,234", "0x234", "#1a4f66"
```

Es sollte schon sichtbar sein, dass sich *MagicExpressions* näher an eine Programmiersprache anlehnen als Shakespearsche Prosa. Es ist nach wie vor einfach zu lesen, aber in der Sprache selbst ist etwas mehr *Boilerplate Code* sichtbar.

Beispiel 2: XML-Tag

Gruppen kann man so oder so machen. Über die Funktion *Group()* bauen wir eine *non-capturing*-Gruppe; ansonsten über *Capture()*- und *CaptureAs()*-Funktionen. Die *Backreference()*-Funktion kann im Zusammenspiel mit *CaptureAs()* verwendet werden.

Hier ist ein Beispiel wie man mit einem *Capture* und einer *BackReference* auf eine predefinierte Gruppe zurück referenzieren kann:

```
var magicWand = Magex.New()
   .Character('<')
   .CaptureAs(
      "tag",
      x => x.CharacterNotIn('>').Repeat.AtLeastOnce()
   )
   .Character('>')
   .Character().Repeat.Any().Lazy()
   .String("</")
   .BackReference("tag")
   .Character('>');

var badHtmlTagDetector =
   new Regex(magicWand.Expression);

// Akzeptiert "<strong>hello world</strong>"
   oder "<h1>A title</h1>"
// Akzeptiert nicht "<h1>A tag mismatch</strong>"
```

Beispiel 3: URL

Um eine URL, die mit *http* oder *ftp* beginnt, zu vergleichen, könnte man Folgendes schreiben:

```
var magicWand = Magex.New();

magicWand.Options = RegexOptions.IgnoreCase;

const string allowedChars = @"!#$%&'*+/=?^_`{|}~-";

magicWand.Alternative(
   Magex.New().String("http"),
   Magex.New().String("ftp")
)
.Character('s').Repeat.AtMostOnce()
.String("://")
.Group(Magex.New().String("www."))
   .Repeat.AtMostOnce()
.CharacterIn(
   Characters.Alphanumeric, allowedChars
);
```

Es gibt mindestens drei Sachen, die man beachten sollte. Zuerst die Optionen auf die *Magex* selbst, um Groß- und Kleinschreibung zu ignorieren. Dann den doppelten Parameter der Funktio*n CharacterIn(params char[]),* und zuletzt die *Alternative()*-Funktion mit den zwei „Sub-Ketten“ darin.

Beispiel 4: IP Adresse

So würde man eine IP-(v4)-Adresse bestimmen:

```
Magex.New()
   .Builder.NumericRange(1, 255).Character('.')
   .Builder.NumericRange(0, 255).Character('.')
   .Builder.NumericRange(0, 255).Character('.')
   .Builder.NumericRange(0, 255);
```

Über diese *.Builder*-Eigenschaft kann man auf vordefinierte Funktionen zugreifen wie *.NumericRange().* Oder auch die Funktion *Literal(string)* aufrufen, um einen bestehenden regulären Ausdruck zu überreichen.

Beispiel 5:

Können Sie erraten, was folgende Ausdrücke machen?

```
Magex.New()
   .Character('#')
   .Alternative(
      Magex.New().CharacterIn(
         Characters.Numeral,
         "abcdefABCDEF"
      )
      .Repeat.Times(6),
      Magex.New().CharacterIn(
         Characters.Numeral,
         "abcdefABCDEF"
      )
         .Repeat.Times(3)
      .EndOfLine());
```

Und dieser hier?

