KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

06/2012

Jahrgang 5

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

Mandel-/Schröding-/ Heisenbug

Wenn wir es als Resultat unserer Software-Entwicklung mit Fehlern zu tun haben, dann handelt es sich in den meisten Fällen um Fehler, die sich reproduzieren und im Normalfall auch finden lassen. Gelegentlich machen sich aber auch Fehler bemerkbar, die schier unauffindbar scheinen oder ein anderes, seltsames Verhalten an den Tag legen.

Bei den „normalen“ Fehlern spricht man gelegentlich auch von Bohrbugs, die – in Anlehnung an das sehr einfache Bohrsche Atommodell – als eher deterministisch, oft verlässlich reproduzierbar und fast schon als banal aufzufinden gelten.

Gelegentlich hat man es aber mit Fehlern zu tun, die, sobald man sie einzugrenzen versucht, einfach nicht mehr aufzufinden sind. So tauchen diese Fehler nicht mehr auf, sobald man Print-Ausgaben eingebaut hat. Oder wenn man die Quellen mit den für den Debugger nötigen Optionen übersetzt, scheint der Fehler wie vom Erdboden verschwunden.

In diesem Fall spricht man von den Heisenbugs, die sich insbesondere auch in parallelen Programmen finden. Heisenberg könnte hier deshalb den Namensgeber geben, weil sich auch in der Quantenmechanik das Problem ergibt, dass ein Beobachter automatisch das Verhalten des zu beobachtenden Systems verändert, oder – gemäß seiner Unschärferelation – in einem solchen System abhängige Größen nicht in jeder Dimension beliebig genau bestimmt werden können.

Mandelbugs legen wie die namensgebende Mandelbrotmenge ein so chaotisches und nichtdeterministisches Verhalten an den Tag, dass auch die motiviertesten Reparaturversuche fehlschlagen oder weitere unvorhersehbare Fehler nach sich ziehen.

Abschließend seien noch die Schrödingbugs erwähnt. Angeregt durch Schrödingers Gedankenexperiment mit der Katze beschreiben sie Programme, die überraschender Weise laufen, obwohl der Entwickler im Nachhinein fest davon überzeugt ist, dass sie unmöglich hätten laufen können. Diese Fehler tauchen oft erst Ewigkeiten nach der Freigabe auf und machen im Extremfall ein Programm in ihrer Gesamtheit unbrauchbar.

Es gibt sicherlich noch viele andere Fehler, die ihre eigene Kategorie verdient hätten. Und der Allegorien und Metaphern gäbe es dann ja genug.

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

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 5, Nummer 6, Juni 2012
ISSN 1865-682X
Bookware – eine Initiative der MATHEMA Software GmbH

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

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

Inhalt

Editorial 3

Beitragsinfo 4

Inhalt 5

User Groups 26

Werbung 28

Das Allerletzte 31

Artikel

Typen wie du und ich

Typklassen vorgestellt 6

(Einheits-)Gewänder für den Androiden

Teil 3 – Selection 9

Nickeldioxid?

NIO.2 einmal vorgestellt 17

Kolumnen

… und der ganze Rest

Des Programmierers kleine Vergnügen 22

Märchen, Niederländisch und anderes

Deutsch für Informatiker 24

Naturkonstante

Kaffeesatz 25

Typen wie du und ich

Typklassen vorgestellt 6

von Rüdiger Keller

Typklassen wurden für *Haskell* entwickelt, um Typen in Klassen einzuteilen. Seit einiger Zeit werden sie auch in *Scala* immer öfter eingesetzt. Dieser Artikel zeigt, was Typklassen sind und welche Unterschiede es zu den aus *Java* und *Scala* bekannten Klassen gibt.

(Einheits-)Gewänder für den Androiden

Teil 3 – Selection 9

von Johannes Köstler

Die vierte Android-Version *Ice Cream Sandwich* wurde im Juni mit dem „Parsons School of Design's User Experience Award" in Gold als „Best System Experience" ausgezeichnet [1]. Einen nicht unwesentlichen Beitrag zu diesem Erfolg leistete dabei das überarbeitete Menükonzept. Nachdem der letzte Artikel die Ersetzung der Optionsmenüs durch die *Action Bar* beschrieben hat, widmet sich dieser Artikel der Ersetzung der Kontextmenüs durch die *Contextual Action Bar* und geht im Praxisbeispiel auf die Portierung der Contextual Action Bar auf Android-Versionen vor *Honeycomb* ein.

Nickeldioxid?

NIO.2 einmal vorgestellt 17

von Frank Goraus

Mit dem *JDK7* kamen mal wieder einige Neuerungen in Java hinzu. Eine davon ist eine neue API für die Arbeit mit Dateien und Dateisystemen, welche in diesem Artikel vorgestellt werden soll, und anhand kleiner Beispiele erläutert wird.

Typen wie du und ich

Typklassen vorgestellt

von Rüdiger Keller

Typklassen wurden für *Haskell* entwickelt, um Typen in Klassen einzuteilen. Seit einiger Zeit werden sie auch in *Scala* immer öfter eingesetzt. Dieser Artikel zeigt, was Typklassen sind und welche Unterschiede es zu den aus *Java* und *Scala* bekannten Klassen gibt.

Was sind Typklassen?

Ähnlich wie die Klassen aus Java beschreiben Typklassen Operationen auf Typen. Das heißt, alle Typen, die zu einer Typklasse gehören, bieten die zu dieser Typklasse gehörenden Operationen. Anders als Javas Klassen, haben Typklassen nichts mit Objektorientierung zu tun und auch Vererbung wird von Typklassen nicht unterstützt. Dies ist aufgrund ihrer haskellschen Wurzeln auch nicht verwunderlich, denn Haskell ist eine rein funktionale Programmiersprache.

Was Typklassen trotz dieser Einschränkungen interessant macht, sind zwei Eigenschaften: zum Einen erlauben sie ad-hoc Polymorphie und zum Anderen kann es zu einem Typen beliebig viele verschiedene Implementierungen einer Typklasse geben. Mit ad-hoc Polymorphie ist gemeint, dass man sich nicht schon beim Schreiben eines Typen entscheiden muss, zu welchen Typklassen er gehört, sondern man kann Typen auch im Nachhinein zu einer Typklasse hinzufügen. Dies ist möglich, weil ein Typ seine Typklassen nicht selbst implementiert. Stattdessen wird die Implementierung separat vorgenommen. Dies erlaubt es auch Typen aus fremdem Code in eigene Typklassen aufzunehmen. Die separate Implementierung der Typklassen ist auch der Grund, warum zu jedem einzelnen Typ beliebig viele Implementierungen einer Typklasse existieren können.

Ein Beispiel

Nachdem dies recht kompliziert klingt, soll die Theorie anhand von Code-Beispielen veranschaulicht werden. Da viele Leser mit Java vertrauter sein werden, als mit Scala, wird das Beispiel erst in Java und dann in Scala gezeigt. Dabei sollte auch deutlich werden, warum Typklassen bisher keine nennenswerte Verbreitung in Java gefunden haben.

Das Beispiel ist eine Typklasse *Printable*, welche eine einzige Methode *print* anbietet. Diese soll die gleiche Semantik wie die *toString*-Methode von *Object* haben: sie liefert eine String-Repräsentation der Instanz zurück. Betrachten wir zunächst die Definition der Typklasse.

```
public abstract class Printable<T> {
  public abstract String print(T t);
}
```

Sie ist eine einfach abstrakte Klasse *Printable* mit einem generischen Parameter *T* und der Methode *print*, welche eine Instanz von *T* entgegennimmt und einen String zurückliefert.

Zwei mögliche Implementierungen dieser Typklasse für den Typ *Integer* sehen folgendermaßen aus.

```
class IntegerIsPrintable extends Printable<Integer> {
  public String print(Integer i) {
    return Integer.toString(i);
  }
}

class IntegerIsPrintableHex extends Printable<Integer>{
  public String print(Integer i) {
    return Integer.toString(i, 16).toUpperCase();
  }
}
```

Diese zwei unterschiedlichen Implementierungen der gleichen Typklasse geben Integerzahlen einmal in dezimaler und einmal in hexadezimaler Notation zurück.

Als Nächstes betrachten wir eine Methode, welche von *Printable* Gebrauch macht, um Instanzen dieser Typklasse auf der Konsole auszugeben.

```
<T> void echo(T t, Printable<T> printable) {
  System.out.println(printable.print(t));
}
```

Diese Methode ist generisch und nimmt eine Instanz von *T* und eine Implementierung von *Printable* für *T* als Parameter entgegen. Die Implementierung der Typklasse wird verwendet um *t* in einen String umzuwandeln und auszugeben.

Leider ist der Aufruf dieser Methode in Java etwas umständlich, da die zu verwendende Implementierung der Typklasse immer explizit angegeben werden muss.

```
echo(23, new IntegerIsPrintable()); // gibt 23 aus
echo(42, new IntegerIsPrintableHex()); // gibt 2A aus
```

Und in Scala

Wie eingangs erwähnt, haben Typklassen in Scala schon eine größere Verbreitung erfahren. Seit Version 2.8 haben Typklassen auch in die Standardbibliothek Einzug gehalten. So ist *Numeric* eine Typklasse, welche beispielsweise von der *Collection*-Methode *sum* benutzt wird, um die Elemente der Collection aufzusummieren.

Scala bietet verschiedene Features, welche die Verwendung von Typklassen angenehmer und weniger umständlich als in Java machen. Dies sind hauptsächlich implizite Werte, implizite Konvertierungen und die sogenannte *Context-Bound*-Schreibweise. Das Scala-Äquivalent zur Definition und den zwei Implementierungen von *Printable* sieht wie folgt aus.

```
abstract class Printable[T] {
  def print(t: T): String
}

object IntIsPrintable extends Printable[Int] {
  def print(i: Int) = i.toString
}

object IntIsPrintableHex extends Printable[Int] {
  def print(i: Int) = Integer.toString(i, 16).toUpperCase
}
```

Dies ähnelt dem Java-Code noch sehr, auch wenn es vielleicht etwas kompakter ist. Nun werden jedoch einige Vorzüge Scalas ins Spiel gebracht. So werden Standardimplementierungen einer Typklasse typischerweise im Kompagnonobjekt als implizit verfügbare Werte definiert. Dies erleichtert die Nutzung der Typklasse, da der Compiler implizite Werte eines Typs standardmäßig im dazugehörigen Kompagnonobjekt sucht. Im Code sieht es dann folgendermaßen aus.

```
object Printable {
  implicit object IntIsPrtbl extends Printable[Int] {
    def print(i: Int) = i.toString
  }

  implicit object StringIsPrtbl extends Printable[String] {
    def print(s: String) = s
  }
}
```

Mit diesen impliziten Objekten, beziehungsweise Werten, sieht die Verwendung der Typklasse dann beispielsweise so aus:

```
def echo[T](t: T)(implicit printable: Printable[T]) =
  println(printable.print(t))

echo("Hallo Welt")
echo(42)
```

Die Methode *echo* hat in Scala eine zweite, implizite Parameterliste bekommen, in der die Implementierung der Typklasse übergeben wird. Da diese zweite Parameterliste als implizit gekennzeichnet wurde, kann sie beim Aufruf weggelassen werden, sofern der Compiler einen vom Typ passenden impliziten Wert findet. Da wir implizite Werte für *Int* und *String* im Kompagnonobjekt definiert haben, gelingt dies auch.