```
Magex.New()
  .Character('0')
  .CharacterIn("xX")
  .CharacterIn(
    Characters.Numeral,
    "abcdefABCDEF"
  )
  .Repeat.Times(6);
```

Natürlich haben Sie es verstanden: Beide Ausdrücke stimmen mit Hexadezimalzahlen überein. Der Erste mit einem Farbenformat #123abc oder #1AC, und der Zweite mit dem Format 0x123aBc.

Und nun?

Mit den *MagicExpressions* haben wir ein bisschen „Spaß" für eine Menge Genauigkeit getauscht und damit eine sehr stabile Basis geschaffen.

Es gibt immer noch kleine Verbesserungen, die wir einbauen wollen, oder Punkte, mit denen wir noch nicht ganz zufrieden sind. Aber im Großen und Ganzen ist es schon eine schöne API.

Hat jemand Ideen, wie wir die *MagicExpressions* weiterentwickeln können, oder hat Lust sie in einer anderen Sprache zu implementieren? Dann möge er uns doch kontaktieren oder direkt auf *Github* „forken".

Ein Feature untersuchen wir übrigens gerade: das *Reverse-Engineering* von regulären Ausdrücken. Wenn das klappt, wäre das nicht magisch?

Referenzen

[1] Ionut Vilcu, Mihai et al. *VerbalExpression*
https://github.com/VerbalExpressions

[2] Bourguignon, Tim *SimpleExpression*
https://github.com/Timothep/SimpleExpressions

[3] Gautreau, Guillaume *Ghusse's Blog*
http://ghusse.com

[5] Bourguignon, Tim; Gautreau, Guillaume *MagicExpression*
https://github.com/ghusse/MagicExpression

[5] NuGet *Gallery*
https://nuget.org

Kurzbiographie

Timothée Bourguignon ist als Senior Developer für die MATHEMA Software GmbH tätig. Sein Spezialgebiet ist die Desktop- und Web-Programmierung mit dem .NET-Framework. Daneben beschäftigt er sich mit den Neuerungen der .NET-Welt und deren Communities. Als Agilist liegt sein Schwerpunkt auf agilen Methoden, guter Teamarbeit und hoher Arbeitsqualität. Hierbei setzt er insbesondere auf die Philosophie des Software Craftmanships.

Herbstcampus

Wissenstransfer par excellence

1. – 4. September 2014, Nürnberg

Moderne Zeiten
Architekturen für eine Next Generation IT
Uwe Friedrichsen

Bullseye
Ein Blick auf Dart
Florian Hirsch

Schemafrei
PostgeSQL: Die NoSQL-Datenbank, die keiner kennt
Stephan Hochdörfer

Risiko Datensicherheit
End-to-End-Verschlüsselung von Anwendungsdaten
Peter Kirchner

Aller klar? Von wegen!
Die Fallstricke des Denkens und wie man sich gegen Fehlentscheidungen wappnet
Dr. Karl Kollischan

Vorsicht: Schuldenfalle
Was die IT aus der Finanzwelt lernen kann
Tammo van Lessen et al.

Built to last
Nachhaltige Software-Entwicklung
Frank Pientka

Trau, schau wem!
Absicherung gegenüber externen Diensten mit Hystrix
Alexander Schwartz

Meine Friseurin und die Agilität
Requirement Engineering beim Haareschneiden
Udo Wiegärtner

und vieles mehr...

Des Programmierers kleine Vergnügen

Qual der Wahl

von Michael Wiedeking

Wenn man beliebig große Zahlen implementieren will, gilt es eine geeignete, effiziente Repräsentation zu finden, die gleichermaßen schnell wie speicherschonend ist. Und das ist nicht ganz so einfach, wie es vielleicht den Anschein haben mag.

Wenn ein Software-Entwickler etwas implementieren muss, dann steht er oft vor dem Problem Entscheidungen treffen zu müssen. Derweil viele Entscheidungen zwischen verschiedenen Alternativen, solange sie zielführend sind, kaum eine Rolle spielen, gibt es Entscheidungen, die einen ein Leben lang verfolgen, weil sie sich nicht mehr rückgängig machen lassen.