Geben wir eine implizite Konvertierung für Typen an, zu denen es eine Implementierung von *Printable* gibt, kann man die Verwendung der Typklasse noch weiter vereinfachen.

```
class PrintableCanPrint[T](t: T, printable: Printable[T]){
  def print: String = printable.print(t)
}

implicit def toPrintableCanPrint[T](t: T)(
  implicit printable: Printable[T]
)
  = new PrintableCanPrint(t, printable)
```

Die Methode *toPrintableCanPrint* gibt eine Instanz der Klasse *PrintableCanPrint* zurück. *PrintableCanPrint* hat eine *print*-Methode, welche wiederum die Methode *print* der dazugehörigen Typklasse aufruft. Macht man bei der Methode *echo* von dieser impliziten Konvertierung Gebrauch, sieht sie folgendermaßen aus.

```
def echo[T](t: T)(implicit printable: Printable[T]) =
  println(t.print)
```

Es ist damit also möglich, die Methode *print* direkt auf *t* aufzurufen. Der Scala-Compiler macht daraus den Aufruf *toPrintableCanPrint(t).print*.

Weiterhin gibt es nun die Möglichkeit beim generischen Parameter *T* die Context-Bound-Syntax [*T: Printable*] zu verwenden. Sie ist syntaktischer Zucker für die zuvor verwendete, zweite, implizite Parameterliste. Dabei gibt es den Unterschied, dass der dadurch generierte, implizite Parameter anonym ist, also nicht mehr direkt benannt werden kann. Da wir ihn aber nur für die implizite Konvertierung von *T* zu *PrintableCanPrint* benötigen, ist

dies auch gar nicht notwendig. Die Methode *echo* sieht nun wie folgt aus.

```
def echo[T : Printable](t: T) = println(t.print)
```

Damit zeigt sich, dass in Scala auch bei Verwendung von Typklassen, sehr kompakter und ohne Boilerplate auskommender Code geschrieben werden kann.

Abschließend betrachten wir noch die Implementierung von *Printable* für Arrays, die Elemente enthalten, die selbst eine Implementierung von *Printable* haben.

```
implicit def arrayIsPrintable[T : Printable] =
  new Printable[Array[T]] {
    def print(arr: Array[T]) = {
      arr.map(_.print).mkString("(", ", ", ")")
    }
  }
```

Hier wird eine implizite Methode statt eines impliziten Wertes verwendet, da ein generischer Parameter *T* für den Typ der Elemente des Arrays gebraucht wird. Über den Context-Bound [*T : Printable*] wird sichergestellt, dass eine Implementierung von *Printable* für die Elemente zur Verfügung steht. Zurückgegeben wird eine Instanz einer anonymen Subklasse von *Printable*, die *print* für alle Elemente des Arrays aufruft und die Ergebnisse zu einem String konkateniert. Zu guter Letzt sind hier noch einige Aufrufe von *echo* mit Arrays.

```
echo(Array(10, 20, 30))
echo(Array(Array(1), Array(1, 2), Array(1, 2, 3)))
```

Die Ausgabe sieht dann folgendermaßen aus.

```
(10, 20, 30)
((1), (1, 2), (1, 2, 3))
```


Kurzbiographie

Rüdiger Keller ist Diplom-Informatiker und seit Anfang 2007 als Software-Entwickler, Trainer und Consultant bei der MATHEMA Software GmbH angestellt. Er beschäftigt sich gerne mit neuen Entwicklungen im Java- und JEE-Umfeld. Zu seinen aktuellen Lieblingsthemen gehören Scala und GWT. Sein Blog zu Software-Entwicklungsthemen ist unter http://ruedigerkeller.blogspot.com zu finden.

Copyright © 2012 Bookware 1865-682x/12/06/001 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

3. – 6. September 2012, Nürnberg

Ausgeschlafen?
Mit BPMN Integrationsprozesse bereichern

JavaScript goes Enterprise
Mit JavaScript Business-Anwendungen erstellen

Schlankheitskur
Lean Webarchitecture with JSF 2.0, CDI & Co.

(Persistenz-)Abenteuer gefällig?
Eine Expedition in den NoSQL-Dschungel

Absturz unerwünscht
Architekturmuster für fehlertolerante Systeme

Apache Buildr
Die Mavenalternative

Besser Gits nicht
Best Practices mit GIT

Continuous Bugfixing
Wie man statische Code-Analyse einführt

Das muss man wissen!
Windows 8 für Entwickler

Leitern mit Stil
Eine Einführung in ScalaFX

Lucky Seven
Java Enterprise Edition 7

Software per Rohrpost
Kontinuierliche Software-Auslieferung über Delivery Pipelines

Scriptease
Was Sie schon immer über JavaScript wissen wollten, aber bisher nicht zu fragen wagten

Testing untestable code
Ideen und Anregungen um Legacy Code ein Schnippchen zu schlagen

Verbindungsprobleme
Offline Web-Anwendungen

(Einheits-) Gewänder für den Androiden

Teil 3 – Selection

von Johannes Köstler

Die vierte Android-Version *Ice Cream Sandwich* wurde im Juni mit dem „Parsons School of Design's User Experience Award" in Gold als „Best System Experience" ausgezeichnet [1]. Einen nicht unwesentlichen Beitrag zu diesem Erfolg leistete dabei das überarbeitete Menükonzept. Nachdem der letzte Artikel die Ersetzung der Optionsmenüs durch die *Action Bar* beschrieben hat, widmet sich dieser Artikel der Ersetzung der Kontextmenüs durch die *Contextual Action Bar* und geht im Praxisbeispiel auf die Portierung der Contextual Action Bar auf Android-Versionen vor *Honeycomb* ein.

Kontextmenüs

Bis zur Tablet-Variante (Honeycomb) von Android galt die Vorgabe, dass bei einem langen Klick auf ein dargestelltes Inhaltselement dessen Kontextmenü erscheint. Die auf dem Element ausführbaren Aktionen werden dabei als einfaches Text-Menü in einem Dialogfenster angezeigt (siehe Abbildung 1). Dieses überlagert weite Teile des Bildschirms, was den visuellen Bezug zum gerade dargestellten Inhalt stört und den Applikationsfluss schwerfällig erscheinen lässt. Hinzu kommt, dass in der Regel keine sichtbaren Anzeichen auf das Vorhandensein eines Kontextmenüs existieren und somit die Gefahr besteht, dass einige Funktionalitäten der Applikation gar nicht wahrgenommen werden.

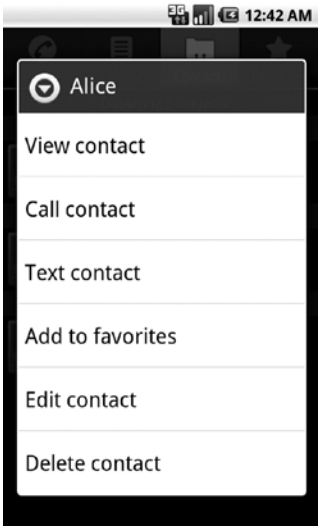


Abbildung 1: Klassisches Kontextmenü

Contextual Action Bar

Seit Honeycomb wird der lange Klick zum Starten des Aktions-/Selektionsmodus (*Contextual Action Mode*) verwendet. Dabei wird die sogenannte Contextual Action Bar eingeblendet, welche die normale Action Bar für die Dauer des Selektionsmodus ersetzt (siehe Abbildung 2). In ihr finden sich die kontextabhängigen Aktionen, die auf der Selektion ausgeführt werden können. Aktionen, für die der Platz in der Leiste nicht mehr ausreicht, werden analog zur Action Bar im *Action Overflow* platziert. Ganz links ermöglicht ein kleiner Button die Beendigung des Selektionsmodus. Alternativ kann jener auch über die „Zurück"-Taste abgebrochen werden.

Abbildung 2: Contextual Action Bar

Die Positionierung der Contextual Action Bar als Überlagerung der Action Bar adressiert die Schwachstellen der Kontextmenüs. Der Verzicht auf die Einblendung großer Dialogfenster lässt die Applikation zum einen

dynamischer wirken und zum anderen bleibt der visuelle Bezug aufrechterhalten, da der Anwender jederzeit sieht auf welchen Daten er operiert. Letzteres wird durch die Anzeige der Anzahl der selektierten Datensätze unterstützt. Der Einsatz von aussagekräftigen Symbolen hilft dem Anwender zusätzlich die Bedeutung der angebotenen Funktionalitäten schneller zu erfassen.

Selektion

Eine Selektion kann sowohl auf Text, auf ein einzelnes Inhaltselement oder auf Daten aus einer Gruppe von Daten in Listen- oder Rasterdarstellungen ausgeführt werden. Das Exklusivrecht des Selektionsmodus an der Langklickgeste soll dem Anwender bei der Bedienung entgegenkommen und die Steuerung von Applikationen besser vorhersehbar machen. Dies bedarf allerdings eines eindeutigen Hinweises auf das Vorhandensein des Selektionsmodus. Bei Mehrfachselektionen kann dies durch die Anzeige eines kurzen Hinweistextes in der Action Bar (siehe Abbildung 3, links) oder durch die Integration von Checkboxen (siehe Abbildung 3, rechts) geschehen. Bei letzterer Variante kann der Selektionsmodus dann auch zusätzlich zum langen Klicken durch die einfache Aktivierung einer Checkbox gestartet werden.

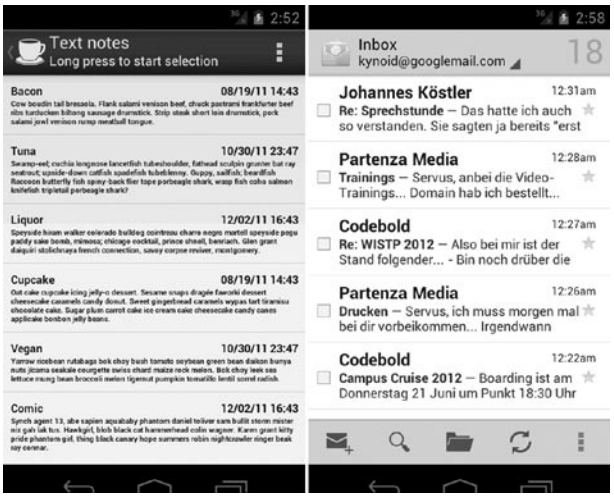


Abbildung 3: Hinweis auf das Vorhandensein des Selektionsmodus

Bei alleinstehenden Inhaltselementen gestaltet sich das Vorhaben schon weitaus schwieriger. Die Platzierung eines Hinweistextes in der Action Bar ist in aller Regel nicht geeignet, da sich die Action Bar auf den gesamten Inhalt der jeweiligen *Activity* beziehen soll. Im Einzelfall sollte über die Erstellung einer weiteren Detailansicht für das jeweilige Inhaltselement nachgedacht werden. Diese wäre dann ganz traditionell über einen einfachen Klick erreichbar und könnte die ausführbaren Aktionen in ihrer eigenen Action Bar anbieten. Die Möglichkeit der Textselektion muss nicht besonders gekennzeichnet werden, da es sich dabei um eine applikationsunabhängige Funktionalität handelt, die vom System für alle editierbaren Textfelder (siehe Abbildung 4, links) und für die in einer *WebView* dargestellten Inhalte (siehe Abbildung 4, rechts) automatisch bereitgestellt wird und auch schon vor Honeycomb durch langes Klicken erreichbar war.