Bei dem Entwurf einer Bibliothek für beliebig lange Zahlen steht man beispielsweise vor dem Problem, dass man sich Gedanken über die Darstellung eben dieser Zahlen machen muss. Sollen die Zahlen im Zweierkomplement dargestellt werden? Sollen diese dann möglicherweise aus einer Anzahl von Wörtern bestehen, die immer eine Zweierpotenz ist? An welcher Stelle soll das niederwertigste Wort stehen, wo das höchstwertigste? Gibt es alternative Darstellungen?

Sind die Ziele, die es zu erreichen gilt, klar definiert, dann ergeben sich einige Entscheidungen wie von selbst. Will man mit beliebig langen Zahlen arbeiten, sollen sie möglichst wenig Speicher benötigen, denn die Zahlen sind ja beliebig lang, und – was vielleicht noch viel wichtiger ist – extrem schnell sein. Dementsprechend sollte die eine oder andere Entscheidung leicht fallen. Allerdings gilt, wie man später sehen wird, auch hier: Man kann nicht alles haben.

Modell

Während es in den letzten Vergnügen darum ging, mit doppelt breiten Wörtern zu rechnen, gestaltet sich die Aufgabe mit beliebig langen Wörtern schon alleine deswegen etwas komplizierter, weil die Operanden einer Binäroperation eine unterschiedliche Länge haben können. Von der Idee her bleibt aber erst einmal alles beim Alten. Man betrachtet die zu einer Kette zusammengefassten Wörter als einzelne Ziffern. Anstatt im Dezimalsystem mit den Ziffern 0 bis 9 zu arbeiten, arbeitet man nun etwa mit den Ziffern 0 bis $2^k - 1$. Dabei ist die Zweierpotenz dem Umstand geschuldet, dass man als Basis ein von dem Prozessor unterstütztes Wort wählt, also etwa ein 32- oder 64-Bit-Wort.

Damit hat man es mit Zahlen zur Basis $B = 2^k$ zu tun, wobei die nachfolgende, fünfstellige Zahl bestehend aus den Ziffern z_0 bis z_4 analog zum Dezimalsystem in der Form

$$z_4 z_3 z_2 z_1 z_0$$

den Wert

$$z_4 \cdot B^4 + z_3 \cdot B^3 + z_2 \cdot B^2 + z_1 \cdot B^1 + z_0 \cdot B^0$$

hat, wobei eventuell das Ganze noch durch ein Vorzeichen ergänzt wird.

Das Rechnen funktioniert dann in den meisten Fällen so, wie wir es in der Schule gelernt und in den vorangegangenen Vergnügen schon oft genug praktiziert haben.

Endigkeit

Die erste Frage, die sich nun stellt, ist, wie denn diese Ziffern organisiert werden sollen. Im Grunde ist es egal, aber entscheiden muss man sich trotzdem. Die Ziffern werden natürlich in der korrekten Reihenfolge, beispielsweise in einem Array, nebeneinander angeordnet. Allerdings lassen sie sich nach auf- oder absteigender Wichtigkeit ablegen. Wird etwa die Zahl 0x12345678 in einem Byte-Array abgelegt (wobei die Ziffern dann von 0 – 255 gehen), dann lässt sich das mit den folgenden zwei Varianten darstellen:

little-	$a[3]$	$a[2]$	$a[1]$	$a[0]$
endian	0x12	0x34	0x56	0x78
big-	$a[0]$	$a[1]$	$a[2]$	$a[3]$

Die erste Variante macht einen sehr natürlichen Eindruck, denn wenn man die Indizes von rechts nach links interpretiert, stellt sich das wie in unserem Dezimalsystem dar. Bei dieser Darstellung, bei der die niederwertigste Ziffer am Index 0 steht (*little-endian*), spiegelt der Index die Potenz an der entsprechenden Stelle dar; der Wert der Zahl ist demnach

$$a[3] \cdot B^3 + a[2] \cdot B^2 + a[1] \cdot B^1 + a[0] \cdot B^0$$

wobei in obigem Byte-Beispiel B = 256 ist.

Nicht dass dies ein unschlagbarer Vorteil wäre, denn in umgekehrter Reihenfolge, in der die höchstwertigste Ziffer am Index 0 steht (*big-endian*), ist es nicht wirklich schwieriger den Wert zu berechnen. Der Zugriff ähnelt dann auch mehr dem eines Strings, wo auch die Leserichtung durch aufsteigende Indizes repräsentiert wird. Allerdings gibt es einen nicht zu unterschätzenden Performanzvorteil: Da sich Überträge immer auf die nächst höhere Stelle auswirken, betreffen sie hier in dieser Darstellung immer die Stelle mit dem niedrigeren Index. Bearbeitet man nun Operationen in einer Schleife, so ist die Stelle mit dem Index 0 die letzte, die bearbeitet werden muss. Damit ist es möglich, die Schleife in der Form