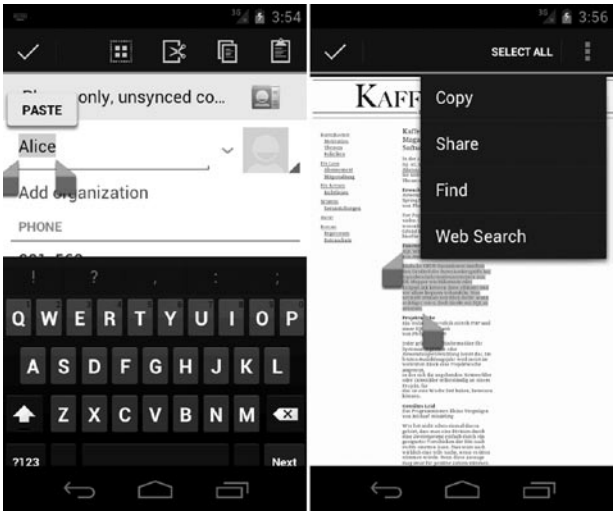


Abbildung 4: Textselektion

Einordnung in das neue Menükonzept

Das neue Menükonzept folgt einer hierarchischen Gliederung. Alle Funktionalitäten die für die gesamte Applikation oder für die jeweilige *Activity* gültig sind, gehören in die Action Bar, die die herkömmlichen Optionsmenüs ersetzt. Beispiele dafür sind eine generelle Suchfunktion oder die Erstellung eines Datensatzes, der den gerade Angezeigten entspricht. Auch die klassischen Kontextmenüs haben mit Honeycomb ihre Gültigkeit verloren. An ihre Stelle tritt der Selektionsmodus der Contextual Action Bar. Die kontextabhängigen Aktionen, wie das Kopieren von Text oder das Löschen von Datensätzen, beziehen sich immer auf die im Selektionsmodus ausgewählten Elemente.

Neben den zwei altbekannten Menüs ist seit Honeycomb das sogenannte Popup-Menü Teil der Android-API, das an dieser Stelle nur der Vollständigkeit halber erwähnt wird. Ein Popup-Menü erscheint unterhalb oder oberhalb der *View*, die es aufgerufen hat, und nimmt im Gegensatz zu Dialogen nur so viel Platz ein, wie es wirklich benötigt. Ein Popup-Menü soll ausdrücklich nicht für kontextabhängige Aktionen verwendet werden. Stattdessen kann es als Spinner-Ersatz

(siehe Abbildung 5, links) oder zur genaueren Spezifizierung einer Aktion verwendet werden (siehe Abbildung 5, rechts).

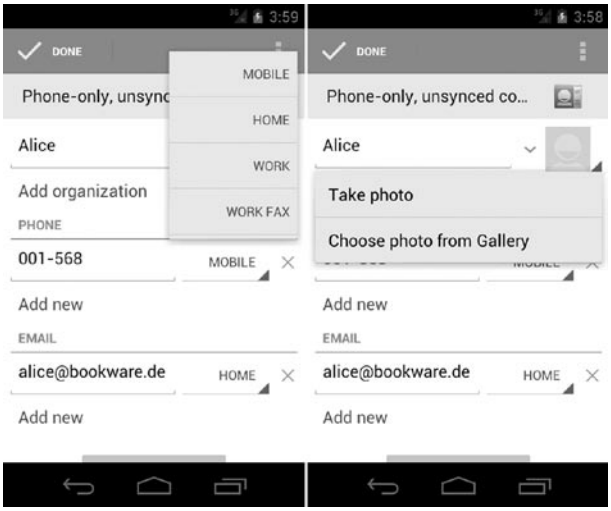


Abbildung 5: Popup-Menü

Praxisbeispiel

Zur Implementierung der Contextual Action Bar unter Android-Versionen ab Honeycomb finden sich im Internet einige Beispiele und Anleitungen (siehe beispielsweise [2] oder [3]), weshalb sich das Praxisbeispiel mit der Unterstützung der Contextual Action Bar auf Geräten, die lediglich über Android-Versionen vor Honeycomb verfügen, widmet. Dazu soll für die *Coffeenotes*-Applikation eine Listenansicht der gespeicherten Bildnotizen implementiert werden, die Mehrfachselektion unterstützt und über Checkboxen auf das Vorhandensein des Selektionsmodus hindeutet.

Android stellt keine Kompatibilitätsbibliothek für die Contextual Action Bar bereit. Das im letzten Artikel kurz vorgestellte Projekt *ActionBarSherlock* bietet hingegen eine Portierung an. Voraussetzung ist die Einbindung des Quelltextes als Bibliothek und die Verwendung der damit verfügbaren *Activity*-Basisklassen. Da die Bildnotizen als Liste angezeigt werden sollen, wird die *ImageNotesActivity* von der *SherlockListActivity* abgeleitet, die wiederum von der herkömmlichen *ListActivity* abgeleitet ist, zusätzlich aber noch die Unterstützung der *ActionBar*-Funktionalitäten für Android-Versionen ab der Version 2.0 (*Eclair*) bereitstellt.

Mit Honeycomb wurden für *ListViews* ein neuer Auswahlmodus (*AbsListView.CHOICE_MODE_MULTIPLE_MODAL*) und ein neuer Listener (*AbsListView.MultiChoiceModeListener*) eingeführt. *ListViews*, bei denen jener Auswahlmodus aktiviert wurde, verwalten automatisch die Selektionszustände ihrer Elemente und stellen über den Listener *Callbacks* für die Steuerung der Contextual Action Bar bereit. Leider steht für diese zwei Vertreter des Android-Frameworks eine Portierung in die *ActionBarSherlock*-Bibliothek zum jetzigen Zeitpunkt noch aus, weshalb deren Verhalten manuell implementiert werden muss.

Aufgrund des Listenverhaltens von Android gestaltet sich dieses Vorhaben allerdings etwas aufwendiger, als man im ersten Moment vermuten könnte. Wenn man von den Standardlayouts für Listenelemente, die auf bestimmte Kombinationen aus wenigen Textfeldern und einem Bild beziehungsweise einer Checkbox beschränkt sind, abweichen möchte, benötigt man ein eigenes Layout, das mittels eines Adapters auf jedes einzelne Listenelement angewandt wird. Enthalten diese Layouts Checkboxen kann die Verwaltung der Selektionszustände von der übergeordneten *ListView* übernommen werden. Voraussetzung ist, dass die Layouts das Interface *Checkable* implementieren.

Hierbei behandelt die *ListView* aber das komplette Layout als Einheit und deaktiviert die Fokussierungsmöglichkeit nachfolgender *Views*. Was dazu führt, dass Klickevents nicht auf einzelnen Komponenten des Layouts wahrgenommen werden, sondern immer auf dem gesamten Layout. Gleiches gilt für die Fokussierung. Für den vorgesehenen Anwendungsfall ist das nicht geeignet, da ein einfacher Klick auf ein Listenelement dessen Detailansicht aufrufen soll, während ein langer Klick auf das Listenelement oder ein einfacher Klick auf die Checkbox den Selektionsmodus starten soll. Dieses Fokussierungsverhalten lässt sich aufheben (*setItemsCanFocus(true)*), was allerdings zur Folge hat, dass die *ListView* nicht mehr in der Lage ist, die Selektionszustände automatisch zu verwalten. Auch die speziellen *Listener* für die Listenelemente (*onListItemClickListener*, *onListItemLongClickLister*) lösen keine Events mehr aus. Das ist im Grunde kein großes Problem, hat allerdings zur Folge, dass diese Funktionalitäten und das Fokussierungsverhalten zur Navigation durch das Layout mittels eines *Trackballs* oder der Tastatur im Folgenden selbst implementiert werden müssen.

Die *Activity* verfügt über einen speziellen Adapter und über eine Referenz auf ein *ActionMode*-Objekt, welches den Selektionsmodus steuert. Im Konstruktor wird zuerst die herkömmliche Action Bar konfiguriert. Anschließend wird die oben angesprochene Möglichkeit zur Fokussierung von Nachfahren-*Views* der Listenelemente aktiviert, bevor der *ListView* der spezielle Adapter zugewiesen wird.

```
public class ImageNotesActivity extends
    SherlockListActivity {
  private ImageNotesAdapter mImageNotesAdapter;
  private ActionMode mActionMode;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getSupportActionBar().setHomeButtonEnabled(true);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    getListView().setItemsCanFocus(true);
    mImageNotesAdapter =
      new ImageNotesAdapter(
        this, IMAGE_NOTES,
        mCheckBoxListener, mOnClickListener,
        mOnLongClickListener,
        mOnSelectionChangedListener
      );
    setListAdapter(mImageNotesAdapter);
  }
}
```

Bevor dessen Details erklärt werden, geht es zuerst noch an die Definition des Layouts für die Listenelemente. Es handelt sich hierbei um eine eigens definierte *CheckedImageNoteView*, die von einem *RelativeLayout* abgeleitet ist. Einige Standardattribute (*layout_width, layout_height, padding, …*) wurden aus Gründen der Lesbarkeit entfernt. Jedes Listenelement verfügt also über eine Checkbox und ein kleines Vorschaubild. Außerdem werden der Titel, das Datum und ein Teil des Textes der jeweiligen Notiz angezeigt. Die einzige Besonderheit in diesem Layout ist die Aktivierung der Fokussierung beim Wurzelelement (*android:focusable="true"*). Damit ist es dem Anwender möglich, sowohl das gesamte Listenelement als auch die Checkbox zu fokussieren, bei der die Fokussierung standardmäßig aktiv ist. Die Attribute *nextFocusLeft* und *nextFocusRight* definieren wohin der Fokus übergehen soll, wenn der Benutzer die entsprechenden Tasten auf der Tastatur drückt oder den *Trackball* entsprechend betätigt. Bei der vertikalen Fokussteuerung wird über das Setzen des Werts *beforeDescendants* im *descendantFocusability*-Attribut das gesamte Listenelement der Checkbox vorgezogen.

```
<?xml version="1.0" encoding="utf-8"?>
<de.bookware.coffeenotes.CheckedImageNoteView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/checked_image_note"
  android:focusable="true"
  android:descendantFocusability="beforeDescendants"
  android:nextFocusLeft="@+id/checkBox"
  android:nextFocusRight="@id/checkBox">

  <CheckBox android:id="@id/checkBox"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"
    android:nextFocusLeft="@id/checked_image_note"
    android:nextFocusRight="@id/checked_image_note" />

  <ImageView android:id="@+id/thumbnail"
    android:layout_centerVertical="true"
    android:layout_toRightOf="@id/checkBox" />

  <TextView android:id="@+id/title"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@id/thumbnail" />

  <TextView android:id="@+id/date"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true" />

  <TextView android:id="@+id/teaser"
    android:layout_below="@id/date"
    android:layout_toRightOf="@id/thumbnail" />
</de.bookware.coffeenotes.CheckedImageNoteView>
```