for (**int** $i = n - 1$; $i \geq 0$; i––)

zu schreiben, was eine Nuance schneller ist, als die umgekehrte Schleife mit der Abbruchbedingung $i < n$, weil zu deren Prüfung keine Zahl n geladen werden muss.

Format

Die nächste Frage, die sich stellt, ist die Frage nach dem Format. So hat die Darstellung von Zahlen im 2er-Komplement durchaus ihre Vorteile. So lassen sich beispielsweise additive Operationen trivial implementieren, und der Zugriff auf einzelne Bits ist analog zu dem, was man von den Maschinenwörtern gewohnt ist.

Diese Darstellung mag suggerieren, dass die Anzahl der verfügbaren Bits eine Zweierpotenz sein muss. Aber das gilt traditionell mehr für Register als für beliebig große Zahlen. Das kann man sich auch einfach klar machen, indem man sich der schriftlichen Addition bedient, so wie man es in der Schule gelernt hat:

1000	1234	9999
+ 3	+ 7	+ 1
100**3**	12**41**	**10000**

Immer nur dann, wenn ein Übertrag stattfindet, vergrößert sich der Aufwand über den kleinen Operanden hinaus, wobei die Wahrscheinlichkeit, dass es einen weiteren Übertrag gibt, immer weiter abnimmt. Das scheint also dafür zu sprechen, sämtliche Elemente nur so groß zu machen wie unbedingt nötig. Das ist besonders dann wichtig, wenn man es anstatt mit linearem Aufwand wie bei der Addition mit quasi-quadratischem Aufwand wie bei der Multiplikation zu tun hat.

Apropos Multiplikation: Wenn es tatsächlich mal um echte Zeitersparnis geht, stellt man fest, dass viele der effizienten Algorithmen annehmen, dass die beteiligten Operanden nie negativ sind. Beispielsweise gibt es Algorithmen für die Multiplikation, die deutlich weniger Zeit brauchen, als die Methode, die wir im Schulunterricht gelernt haben; und die gestalten sich einfacher, wenn man es dabei nicht mit negativen Zahlen zu tun hat.

Dies legt nun nahe, den Betrag der Zahlen vorzeichenlos und das Vorzeichen separat zu speichern. Der Vorteil bezüglich der Algorithmen liegt auf der Hand: Die Vorzeichen können getrennt behandelt und anschließend kann mit nicht negativen Operanden gearbeitet werden. Dafür werden die einfachsten Dinge durchaus kompliziert: Wird etwa ein bestimmtes Bit benötigt, so muss es – wenn die Zahl negativ ist – korrigiert werden, was erforderlich macht herauszufinden, ob es irgendwo „vor“ dem gesuchten Bit ein Null-Bit gibt.

Modus Operandi

Hat man sich einmal für die Darstellungsform entschieden, so gilt es nur noch die Operationen effizient umzusetzen. Das bekannte schriftliche Rechnen soll dafür ersteinmal genügen. Aber auch hier gilt es, die richtigen Entscheidungen zu treffen.

Entscheidet man sich etwa für vorzeichenlose Zahlen, so benötigt man auch vorzeichenlose Operationen. Die sind aber nicht überall verfügbar. In Java beispielsweise stehen einem nur vorzeichenbehaftete arithmetische Funktionen zur Verfügung. Darüber hinaus benötigt man Rechenoperationen, bei denen man an einen etwaigen Überlauf kommt (bei der Addition ist das höchstens ein Bit, bei der Multiplikation aber ein ganzes Wort).

Auf einer 64-Bit-Maschine scheint es deshalb eine gute Idee zu sein, als Zifferngröße ein 32-Bit-Wort zu wählen und sämtliche Operationen in 64-Bit zu berechnen. Damit enthält das niederwertige 32-Bit-Wort das gewünschte Teilergebnis und das höchstwertige 32-Bit-Wort den Übertrag. Es ist schon schade, dass einem in den Hochsprachen diese Funktionalität in der Regel nicht zu Verfügung steht. Und so wird viel Zeit verplempert etwas zu emulieren, was man mit einem *Assembler* geschenkt bekäme.