Als nächstes geht es an die Implementierung dieses Layouts. Es verfügt über Referenzen zu den gerade via XML definierten *Views* und die Standardkonstruktoren (im Listing entfernt). Außerdem wird ein Index gespeichert, der bei einem *ArrayAdapter* der Position im *Array* oder bei einem *CursorAdapter* dem Zeilenindex entspricht. Daneben existieren eine Methode zum Zuweisen der eigentlichen Bildnotizinformationen und eine Methode zum Zuweisen der erforderlichen *Listener*, die weiter unten genauer beschrieben werden. Darauf folgen drei selbsterklärende Methoden, die die Checkbox verwalten. Da dem Listenelement abhängig vom Zustand der Checkbox eine andere Hintergrundfarbe zugewiesen wird, muss diese auch nach einer Fokussierung wiederhergestellt werden. Darum kümmert sich die *onFocusChange*-Methode.

```
public class CheckedImageNoteView
  extends RelativeLayout
  implements OnFocusChangeListener
{
  private long mIndex;
  private CheckBox mCheckBox;
  private ImageView mThumbnail;
  private TextView mTitle;
  private TextView mDate;
  private TextView mTeaser;
  @Override
  protected void onFinishInflate() {
    mCheckBox =
      (CheckBox) findViewById(R.id.checkBox);
```

```
        mThumbnail =
          (ImageView) findViewById(R.id.thumbnail);
        mTitle = (TextView) findViewById(R.id.title);
        mDate = (TextView) findViewById(R.id.date);
        mTeaser = (TextView) findViewById(R.id.teaser);
        setOnFocusChangeListener(this);
    }

    public long getIndex() {
        return mIndex;
    }

    public void populate(long index, ImageNote imageNote) {
        mIndex = index;
        mThumbnail.setImageResource(
          imageNote.getThumbResId()
        );
        mTitle.setText(imageNote.getTitle());
        mDate.setText(imageNote.getDate());
        mTeaser.setText(imageNote.getText());
    }

    public void initializeListeners(
        OnClickListener checkBoxListener,
        OnClickListener onClickListener,
        OnLongClickListener onLongClickListener
    ) {
        mCheckBox.setOnClickListener(checkBoxListener);
        this.setOnClickListener(onClickListener);
        this.setOnLongClickListener(onLongClickListener);
    }

    public boolean isChecked() {
        return mCheckBox.isChecked();
    }

    public void setChecked(boolean checked) {
        mCheckBox.setChecked(checked);
        if (checked) {
            setBackgroundColor(0xFF877C6F);
        } else {
            setBackgroundColor(0x00FFFFFF);
        }
    }

    public void toggle() {
        setChecked(!isChecked());
    }

    @Override
    public void onFocusChange(View v, boolean hasFocus) {
        if (hasFocus) {
            setBackgroundColor(0xFF00FF00);
        } else {
            setChecked(isChecked());
        }
    }
}
```

Für die Zuweisung des Layouts an die einzelnen Listenelemente zeichnet sich der *ImageNotesAdapter* verantwortlich. Im vorliegenden Beispiel verwaltet dieser die Bildnotizen der Einfachheit halber in einem *Array* – normalerweise käme dafür wohl eher eine Datenbank in Betracht. Die Zuweisung der Daten erfolgt in der *getView*-Methode, die bei der Integration der Listenelemente in die *ListView* vom System automatisch aufgerufen wird. Für jede Bildnotiz wird dabei eine neue *CheckedImageNoteView* erzeugt. Bei *ArrayAdaptern* sollte man unbedingt von der *convertView* Gebrauch machen. Dabei handelt es sich um eine *View*, die beim Scrollen der Listen aus dem Anzeigebereich verschwindet. Diese kann dann am anderen Ende des Anzeigebereichs wieder eingefügt werden, was die ressourcenintensive Erzeugung der *View* aus der entsprechenden XML-Datei vermeidet und einen Geschwindigkeitsvorteil von etwa 150% (vgl. [4]) verspricht. Auch die Zuweisung der erforderlichen *Listener* ist nur bei der initialen Erzeugung der *View* nötig, da die *Listener* bei recycelten *Views* noch gesetzt sind. Am Ende der Methode werden die Daten der Bildnotiz in die *View* integriert. Der Selektionszustand der Listenelemente wird in einem *Set* verwaltet, welches die Indizes der selektierten *Views* enthält. Die Methoden zur Verwaltung dieser Zustände sind wiederum selbsterklärend. Immer wenn ein Listenelement seinen Selektionszustand verändert, wird der *OnSelectionChanged-Listener* über diesen Vorgang informiert.

```
public class ImageNotesAdapter extends
    ArrayAdapter<ImageNote> {

    private final ImageNote[] mImageNotes;
    private final LayoutInflater mLayoutInflater;
    private final Set<Long> mSelected;
    private final OnClickListener mCheckBoxListener;
    private final OnClickListener mOnClickListener;
    private final OnLongClickListener
      mOnLongClickListener;
    private final OnSelectionChangedListener
      mOnSelectionChangedListener;

    public ImageNotesAdapter(
        Context context, ImageNote[] imageNotes,
        OnClickListener checkBoxListener,
        OnClickListener onclickListener,
        OnLongClickListener onLongClickListener,
        OnSelectionChangedListener onSelectionChangedListener
    ) {
        // Assigns the submitted values...
    }

    @Override
    public View getView(
        int position, View convertView, ViewGroup parent
    ) {
```

```
        CheckedImageNoteView view;
        if (convertView != null) {
            view = (CheckedImageNoteView) convertView;
            view.setChecked(isSelected(position));
        } else {
            view =
                (CheckedImageNoteView) mLayoutInflater.inflate(
                    R.layout.checked_image_note, parent, false
                );
            view.initializeListeners(
                mCheckBoxListener, mOnClickListener,
                mOnLongClickListener
            );
        }
        view.populate(position, mImageNotes[position]);
        return view;
    }

    public ImageNote getImageNote(
        CheckedImageNoteView view
    ) {
        return mImageNotes[view.getIndex()];
    }

    public void changeSelectionState(
        CheckedImageNoteView view
    ) {
        boolean selected = !isSelected(view);
        view.setChecked(selected);
        if (selected) {
            mSelected.add(Long.valueOf(view.getIndex()));
        } else {
            mSelected.remove(Long.valueOf(view.getIndex()));
        }
        mOnSelectionChangedListener.onSelectionChanged(
            view, selected, mSelected.size()
        );
    }

    public boolean isSelected(CheckedImageNoteView view) {
        return mSelected.contains(
            Long.valueOf(view.getIndex())
        );
    }

    private boolean isSelected(long index) {
        return mSelected.contains(Long.valueOf(index));
    }

    public int getSelectedCount() {
        return mSelected.size();
    }

    public void resetSelection() {
        mSelected.clear();
        notifyDataSetChanged();
    }

    public interface OnSelectionChangedListener {
        void onSelectionChanged(
            CheckedImageNoteView view, boolean selected,
            int selectedCount
        );
    }
}
```

Damit sind die nötigen Arbeiten zur Darstellung der Liste abgeschlossen. Es folgt die Implementierung der Benutzerinteraktion. Dazu werden die benötigten *Listener* in der *ImageNotesActivity* erzeugt. Für das Aufrufen der Detailansicht einer Bildnotiz wird ein einfacher *OnClickListener* erzeugt, der beim Klicken auf ein Listenelement aktiv wird. Wenn der Selektionsmodus aktiv ist, soll anstelle des Aufrufs der Detailansicht der Selektionszustand des Elements geändert werden. Zum Starten des Selektionsmodus existieren zwei weitere *Listener*, die bei einem langen Klick auf ein Listenelement oder bei der Aktivierung einer Checkbox zum Einsatz kommen. Diese versuchen zuerst den Contextual Action Mode zu starten und informieren dann den Adapter über die Änderung ihres Selektionszustands.

```
private final OnClickListener mOnClickListener =
    new OnClickListener() {
    @Override
    public void onClick(View view) {
        if (mActionMode == null) {
            Intent intent =
                new Intent(this, ImageNoteActivity.class);
            intent.putExtra(
                ImageNoteActivity.IMAGE_NOTE_ID,
                ((CheckedImageNoteView) view).getIndex()
            );
            startActivity(intent);
        } else {
            mImageNotesAdapter.changeSelectionState(
                (CheckedImageNoteView) view
            );
        }
    }
};


private final OnClickListener mCheckBoxListener =
    new OnClickListener() {
    @Override
    public void onClick(View view) {
        tryInvokeCab();
        mImageNotesAdapter.changeSelectionState(
            (CheckedImageNoteView) view.getParent()
        );
    }
};


private final OnLongClickListener mOnLongClickListener =
    new OnLongClickListener() {
    @Override
    public boolean onLongClick(View view) {
        tryInvokeCab();
        mImageNotesAdapter.changeSelectionState(
            (CheckedImageNoteView) view
        );
        return true;
    }
};
```

Falls gerade kein Selektionsmodus aktiv ist, kann dieser über die *startActionMode*-Methode, die von der *Activity* bereitgestellt wird, gestartet werden. Die Methode erwartet ein *Callback*-Objekt, dass verschiedene *Hooks* in den Lebenszyklus der Contextual Action Mode zur Verfügung stellt. Beim Erzeugen des Modus muss die Contextual Action Bar erzeugt und mit *Action Items* gefüllt werden. Dies erfolgt wie gewohnt mit Hilfe von in XML erzeugten Menüdateien. Die *onPrepareActionMode*-Methode ermöglicht das Aktivieren/Deaktivieren von bestimmten Menüelementen. Dies ist dann nötig, wenn sich die Anzahl der ausgewählten Bildnotizen ändert und einige Aktionen nicht mehr auf die aktuelle Selektion anwendbar sind. Wenn der Benutzer auf ein *Action Item* klickt, kann in der *onActionItemClicked*-Methode die erforderliche Aktion angestoßen werden. Bei der Beendigung des Selektionsmodus bietet die *onDestroyActionMode*-Methode den richtigen Ort für die entsprechenden Aufräumarbeiten. In diesem Fall wird das *ActionMode*-Objekt dereferenziert und der Selektionszustand aller Listenelemente zurückgesetzt.

```
private void tryInvokeCab() {
  if (mActionMode == null) {
    mActionMode = startActionMode(mActionModeCallback);
    mActionMode.setTitle(
      "1 " + getResources().getString(R.string.note_selection)
    );
  }
}
private ActionMode.Callback mActionModeCallback =
  new ActionMode.Callback() {
  @Override
  public boolean onCreateActionMode(
    ActionMode mode, Menu menu
  ) {
    mode.getMenuInflater().inflate(
      R.menu.image_notes_cab, menu
    );
    return true;
  }

  @Override
  public boolean onPrepareActionMode(
    ActionMode mode, Menu menu
  ) {
    if (mImageNotesAdapter.getSelectedCount() == 1) {
      menu.findItem(R.id.menu_share).setVisible(true);
      menu.findItem(R.id.menu_share).setEnabled(true);
    } else {
      menu.findItem(R.id.menu_share).setVisible(false);
      menu.findItem(R.id.menu_share).setEnabled(false);
    }
    return true;
  }

  @Override
  public boolean onActionItemClicked(
    ActionMode mode, MenuItem item
  ) {
    switch (item.getItemId()) {
    // Executes the selected action...
  }

  @Override
  public void onDestroyActionMode(ActionMode mode) {
    closeCab();
  }
};

private void closeCab() {
  mImageNotesAdapter.resetSelection();
  mActionMode = null;
}
```

Am Ende bleibt noch die Definition eines *OnSelectionChangedListener*, der über jede Auswahländerung informiert wird. Seine Aufgabe ist es an Hand der übergebenen Anzahl der ausgewählten Elemente den Titel der Contextual Action Bar auf den neuesten Stand zu bringen und gegebenenfalls eine Aktualisierung der *Action Items* vorzunehmen, indem er die *invalidate*-Methode auf dem *ActionMode*-Objekt aufruft. Dies führt dann zur Ausführung der weiter oben bereits beschriebenen *onPrepareActionMode*-Methode. Falls keine Listenelemente mehr ausgewählt sind, beendet er den Selektionsmodus.

```
private final OnSelectionChangedListener
mOnSelectionChangedListener =
  new OnSelectionChangedListener() {
  @Override
  public void onSelectionChanged(
    CheckedImageNoteView view,
    boolean selected, int selectedCount) {
    if (selectedCount < 1) {
      mActionMode.finish();
    } else if (selectedCount == 1) {
      mActionMode.setTitle(
        selectedCount + " "+ getResources().getString(
          R.string.note_selection
        )
      );
      mActionMode.invalidate();
    } else {
      mActionMode.setTitle(
        selectedCount + " "+ getResources().getString(
          R.string.notes_selection)
        );
      if (selected) {
        mActionMode.invalidate();
      }
    }
  }
};
```

Abbildung 5: Coffeenotes unter Android 2.2 und 4.0

Fazit

Die Contextual Action Bar führt ganz klar zu einer Steigerung des Benutzererlebnisses. Das Praxisbeispiel zeigt, dass es mit Hilfe der *ActionBarSherlock*-Bibliothek relativ einfach möglich ist dieses Benutzererlebnis auch auf ältere Geräte zu transportieren. Da diese immer noch den weitaus größten Teil, der sich im Umlauf befindlichen Android-Geräte ausmachen, führt – meiner Ansicht nach – in der nächsten Zeit wohl kein Weg an solchen Kompatibilitätsbibliotheken vorbei, wenn man mit den neuen Konzepten eine große Zahl von Anwendern erreichen möchte.