Der eigentliche Spaß würde jetzt anfangen, aber leider reicht an dieser Stelle der Platz nicht aus. So werden sich die nächsten Vergnügen mit den diversen Freuden beschäftigen, die durch die unterschiedlichen Möglichkeiten entstehen. Und apropos divers: Vielleicht kommt dann auch endlich die Division zum Zuge.

Kaffeesatz

Sie haben Ihr Ziel erreicht

von Lisa-Marie Wehlmann-Wiedeking

Endlich haben auch in Bayern die lang ersehnten und absolut wohlverdienten Sommerferien begonnen. Auch wenn das schwer vorstellbar sein mag, kann man sich in sechs Wochen schon anfangen zu langweilen.

Allerdings ist das ein sehr kurzweiliges Unterfangen, spätestens wenn es auf das nächste Schuljahr zugeht, die Materialien für eine weitere Klassenstufe gekauft werden, gerät man unter Zeitdruck. Wahrscheinlich schaltet sich bei Schülern ab dem ersten Ferientag einfach ein wichtiger Teil des Hirns ab. Man kann dann ja auch spontan nicht mehr Kopfrechnen. Auf jeden Fall sind die Schule und der Stress ganz schnell vergessen und überraschen einen im Herbst um so mehr. Wenn man den Sommer genießt, werden solch banale Dinge vom Endorphin schon mal rausgeschmissen.

Da wir hierzulande immer mit den Letzten in die Sommerpause entlassen werden, ist es beinahe schon Gewohnheit den Sommer verpasst zu haben. Wenn Freibadwetter war, saß man in Bayern noch im Nachmittagsunterricht und konnte sich wegen dem Haufen Hausaufgaben auch keinen Ausflug am Abend mehr leisten. Deshalb sind viele Eltern so großzügig und nehmen ihre Kinder mit in den Urlaub. Denn selbst wenn auch am Urlaubsort kein Sommer mehr ist, tut ein Ortswechsel meist ganz gut. Und man kann die Schule noch schneller aus dem Bewusstsein verdrängen.

Abenteuerlich wird dann die Fahrt mit der Familienkutsche: Sack und Pack aufs Dach geschnürt, die Kinder auf den billigen Plätzen, und Verpflegung für ein ganzes Jahr quetscht jedem Fahrgast die Beine ab. Wenn dann noch ein Stau in der Ferne durch viele rote Punkte angekündigt wird, kann das schon mal die Stimmung drücken. Besonders dann, wenn die Lady in dem kleinen Kästchen mit dem besten Ausblick erst *nach* der letztmöglichen Abfahrt von der Autobahn herausposaunt, dass eine Verkehrsbehinderung bevorsteht, möchte man sie aus dem Fenster werfen. Aber sie hat die Karte und sie sagt wo es lang geht.

Böse Zungen mögen behaupten, dass es heutzutage auch nicht besser klappt, als zu den Zeiten, in denen die Ehefrau noch mit der Papierkarte auf dem Beifahrersitz saß und darauf drängte, nach dem Weg zu fragen. Zwar fragt das Navi nicht nach dem Weg, aber es beharrt meist vehement auf einer Route. Wer hatte nicht schon mal eine Unterhaltung mit der freundlichen Dame im Navigationssystem, die sehr einseitig verlief?

„Wenn möglich bitte wenden!“

Die Straße führt ins Nichts und man kann froh sein, dass es keinen Gegenverkehr gibt, der einen von der Straße abbringt und die Klippen hinunter drängt.

„Wenn möglich bitte wenden!“

Wenn es möglich wäre, würde man es ja machen. Aber wieso schickt sie einen nicht gleich in die richtige Richtung?

Auch Routen, die Fußgängerwege mit einbeziehen, Gassen auswählen, in denen das Auto unweigerlich stecken bleibt oder vorbei an der neuen Umgehungsstraße mit der Kirche um viele bis dahin unbekannte Dörfer, weil man nicht die neueste Version einer Navigations-DVD besitzt. Es ist auch schon vorgekommen, dass sich das Auto samt Insassen plötzlich nach dem vorgeschriebenen Abbiegen in einem See wieder fand. Wahlweise auch gegen eine Mauer gedrückt, die da vorher wohl nicht stand.