Referenzen

[1] USER EXPERIENCE AWARDS, 2012 *2012 Winners*
http://userexperienceawards.com/2012-winners

[2] DEVELOPERS *Samples*
http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/view/List16.html

[3] ANDROID LISTVIEW AND LISTACTIVITY *Tutorial*
http://www.vogella.com/articles/AndroidListView/article.html#listview_actionbar

[4] GOOGLE *The world of list view*
http://dl.google.com/googleio/2010/android-world-of-listview-android.pdf

Kurzbiographie

JOHANNNES KÖSTLER (johannes.koestler@mathema.de) studiert IT-Sicherheit an der Universität Passau. Neben seinem Studium beschäftigt er sich überwiegend mit der Android-Plattform.

COPYRIGHT © 2012 BOOKWARE 1865-682X/12/06/002 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

Der Herbstcampus möchte sich ein bisschen von den üblichen Konferenzen abheben und deshalb konkrete Hilfe für Software-Entwickler, Architekten und Projektleiter bieten.

Dazu sollen die in Frage kommenden Themen möglichst in verschiedenen Vorträgen besetzt werden: als Einführung, Erfahrungsbericht oder problemlösender Vortrag. Darüber hinaus können Tutorien die Einführung oder die Vertiefung in ein Thema ermöglichen.

Haben Sie ein passendes Thema oder Interesse, einen Vortrag zu halten? Dann fragen Sie einfach bei info@bookware.de nach den Beitragsinformationen oder lesen Sie diese unter www.herbstcampus.de nach.

3.– 6. September 2012
in Nürnberg

Nickeldioxid?

NIO.2 einmal vorgestellt

von Frank Goraus

Mit dem *JDK7* kamen mal wieder einige Neuerungen in Java hinzu. Eine davon ist eine neue API für die Arbeit mit Dateien und Dateisystemen, welche in diesem Artikel vorgestellt werden soll, und anhand kleiner Beispiele erläutert wird.

Was ist überhaupt NIO.2?

Auch wenn es sich genauso anhört, gemeint ist nicht die chemische Verbindung NiO_2 für Nickeldioxid. Die Abkürzung steht für „More New I/O APIs for the Java™ Plattform", kurz *NIO.2*. Hervorgegangen ist dies aus dem *JSR-203*, welcher noch aus dem Jahre 2003 stammt und einige Probleme der „alten" API beheben sollte, denn mit den unter *java.io* vorhandenen Klassen ließen sich einige Aufgaben nur sehr umständlich lösen. Wenn man z. B. Dateien kopieren wollte, musste man selbst die neue Datei erzeugen und den Inhalt Byte für Byte übertragen. Um Änderungen an Dateien mitzubekommen, musste man selbst zyklisch das Datum der letzten Änderung überwachen. Und an symbolische Links und virtuelle oder entfernte Dateisysteme brauchte man gar nicht erst zu denken.

All dies soll nun besser werden mit NIO.2 …

Für die Neugierigen: NiO_2 ist ein giftiges Oxydationsmittel, dass sich durch Erhitzung oder bei Kontakt mit Wasser zersetzt. Es selbst brennt nicht, erhöht aber die Brandgefahr erheblich und fördert bestehende Brände. Und hat zum Glück keine Implikationen für unsere Arbeit mit Dateien unter NIO.2.

Grundlegende Neuerungen

Die neue API ist kein Ersatz für die alte, welche nach wie vor im Package *java.io* zu finden ist. Und obwohl sie eigentlich „deprecated" wäre, ist sie noch nicht als solche markiert. Die neue IO-Implementierung befindet sich deswegen im Package *java.nio*. Die wichtigsten neuen Klassen *Path* und *FileSystem* findet man im Unterpackage *java.nio.file*, ebenso wie die wichtigen Helper-Klassen *Files*, *Paths* und *FileSystems*. Doch alles der Reihe nach.

NIO.2 bringt einen neuen Ansatz mit sich um mit den Dateien umzugehen. Denn die neue Klasse *Path* repräsentiert im Gegensatz zu *File* nur noch den Pfad zu einer Datei oder einem Verzeichnis. Somit stehen auch nur pfadbezogene Methoden zur Verfügung. Eine weitere Neuerung ist die abstrakte Klasse *FileSystem*, die nur grundlegend ein Dateisystem beschreibt. Darauf aufsetzend gibt es verschiedene konkrete Implementierungen z. B. für ein lokales Dateisystem oder ein Zip-Archiv. Über die statische Methode *FileSystems.getDefault()* kommt man außerdem jederzeit an das aktuelle Dateisystem heran.

Pfadfinder

Path selbst ist nur ein Interface und kann nicht direkt instanziiert werden. Es gibt jedoch zig Möglichkeiten um an ein konkretes Objekt heranzukommen.

Zum einen der Weg über ein *FileSystem*. In diesem Falle über das per Default verfügbare lokale Dateisystem.

```
FileSystem fs = FileSystems.getDefault();
Path p = fs.getPath("D:/demo/nio2.txt");
```

Kürzer geht es über die Helper-Klasse *Paths*, funktional bleibt es beim selben.

```
Path p = Paths.get("D:/demo/nio2.txt");
```

Wobei man hier auch den Pfad in einzelnen Teilen angeben kann und die Trennzeichen (Separator) automatisch systemspezifisch ergänzt werden.

```
Path p = Paths.get("D:", "demo", "nio2.txt");
```

Einen ähnlichen Effekt kann man auch über die *resolve()*-Methode erreichen. Das Ergebnis bleibt das gleiche.

```
Path p = Paths.get("D:").resolve("demo").resolve("nio2.txt");
```

Es wurde jedoch auch die alte *File*-Klasse erweitert, sodass diese nun auch eine *toPath()*-Methode besitzt.

```
File f = new File("D:/demo/nio2.txt");
Path p = f.toPath();
```

Path kann jedoch nicht nur den Pfad aufnehmen, sondern stellt eine hierarchische Liste der einzelnen Pfadteile dar, welche man auch abfragen kann.

```
Path p = Paths.get("D:/demo/nio2.txt");
Path demo = p.getName(1);
System.out.println(demo); // "demo"
Path nio2txt = p.subpath(0, 2);
System.out.println(nio2txt); // "demo\nio2.txt"
```

Path implementiert zudem auch noch *Comparable*, weswegen man auch Pfade vergleichen kann. Da *equals()* jedoch nur die Namen berücksichtigt und die Pfade nicht auflöst, gibt es auch die Methode *File.isSameFile()*, die genau dieses Manko beseitigt.

```
p.equals(nio2txt);
Files.isSameFile(p, nio2txt);
```

Ok, jetzt wissen wir wie wir einen Pfad zu einer Datei bekommen, doch was hilft uns das wenn wir mit der darunterliegenden Datei etwas machen möchten?

X Files

Hierfür gibt es die Klasse *Files*, welche Methoden anbietet, für alles was man sich mit Dateien vorstellen kann. Dazu muss man nur die gewünschte Methode statisch aufrufen und einen *Path* (und eventuell weitere Parameter) mitliefern.

Hier einmal die gängigen Aufrufe um die Größe einer Datei abzufragen, sie zu kopieren, verschieben und zu löschen.

```
Path from = Paths.get("D:/demo/nio2.txt");
Path to = Paths.get("D:/demo/nio2-copy.txt");
long size = Files.size(from);
Files.copy(from, to);
Files.move(
    from, to, StandardCopyOption.REPLACE_EXISTING
);
Files.delete(to);
```

Der Vorteil dieser Methoden ist vor allem, dass hier im Hintergrund die Betriebssystemoperationen verwendet werden, welche wesentlich schneller und ressourcenschonender sein sollten als die frühere Byte-Kopiererei.

Besondere Erleichterungen sind aber auch die Methoden um eine Datei komplett als Strings oder Bytes einzulesen.

```
Path p = Paths.get("D:/demo/nio2.txt");
List<String> allLines =
    Files.readAllLines(p, StandardCharsets.UTF_8);
byte[] allBytes = Files.readAllBytes(p);
```

Doch auch das Schreiben einer Datei ist ganz einfach als Einzeiler möglich.

```
Files.write(p, allBytes);
```

Somit sind alle wichtigen Dateioperationen vorhanden, einfach zu benutzen und auch zentral an einer Stelle zusammengefasst.

400, 644 & 755

Auch im Bezug auf die Dateiattribute hat sich einiges getan. Früher hat man diese über die *File*-Klasse abgefragt, hatte jedoch keinen Zugriff auf betriebssystemspezifische Attribute. Dies ist nun einerseits etwas aufgeräumter und andererseits möglich. NIO.2 definiert hier (im Package *java.nio.file.attribute*) mehrere Klassen *xxxFileAttributes* und *xxxFileAttributeView*. Beide bilden jeweils ein Paar. Während *xxxFileAttributeView* die entsprechenden Methoden zum Auslesen der Attribute deklariert, definiert *xxxFileAttributes* den Zugriff auf die Properties, sprich welche es gibt und welchen Typ sie haben. Als gemeinsamer Nenner gelten hier die Klassen mit dem Prefix *Basic*, von denen es Ableitungen mit dem Prefix *Dos* für DOS- und *Posix* für Linux-Systeme gibt.

Möchte man nun die Attribute für eine konkrete Datei einlesen, so braucht man beim Aufruf der *readAttributes()*-Methode der *Files*-Helperklasse nur die entsprechende *xxxFileAttributes*-Klasse mitzugeben.

```
BasicFileAttributes attributes =
    Files.readAttributes(p, BasicFileAttributes.class);
```

Nun kann man verschiedene Attribute abfragen, wie Erstellungszeit, letzter Zugriff, ob es sich um einen Ordner oder einen symbolischen Link handelt.

```
FileTime creationTime = attributes.creationTime();
FileTime lastAccessTime = attributes.lastAccessTime();
boolean isDirectory = attributes.isDirectory();
boolean isSymbolicLink = attributes.isSymbolicLink();
```

Aufpassen muss man hier mit der Darstellung des Datums, da es sich nicht um den üblichen Typ *java.util.Date* sondern *FileTime* handelt. Hier muss man den Umweg über den Timestamp nehmen.

```
Date creationDate = new Date(creationTime.toMillis());
FileTime lastAccessTime =
  FileTime.fromMillis(new Date().getTime());
```

Wenn man nun weiß, dass man sich auf einem DOS-basierten System befindet, kann man auch dessen spezifische Attribute aufrufen.

```
DosFileAttributes attributes =
  Files.readAttributes(p, DosFileAttributes.class);
  attributes.isArchive();
  attributes.isSystem();
```

Die Attribute von *BasicFileAttributes* stehen dabei ebenfalls zur Verfügung, da *DosFileAttributes* diese implementiert.

Change is good

Bisher haben wir nur Attribute gelesen, doch was wenn man diese verändern möchte? Hierzu führt der Weg über die *xxxFileAttributeView*. Diese bietet jedoch nur eine Methode *setTimes()* an, mit der man alle Zeiten in einem Rutsch ändern kann.

```
BasicFileAttributeView fileAttributeView =
    Files.getFileAttributeView(
      p, BasicFileAttributeView.class
    );
fileAttributeView.setTimes(null, lastAccessTime, null);
```

Interessant ist hierbei auch, dass man von der *xxxFileAttributeView* auch die korrespondierenden *xxxFileAttributes* bekommen kann.

```
BasicFileAttributes fileAttributes =
  fileAttributeView.readAttributes();
```

Umgedreht geht dies jedoch nicht.

Einen Einblick in die Mächtigkeit der neuen API erhält man dann, wenn es darum geht eigene Attribute verarbeiten zu können. Denn wenn man sogar ein eigenes Dateisystem definieren kann, dann kann dieses auch weitere Attribute definieren auf die man zugreifen möchte. Möglich wird dies indem man entsprechend seine eigene *xxxFileAttributeView* und *xxxFileAttributes* definiert.

An dieser Stelle möchte ich dies jedoch nur erwähnen und nicht weiter darauf eingehen, da dies genug Stoff für einen weiteren Artikel ist.

Der Anfang und das Ende

Die Klasse *Files* liefert auch eine ganze Litanei an Methoden zum Anlegen verschiedenster Dinge. Zum Anlegen einer Datei oder eines Verzeischnisses gibt es die entsprechenden Methoden *createFile()* und *createDirectory()*. Angeben muss man dabei mindestens den Pfad für dieses und optional kann man eine Liste von *FileAttribute*-Objekten übergeben, die dann gleich vergeben werden. Es gibt jedoch bisher nur eine *Utility*-Klasse, welche diese erzeugen kann: *PosixFilePermisssions*. Somit geht dieser einfache Weg auch nur für Linux-Dateisysteme.

```
Path p = Paths.get("D:/demo/nio2.txt");
Files.createFile(p);
Path directory = Paths.get("/usr/home/frank/nio");
Set<PosixFilePermission> perms =
  PosixFilePermissions.fromString("rw-r--r--");
Files.createDirectory(
  directory, PosixFilePermissions.asFileAttribute(perms)
);
```

Beide Methoden gibt es auch noch in einer temporären Variante, das heißt die erzeugte Datei wird automatisch gelöscht, sofern sie geschlossen wurde oder die Anwendung beendet wird.

```
Path p = Paths.get("D:/demo");
Path directory = Paths.get("/usr/home/frank/nio");
Files.createTempFile(p, "temp-", "");
Files.createTempDirectory(
  directory, "temp-",
    PosixFilePermissions.asFileAttribute(perms)
);
```

Sollte man beim Erstellen eines Ordners einen kompletten Pfad anlegen wollen, oder zumindest außer dem letzten Knoten noch weitere Teile haben, welche erst angelegt werden müssen, so wird bei *createDirectory()* eine *IOException* auftreten. Doch auch hieran wurde gedacht, man muss stattdessen einfach nur *createDirectories()* benutzen.

```
Path p = Paths.get("D:/demo/gibt/es/noch/nicht");
Files.createDirectories(p);
```

Und wie eingangs erwähnt, werden nun auch symbolische Links unterstützt. Allerdings funktionieren diese auch nur wenn das Dateisystem dies unterstützt. Um einen Link zu verfolgen, kann man die Methode *readSymbolicLink()* verwenden.

```
Path link = Paths.get("D:/demo/wo-bin-ich");
Path target = Paths.get("D:/demo/hier/bin/ich");
Files.createSymbolicLink(link, target);
Path p = Files.readSymbolicLink(link);
```

Entsprechend einfach ist es auch bestehende Dateien, Verzeischnisse oder Links zu löschen.

```
Path p = Paths.get("D:/demo/nio2.txt");
Files.delete(p);
Files.deleteIfExists(p);
```

Die zweite Variante löscht, wie der Name vermuten lässt, nur wenn der Pfad auch existiert. Bei ersterer Variante fliegt hier noch eine IOException. Bei symbolischen Links wird entsprechend auch nur der Link gelöscht, nicht das Ziel. Verzeichnisse können nur gelöscht werden wenn sie leer sind. Einerseits ganz gut, weil man so nicht versehentlich einen ganzen Verzeichnissbaum löschen kann, aber anderseits muss man ein rekursives Löschen dann selbst implementieren, sofern man es braucht.

Pantomimen-Typen

Auch die Erkennung von *MIME*-Types hat sich verbessert. Hierfür bietet die *Files*-Klasse eine Hilfsmethode *probeContentType()* an.

```
Path p = Paths.get("D:/demo/nio2.txt");
String mimeType = Files.probeContentType(p);
System.out.println(mimeType); // text/plain
```

Intern funktioniert dies so, dass es eine Liste von registrierten *FileTypeDetector*-Implementierungen gibt, welche der Reihe nach gefragt werden. Sollte keiner einen Treffer liefern, so wird eine Standard-Implementierung gefragt. Eigene *FileTypeDetector*-Implementierungen kann man über die Datei *META-INF/services* in seinem *Jar-File* bekannt machen.

Baumwanderung

Will man durch den kompletten Inhalt eines Verzeichnisses wandern, so bietet NIO.2 auch hier eine schicke neue Version. Über *File.list()* ging dies zwar auch früher schon, doch lieferte dies immer ein Array mit allen Einträgen zurück, welche demzufolge initial erst einmal alle eingelesen werden mussten. In der neuen Version ist dies nun als *Stream* gelöst und liest nur noch Stück für Stück, dafür muss man jedoch daran denken den *Stream* am Ende auch wieder zu schließen.

```
Path p = Paths.get("D:/demo");
try (
   DirectoryStream<Path> files =
      Files.newDirectoryStream(p)
) {
   for (Path path : files) {
      System.out.println(path);
   }
}
```

Besonders schick ist es, dass man an dieser Stelle auch mittels *Regular Expressions*, einer ähnlichen so genannten „Globbering-Syntax“ oder einer Implementierung von *DirectoryStream.Filter* das *File-Listing* noch filtern kann. Der Aufruf

```
Files.newDirectoryStream(p, "*.txt");
```

liefert somit z. B. alle Dateien mit der Endung *.txt*. Zu beachten sei an dieser Stelle, dass der *DirectoryStream* nicht rekursiv arbeitet, sondern immer nur den Inhalt des angegebenen Verzeichnisses betrachtet. Um dieses Manko zu umgehen, kann man hier entweder selbst rekursiv vorgehen oder stattdessen den *FileVisitor* verwenden. Die Hilfsklasse *Files* bietet hierfür eine Methode *walkFileTree()* an, die als Parameter wie immer einen Pfad und zusätzlich eine Instanz von *FileVisitor* erwartet. *FileVisitor* ist selbst nur ein Interface, welches die Methoden *postVisitDirectory()* und *preVisitDirectory()*, für Dinge die vor oder nach dem Durchlaufen eines Verzeichnisses passieren sollen, sowie *visitFile()* und *visitFileFailed()*, in denen man spezifiziert was mit einer Datei passieren soll bzw. was passiert wenn dies fehlschlägt, beinhaltet. Für das Standardverhalten kann man hier auf einen bereits definierten *SimpleFileVisitor* zurückgreifen.

```
Path p = Paths.get("D:/demo");
Files.walkFileTree(p, new SimpleFileVisitor<Path>()
  {
  @Override
  public FileVisitResult visitFile(
     Path path, BasicFileAttributes attribs
  ) {
     try {
     System.out.println(
        "Datei "+path+" ist "+Files.size(path)+" Bytes groß."
     );
     } catch (IOException e) {}
     return FileVisitResult.CONTINUE;
  }
});
```

Mit diesem Aufruf kann man sich z. B. die Größe aller Dateien in einem Verzeichnis (und Unterverzeichnissen) ausgeben lassen.

Wachhund

Als letzte Neuerung im Rahmen dieses Artikels wollen wir uns noch mit dem Überwachen von Veränderungen in Verzeichnissen beschäftigen. Das Schlüsselelement ist hierfür die Klasse *WatchService*, welche wir uns spezifisch für ein Dateisystem erzeugen und dann über die Methode *register()* auf einem Pfad aktivieren können.

```
FileSystem fs = FileSystems.getDefault();
WatchService watchDog = fs.newWatchService();
Path p = Paths.get("D:/demo");
p.register(
   watchDog, StandardWatchEventKinds.ENTRY_MODIFY
);
```

Damit geben wir an, dass der *WatchService* für jegliche Dateiänderungen aktiv werden soll. Möglich wäre ebenso das Erstellen und Löschen von Dateien, die man auch per *Varargs* kombinieren kann.

Doch bisher können wir die Änderungen noch nicht auswerten, dazu muss man in einer (Endlos)Schleife die einzelnen Events auslesen.

```
while (true) {
  WatchKey key = watchDog.take();
  for (WatchEvent<?> event : key.pollEvents()) {
    System.out.println(
      "Datei "+event.context()+" wurde geändert."
    );
  }
  key.reset();
}
```

Mit *take()* wird der *WatchService* für weitere Zugriffe blockiert, solange bis man den erhaltenen *WatchKey* mit *reset()* wieder freigibt. Alternativ kann man auch mit *poll()* nicht blockierend einen *WatchKey* erhalten.

Fazit

Es ist erstaunlich was in der neuen API so alles drin steckt und berücksichtigt wurde. Praktisch alles was man von modernen Anwendungen her so kennt, lässt sich mit NIO.2 umsetzen. Und dabei haben wir die Möglichkeiten der neuen Dateisystem-API noch gar nicht betrachtet. Auch wenn wir praktisch 8 Jahre darauf warten mussten, gelohnt hat es sich allemal.

Weiterführende Literatur

- The Java Tutorials *File I/O (Featuring NIO.2)*
  http://docs.oracle.com/javase/tutorial/essential/io/fileio.html
- Galileo <Open book> *Java 7- Mehr als eine Insel*
  http://openbook.galileocomputing.de/java7/1507_05_003.html

Kurzbiografie

Frank Goraus (frank.goraus@mathema.de) ist Senior Developer bei der MATHEMA Software GmbH in Erlangen. Seit 2006 beschäftigt er sich bereits mit der Entwicklung von JEE-Anwendungen, u. a. in Verbindung mit einem Portal-Server. Seine Liebe zum Detail verwirklicht er mit seinen Web-Design-Kenntnissen. In seiner Freizeit beschäftigt er sich außerdem mit Android-Entwicklung, verschiedensten Web-Frameworks und einem eigenen Projekt für eine Sammlungsverwaltung.

Herbstcampus

Wissenstransfer par excellence

3.– 6. September 2012
in Nürnberg

Des Programmierers kleine Vergnügen

… und der ganze Rest

von Michael Wiedeking

Wie schon bei der ganzzahligen Division im letzten Vergnügen, wird natürlich auch deren Rest durch die Art der Rundung bestimmt. Könnte man zur Division durch eine Zweierpotenz einfach ein *Shift* benutzen, so könnte man den Rest genau so einfach über ein Bit-weises *Und* bestimmen. Aber das wäre ja zu einfach.

Hat man es nur mit positiven Zahlen zu tun, so ist alles ganz leicht. Möchte man die positive Zahl x durch die Zweierpotenz 2^k teilen, so genügt es – wie im letzten Vergnügen [1] gezeigt – die Zahl x um k Stellen nach rechts zu verschieben:

$$x/2^k = x \gg k$$

Teilt man analog im Dezimalsystem 123 durch $10 = 10^1$, erhält man 12, also erwartungsgemäß das, was man durch Verschieben um eine Stelle erhält. Bei dem, was da rausgeschoben wurde, handelt es sich um den Rest der ganzzahligen Division, denn $123 = 12 \cdot 10 + 3$.