Allerdings zeigt das nur wieder, dass es nicht glimpflich ausgehen kann, wenn der Mensch blind den Computern folgt, denn hinter denen stecken ja immer wieder Menschen. Das klingt beinahe philosophisch. Aber wenn ein Urlaub mit einer solchen Farce beginnt, sollte man sich ernsthaft überlegen, entweder auf den Orientierungssinn des Beifahrers oder das gute alte Nach-dem-Weg-fragen zu vertrauen, die freundliche Dame abzuschalten und nur hin und wieder auf dem Display abzugleichen, ob kleiner Pfeil und vorgegebene Route noch etwa in näherer Umgebung zueinander liegen. Oder auf eine andere Art des Reisens umzusteigen.

Oder aber man schaltet es dankbar aus, das Navi, und freut sich auf die Ferien.

„Sie haben Ihr Ziel erreicht.“

Kurzbiographie

Lisa-Marie Wehlmann-Wiedeking hat gerade die zehnte Klasse hinter sich und spricht in Sachen große Ferien aus langjähriger Erfahrung. Die Sechzehnjährige versucht, sich momentan – in den wohlverdienten Ferien – die Fähigkeit anzueignen, Entspannung zu tanken und zu einem späterem Zeitpunkt, nämlich in der Oberstufe, anzuwenden.

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)

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 Stefan Koospal
http://www.java.de (office@java.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: Frau Natalia Wilhelm
(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 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
Tieto Deutschland GmbH
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
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
15. – 17. September 2014, 1.250,– € (zzgl. 19 % MwSt.)

HTML5, CSS3 und JavaScript
22. – 25. September 2014, 1.650,– € (zzgl. 19 % MwSt.)

Anwendungsentwicklung mit der Java Enterprise Edition
2.150,– € (zzgl. 19 % MwSt.), 6. – 11. Oktober 2014

Fortgeschrittenes Programmieren mit Java
Ausgewählte Pakete der Java Standard Edition
13. – 15. Oktober 2014, 1.350,– € (zzgl. 19 % MwSt.)

Weiterführende Programmierung unter C#
Umstieg auf C# 4.5/5 und Einführung in fortgeschrittene Konzepte
24. – 27. November 2014, 1.350,– € (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

„Die Herausforderung, jeden Tag etwas Neues zu lernen, habe ich gesucht und bei MATHEMA gefunden."

Tim Bourguignon, Senior 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

Herbstcampus

Wissenstransfer par excellence

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.

1.– 4. September 2014 in Nürnberg

Moderne Zeiten
Architekturen für eine Next Generation IT
Uwe Friedrichsen

Bullseye
Ein Blick auf Dart
Florian Hirsch

Schemafrei
PostgeSQL: Die NoSQL-Datenbank, die keiner kennt
Stephan Hochdörfer

Risiko Datensicherheit
End-to-End-Verschlüsselung von Anwendungsdaten
Peter Kirchner

Aller klar? Von wegen!
Die Fallstricke des Denkens und wie man sich gegen Fehlentscheidungen wappnet
Dr. Karl Kollischan

Vorsicht: Schuldenfalle
Was die IT aus der Finanzwelt lernen kann
Tammo van Lessen et al.

Built to last
Nachhaltige Software-Entwicklung
Frank Pientka

Trau, schau wem!
Absicherung gegenüber externen Diensten mit Hystrix
Alexander Schwartz

Meine Friseurin und die Agilität
Requirement Engineering beim Haareschneiden
Udo Wiegärtner

und vieles mehr...

www.herbstcampus.de

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

Das Allerletzte

```
set -- `ls | grep u > /dev/null`
```

Dies ist kein Scherz!
Dieser Code-Abschnitt 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

1. – 4. September 2014
in Nürnberg