Für die Zweierpotenzen heißt dies, dass nur eine geeignete Maske gefunden werden muss, bei der die niederwertigen k Bits gesetzt sind. Dann können durch ein Bit-weises *Und* die nicht benötigten höherwertigen Bits „gelöscht“ werden. Schon aus dem Kleinen Weihnachtsvergnügen von 2008 [2] wissen wir, dass sich dies bewerkstelligen lässt, indem man von 2^k einfach 1 abzieht. Also gilt

$$\mathrm{rem}(x, 2^k) = x \,\&\, (2^k - 1).$$

Leider gilt das ja für die negativen Zahlen nur dann, wenn gegen $-\infty$ gerundet wird, aber eben nicht beim Runden gegen Null, wie man das in den meisten Programmiersprachen wegen der modernen Prozessoren tun.

Geht man davon aus, dass das Ergebnis q der Division x/y des letzten Vergnügens optimal ist, lässt sich wegen $q = x/y$ der Rest r dadurch bestimmen, dass man das Produkt $q \cdot y$ berechnet und dieses von x abzieht:

$$r = x - q \cdot y$$

Da es sich bei y um die Zweierpotenz 2^k handelt, ist die Berechnung des Produkts mit einem *Shift* trivial. Zusammen mit der Subtraktion benötigt man also nur zwei Extrainstruktionen um den Rest zu berechnen:

$$p = q \ll k$$
$$r = x - p$$

Damit erhält man, unter Zuhilfenahme der Lösung vom letzten Vergnügen, die folgende Funktion zur Berechnung des Rests (*remainder*) der Division durch die Zweierpotenz 2^k:

```
int remOfPowerOf2(int x, int k) {
  int t = x >> (k - 1);
  t >>>= 32 - k;
  x += t;
  q = x >> k;
  int p = q << k;
  return q - p;
}
```

In der vorvorletzten und vorletzten Anweisung werden die Bits hinausgeschoben und durch das Einschieben Nullen nachgefüllt. Das Löschen der niederwertigen Bits lässt sich aber auch einfacher bewerkstelligen, indem man zunächst – wie oben gezeigt – eine Maske erstellt

und diese dann aber vor dem Bit-weisen *Und* mit dem original x zusätzlich noch negiert. Leider haben wir das original x verändert, aber das lässt sich leicht dadurch beheben, indem wir statt x += t, was für die Summe $x = x + t$ steht, $t = x + t$ schreiben. Damit haben wir letztlich nur die Variable umbenannt.

```
int remOfPowerOf2(int x, int k) {
  int t = x >> (k – 1);
  t >>= 32 – k;
  t = x + t;
  q = t & ~(2^k – 1);
  return x – t;
}
```

Um noch eine Instruktion einzusparen, können wir wegen der Beziehung $-z = \sim z + 1$ bzw. $\sim z = -z - 1$ im Zweierkomplement, das Bestimmen der Maske vereinfachen:

$$\sim(2^k - 1) = -(2^k - 1) - 1 = -2^k + 1 - 1 = -2^k$$

Alternativ hätte man zur Berechnung des Rests wieder wie beim letzten Mal vorgehen können. Die Berechnung für ein positives x ist t, wie oben gezeigt, x & M, wobei die Maske $M = 2^k - 1$ ist. Für ein negatives x müsste erst das Vorzeichen gewechselt werden, dann die Bits gelöscht und anschließend das Ergebnis wieder negiert werden:

$$\text{rem}(x, 2^k) = \begin{cases} x \,\&\, \text{M}, & x \geq 0 \\ -((-x) \,\&\, \text{M}), & x < 0 \end{cases}$$

In dieser Variante geht es also wieder einmal darum, in Abhängigkeit vom Vorzeichen die Negation vorzunehmen. Dazu bestimmt man zunächst $t = x \gg 32$ und erhält damit –1, wenn die Zahl negativ ist und ansonsten 0. Der Absolutbetrag *abs(x)* liefert ja immer einen positiven Wert, also werden mit *abs(x)* & M korrekt die niederwertigen Bits gelöscht. Auch jetzt wird wieder die Beziehung $-z = \sim z + 1$ genutzt um den Wert zu negieren.

Nun geht es also nur noch darum, t so einzusetzen, dass der Wert entweder negiert wird oder unverändert bleibt. Das Invertieren der Bits kann mit Hilfe eines Exklusiv-*Oders* bewerkstelligt werden, denn $\sim z = -1 \wedge z$. Mit $(z \wedge t) - t$ erhält man dann entweder die gewünschte Negation, oder der Wert z bleibt wegen der neutralen Null in $((z \wedge 0) - 0)$ unverändert.

```
int remOfPowerOf2(int x, int k) {
  int t = x >> 31;
  return ((abs(x) & (2^k – 1))^t) – t;
}
```

Steht der Absolutwert nicht als eigenständige Instruktion auf dem Prozessor zur Verfügung, so kann man dies – wie im Vergnügen [3] gezeigt – mit Hilfe von

```
int abs(int x) {
  int t = x >> 31;
  return x^t – t;
}
```

bewerkstelligen. Da t bereits berechnet wurde, würde das bei entsprechendem *Inlining* nur zwei zusätzliche Instruktionen kosten.

Bei der Bestimmung von Quotient und Rest einer Division durch eine Zweierpotenz kann sich der Einsatz dieser Optimierung sehr wohl lohnen. Ist in diesem Fall der Dividend positiv sind die Berechnungen trivial, lassen sie sich doch auf *Shift* bzw. *Und*-Verknüpfung reduzieren. Sind auch negative Dividenden zugelassen, wird die Sache zwar komplizierter, kann sich aber auf vielen Architekturen immer noch lohnen, insbesondere dann, wenn man sowohl Quotient als auch Rest benötigt und die Programmiersprache der Wahl keine entsprechende Funktion anbietet.

Aber wie immer muss man sehr sorgfältig prüfen, ob dies auch für den tatsächlichen Anwendungszweck taugt. Denn alles in allem ist es insgesamt immer noch ganz schön viel Arbeit für das kleine bisschen Rest.

Referenzen

[1] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Geteiltes Leid*, KaffeeKlatsch, Jahrgang 5, Nr. 12, Seiten 21 – 23, Bookware, Mai 2012 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2012-05.pdf

[2] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Potenzweierlei*, KaffeeKlatsch, Jahrgang 1, Nr. 12, Seite 3, Bookware, Dezember 2008 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2008-12.pdf

[2] Hahn, Jörn *Des Programmierers kleine Vergnügen – Absolut ohne Sprünge*, KaffeeKlatsch, Jahrgang 1, Nr. 5, Seite 24, Bookware, Mai 2008 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2008-05.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.

Copyright © 2012 Bookware 1865-682x/12/06/004 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.

Deutsch für Informatiker

Märchen, Niederländisch und anderes

von Alexandra Specht

Was hat Niederländisch mit Deutsch für Informatiker zu tun? Auf den ersten Blick nichts, außer, dass die Antwort auf die Frage „Wie heißt Vorrundenaus auf Niederländisch?“ (Na, wissen Sie es?) eine schöne deutsche Wortschöpfung ist, nämlich: „Heimrobben.“ Gut, das soll mein Beitrag zur EM 2012 sein.

Kommen wir zu einem anderen Thema.

> *„In den alten Zeiten, wo das Wünschen noch geholfen hat, lebte ein König, dessen Töchter waren alle schön, aber die jüngst war so schön, dass die Sonne selber, die doch so vieles gesehen hatte, sich verwunderte, so oft sie ihr ins Gesicht schien.“*

So beginnt eines der bekanntesten Märchen im deutschen Sprachraum, „Der Froschkönig oder der eiserne Heinrich“. Dieses und unzählige andere Märchen haben die Gebrüder Grimm zusammengetragen. Das dürfte allgemein bekannt sein. Was sicherlich weniger bekannt ist, ist, dass die Gebrüder Grimm, also Jacob und Wilhelm Grimm, auch viel auf dem Gebiet der Sprachwissenschaft gearbeitet haben und als Gründungsväter der Deutschen Philologie bzw. Germanistik gelten.

Um das Jahr 1821 begann Jacob Grimm mit seiner Arbeit an der „Deutschen Grammatik“. Der Titel ist irreführend, denn es handelt sich nicht um eine trocken-schematische Beschreibung des Aufbaus der zeitgenössischen Sprache. Jacob wollte vielmehr „ein historisches Leben mit allem Fluß freudiger Entwickelung in sie zaubern“. Das Werk bezieht sich jedoch auf alle germanischen Sprachen, ihre Zusammenhänge und ihre geschichtliche Entwicklung. Der erste Band beschäftigte sich zunächst mit Flexion, der zweite mit Wortbildung. Jacob stellte kein vollständiges Manuskript fertig, sondern ließ Druckbogen für Druckbogen drucken, sobald er die benötigte Menge Text geschrieben hatte. Den ersten Band überarbeitete er nochmals komplett, sodass dieser dann eher die Lautbildung zum Inhalt hatte.

In seiner Grammatik verfolgte Jacob als Erster die Entwicklung der heute „indoeuropäisch“ genannten Sprachen und die Gesetze des Lautwechsels bei Vokalen und Konsonanten. Damit legte er das Fundament für die moderne Etymologie, die Forschung zum Bedeutungswandel in verschiedenen (verwandten) Sprachen.

Die Leipziger Verleger Karl Reimer und Salomon Hirzel unterbreiten ihnen später den Vorschlag, ein „Deutsches Wörterbuch“, „den Grimm“, zu schreiben, das 1854 erschien. Die Gebrüder selbst arbeiteten das Wörterbuch bis zum Buchstaben D (Wilhelm) bzw. F (Jacob) aus. Sie konzipierten das Wörterbuch als Sammlung sämtlicher Wörter aus der Zeit „von Luther bis Goethe“. Hier stand weniger die Vereinheitlichung der Schreibweise im Mittelpunkt, wie später in Konrad Dudens „Vollständiges Orthographisches Wörterbuch“ (1880), sondern die Bedeutungsgeschichte des jeweiligen Wortes, dessen historische Verwendung anhand von zahlreichen Zitaten aus der Literatur, aber auch aus Fachsprache und Alltagsgebrauch nachvollzogen wird. Bei der Sammlung der Belege standen den Brüdern Grimm zahlreiche Helfer zur Seite, die zumeist zum Kreis ihrer Freunde und wissenschaftlichen Kollegen gehörten oder ihnen von Freunden und Kollegen vermittelt wurden. Die Sammlung der Belege wurde ebenso wie die Ausarbeitung des Wörterbuchs durch den Verlag bezahlt. Für das Großprojekt des Wörterbuchs mussten die Brüder Grimm eigene Pläne und laufende Arbeiten zurückstellen, was ein Grundproblem ihrer letzten beiden Lebensjahrzehnte werden sollte.

Und somit haben wir nicht nur diese wundervolle Märchensammlung den Gebrüdern Grimm zu verdanken, sondern auch den Grundstein der heutigen Germanistik.

Und wir, die wir uns täglich des Deutschen bedienen, um uns zu verständigen, um zu streiten, Koseworte zu sagen, unseren Kindern etwas beizubringen, um unserer Arbeit nachzugehen, sind sicherlich nicht mit Mark Twain einer Meinung, der da sagte:

> *Die deutsche Sprache sollte sanft und ehrfurchtsvoll zu den toten Sprachen abgelegt werden, denn nur die Toten haben die Zeit, diese Sprache zu lernen.*

Kaffeesatz

Naturkonstante

von Michael Wiedeking

Reisen ist ein Privileg. Und wenn selbst berufliche Reisen ein bisschen Zeit zum Umgucken lassen, so kann man sich nicht glücklicher schätzen. In diesem Zusammenhang begegnet einem natürlich auch immer wieder Software der unterschiedlichsten Art, mit der man aber in der Regel überraschend gut zurecht kommt.

Neulich durfte ich zu einer Konferenz nach Peking, in das Land der Mitte. China ist – nebenbei bemerkt – unbedingt eine Reise wert; und wenn ich in diesem Zusammenhang eine Empfehlung aussprechen sollte, dann die, dass man unbedingt auch aufs Land fahren sollte, damit man ein bisschen von der wundervollen Gegend hat.

Zu den Besonderheiten von China gehört ja auch die Schrift. So schön diese auch anzusehen ist, so wenig kann ich sie lesen, geschweige denn reproduzieren. Dankbarer Weise stehen in Peking bei allen wichtigen Hinweisen englische Übersetzungen, so dass man sich trotz alledem ganz gut zurechtfinden kann.

Unabhängig davon sprechen in der Großstadt viele Chinesen auch Englisch, und wenn nicht, so weiß man sich mit Zetteln oder Taschenrechnern zu helfen, um das Wichtigste zu erledigen: das Bezahlen. Dafür wird bei fast allen Automaten oder Web-Seiten eine englische Version angeboten, so dass dort wirklich fast keine Wünsche offen bleiben.

Um so mehr habe ich mich gefreut, als ich mich beim Kauf einer U-Bahn-Fahrkarte am Automaten davon überzeugen konnte, dass es überall auf der Welt vergleichbare Probleme gibt. Zwar verschleiert das blendende Licht auf dem Foto ganz ungeschickt den Hersteller des Virenprüfers, aber auch ohne diesen und ohne die Übersetzung konnte man doch das Wichtigste sofort verstehen.

Ach ist das schön, wenn man sich überall auf der Welt zurechtfinden kann.

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

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

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
Universitätsstr. 12, 45127 Essen
http://www.ruhrjug.de
Kontakt: Herr Heiko Sippel (heiko.sippel@ruhrjug.de)

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

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

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

JUG Karlsruhe
Java User Group Karlsruhe
Universität Karlsruhe, Gebäude 50.34
Am Fasanengarten 4, 76131 Karlsruhe
http://jug-karlsruhe.de
jug-karlsruhe@gmail.com

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

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

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

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

JUG Ostfalen
Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
Siekstraße 4, 38444 Wolfsburg
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
Schönaicherstraße 3, 70597 Stuttgart
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)

Schweiz

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

.Net User Groups

Deutschland

.NET User Group OWL
http://www.gedoplan.de/cms/gedoplan/ak/ms_net
c/o GEDOPLAN GmbH
Stieghorster Str. 60, 33605 Bielefeld

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

Die Dodnedder
.NET User Group Franken
http://www.dodnedder.de
Kontakt: Herr Bernd Hengelein
Herr Thomas Müller, Udo Neßhöver
(dodned@googlemail.com)

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

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

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

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

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

.NET 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
Sachsenstr. 24, 26121 Oldenburg
http://www.dotnet-oldenburg.de
Kontakt: Herr Hilmar Bunjes
(hilmar.bunjes@dotnet-oldenburg.de)
Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)

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

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

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

Österreich

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

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

Software Craftsmanship Communities

Deutschland

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

Persistenz mit Hibernate

OR-Mapping von Java-Objekten,
4. – 6. Juli 2012, 29. – 31. Oktober 2012,
1.315,– € (zzgl. 19 % MwSt.)

Objektorientierte Analyse und Design mit UML und Design Patterns

Methoden und Prinzipen für die Entwicklung von OO-Modellen und die Dokumentation mit der UML
23. – 25. Juli 2012, 15. – 17. Oktober 2012
1.180,– € (zzgl. 19 % MwSt.)

Neues in Java 7

Die nächste Java Generation,
20. – 21. August 2012, 5. – 6. November 2012,
835,– € (zzgl. 19 % MwSt.)

Refactoring

Verbesserung von Design und Quellcode bestehender Software
16. – 17. August 2012, 25. – 26. Oktober 2012
925,– € (zzgl. 19 % MwSt.)

Web-Anwendungen mit JavaServer Faces (JSF)

Komponentenbasierte Web-Entwicklung mit JavaServer Faces (JSF)
29. – 31. August 2012,
21. – 23. Nov. 2012
1.180,– €
(zzgl. 19 % MwSt.)

Lesen bildet. **Training macht fit.**

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

MATHEMA

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

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

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

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

Wir freuen uns auf Ihre Bewerbung!

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

Exklusive Sponsoren:

ORACLE · NOVATEC make IT happen! · GEBIT Solutions · exxcellent solutions · logica be brilliant together · COMPUWARE APM DYNATRACE | GOMEZ

JAVA FORUM stuttgart 2012

Die führende non-profit-Konferenz und Ausstellung rund um das Thema Java mit einem breiten Spektrum aktueller Themen der Software-Entwicklung

Donnerstag, 5. Juli 2012
Kultur- & Kongresszentrum Liederhalle Stuttgart

Eine Veranstaltung der
Java User Group Stuttgart e.V.
JUGS java user group stuttgart

Vorträge

42 Vorträge in 6 parallelen Tracks vermitteln ein breites Spektrum der aktuellen Java-Technologie: Technologievorträge, Projekterfahrungsberichte & Produktpräsentationen

Ausstellung

An den etwa 30 Ausstellungsständen namhafter Firmen können Sie sich während des ganzen Tages über deren Leistungen und Produkte informieren.

Info und Anmeldungen: www.java-forum-stuttgart.de

Anmeldung

ausschließlich via Internet:
www.java-forum-stuttgart.de

JAVA FORUM stuttgart 2012

	Workshop Java für Entscheider 4. Juli 2012	Java Forum Stuttgart 5. Juli 2012	Kombiticket 4. Juli & 5. Juli 2012
Frühbucherzeitraum: 19. April - 09.Mai 2012	350 €	129 €	469 €
Normalbucherzeitraum: 10. Mai - 10. Juni 2012	390 €	150 €	499 €
Spätbucherzeitraum: ab 11. Juni 2012	390 €	180 €	529 €

Beträge inkl. gesetzl. MwSt.. Eine Bestätigung und eine Rechnung über die Teilnahmegebühr erhalten Sie ca. eine Woche nach Anmeldung. Die Rechnung ist nach Erhalt zur Zahlung fällig.

Mittwoch 4. Juli 2012: Workshop Java für Entscheider

Wenn Sie aus der IT oder IT-nahen Einsatzfeldern kommen, aber keinen bis wenig Kontakt mit Java hatten, dann könnte diese eintägige Überblicksveranstaltung der JUGS Sie interessieren.

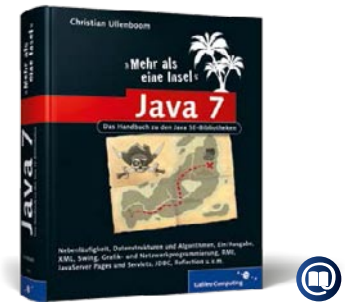
Freitag, 6. Juli 2012
9. Experten-Forum-Stuttgart

Einen Tag nach dem Java Forum Stuttgart findet das EFS 2012 statt. Organisiert wird es wieder von SENS und als neuem Partner vom Projekthaus Stuttgart (www.projekthaus-stuttgart.de).

Das Forum bietet am Vor- und am Nachmittag jeweils 5-6 Veranstaltungen (je 4 Seminarstunden) mit Referenten anerkannter Expertise aus Software-Entwicklung und IT-Projektmanagement an.

www.experten-forum-stuttgart.de

Galileo Computing: Java & Android



Christian Ullenboom

Java 7 – Mehr als eine Insel

Das Handbuch zu den Java SE Bibliotheken

Die Profi-Insel bietet umfassendes Praxiswissen zu den Bibliotheken und Technologien von Java in einem Band. Egal, ob es sich um Swing, XML, RMI und Webservices, JSP, Servlets, Applets, JDBC, Reflection und Annotationen oder Logging und Monitoring handelt, hier ist alles drin, was für die Arbeit an Java-Projekten wichtig ist. Ein Buch, das an den Arbeitsplatz jedes Industrieprogrammierers gehört!

» www.GalileoComputing.de/2253

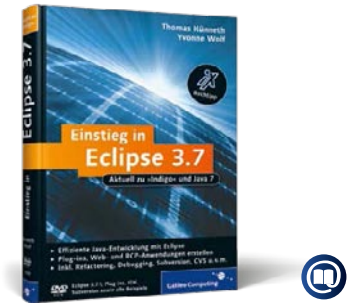
Christian Ullenboom

Java ist auch eine Insel

Das umfassende Handbuch

Das Java-Kultbuch in der 10. Auflage! Es bietet alles, was man zum Programmieren mit der Java Standard Edition wissen muss. Das Buch wurde gründlich überarbeitet, die Grundlagen werden jetzt noch ausführlicher besprochen. Die Insel ist erste Wahl, wenn es um aktuelles und praktisches Java-Wissen geht. Aktuell zu Java 7

» www.GalileoComputing.de/2672



Thomas Künneth, Yvonne Wolf

Einstieg in Eclipse 3.7

Aktuell zu Indigo und Java 7

Effiziente Java-Entwicklung mit Eclipse: Dieses Buch zeigt Ihnen, wie Sie die Möglichkeiten von Eclipse voll nutzen. Ob Grundlagen, Testen, Plugin- und RCP-Entwicklung, GUI- oder Web-Entwicklung - Sie lernen alles an anschaulichen Praxisbeispielen, so dass Ihnen die Umsetzung in Ihren Projekten mühelos gelingen wird!

» www.GalileoComputing.de/2463

Entdecken Sie unser komplettes Programm im Web!

» www.GalileoComputing.de

Thomas Künneth

Android 4

Apps entwickeln mit dem Android SDK

Sie möchten Apps für Android Tablets und Smartphones entwickeln? Java-Kenntnisse vorausgesetzt, wird Ihnen das durch die verständlichen Erklärungen und zahlreichen Praxisbeispiele schnell gelingen. Ob GUIs, Datenbanken, Kamera, Multimedia, Kontakte oder GPS - hier erfahren Sie alles, was Sie wissen müssen!

» www.GalileoComputing.de/3167

Uwe Post

Android-Apps entwickeln

Für Programmiereinsteiger geeignet

Android-Apps programmieren ohne Vorkenntnisse! Hier lernen Sie auf einfache und unterhaltsame Weise, wie Sie Apps für Android entwickeln. Schritt für Schritt programmieren Sie z. B. ein eigenes Spiel, das sich sehen lassen kann. Die benötigte Software finden Sie auf der DVD, so dass Sie sofort loslegen können!

» www.GalileoComputing.de/3158

Mike Bach

Apps entwickeln für Android 4

Das umfassende Training

Sie suchen einen anschaulichen Lernkurs zur App-Entwicklung mit Android 4? Lernen Sie in diesem Training, wie Sie eigene Android-Apps programmieren. Inkl. Java-Crashkurs und Video-Workshops zu GUI, GPS, Datenverwaltung, Spracheingabe, Multimedia und Sensorsteuerung.

» www.GalileoComputing.de/2955

» facebook.com/GalileoPressVerlag
» twitter.com/Galileo_Press
» gplus.to/GalileoPress

Galileo Computing
Wissen, wie's geht.

Das Allerletzte

Dies ist kein Scherz!
Diese Meldung 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 Juli.

Herbstcampus

Wissenstransfer par excellence

3.– 6. September 2012

in Nürnberg