HANSER

Dierk König, Andrew Glover, Paul King, Guillaume Laforge, Jon Skeet

Groovy im Einsatz

ISBN-10: 3-446-41238-7
ISBN-13: 978-3-446-41238-5

Leseprobe

Weitere Informationen oder Bestellungen unter
http://www.hanser.de/978-3-446-41238-5
sowie im Buchhandel

1 Ihr Weg zu Groovy

> *One main factor in the upward trend of animal life has been the power of wandering.*
>
> *Alfred North Whitehead*

Willkommen in der Groovy-Welt.

Sie kennen Groovy wahrscheinlich schon aus Blogs und Mailinglisten oder haben kurze Programmstücke davon gesehen. Vielleicht hat auch ein Kollege Sie auf ein Stück Ihres Codes aufmerksam gemacht und behauptet, dasselbe ließe sich mit Groovy in wenigen Zeilen erledigen. Womöglich haben Sie dieses Buch auch nur in die Hand genommen, weil der Titel interessant klingt. Auf jeden Fall werden Sie sich fragen, warum Sie Groovy lernen sollten und welchen Gewinn Sie erwarten können.

Groovy wird Ihnen rasch einige Vorteile einbringen, ob Sie damit einfacher Java-Code schreiben, wiederkehrende Aufgaben automatisieren oder in Ihrer alltäglichen Arbeit als Programmierer durch Ad-hoc-Scripting unterstützt werden. Auf längere Sicht bringt Ihnen Groovy Erfolg ein, weil es Ihren Code *lesbarer* macht. Doch was vielleicht am wichtigsten ist: Es macht einfach Spaß damit zu arbeiten.

Groovy zu lernen ist eine kluge Investition, denn Groovy verleiht der Java-Plattform die Mächtigkeit fortgeschrittener Sprachfunktionen, wie zum Beispiel Closures, dynamische Typisierung und das Metaobjektprotokoll. Ihr Java-Wissen wird nicht überflüssig, wenn Sie den Weg von Groovy einschlagen. Groovy baut auf Ihren bestehenden Kenntnissen und Ihrer Vertrautheit mit der Java-Plattform auf, sodass Sie selbst wählen können, wann Sie welche Sprache verwenden, und wann Sie beide nahtlos kombinieren möchten.

Falls Sie je bewundert haben, wie Ruby-Cracks eine vollständige Webanwendung an einem Nachmittag implementieren können, wie Python-Leute mit Collections jonglieren, wie Perl-Hacker eine Serverfarm mit ein paar Tastendrücken managen, oder wie Lisp-Gurus ihre gesamte Codebasis mit einer kleinen Änderung vom Kopf auf die Füße stellen, dann sollten Sie bedenken, welche *Sprachfähigkeiten* ihnen zur Verfügung stehen. Das Ziel von Groovy ist es, Sprachfunktionen mit vergleichbarer Wirkung für die Java-Plattform bereitzustellen,

aber zugleich das Java-Objektmodell zu befolgen und die Perspektive eines Java-Programmierers einzunehmen.

Dieses erste Kapitel liefert Hintergrundinformationen über Groovy und erklärt alles, was Sie für die ersten Schritte wissen müssen. Am Anfang steht die Geschichte von Groovy: warum es erschaffen wurde, welche Überlegungen seinen Entwurf bestimmten und wie es sich in die Landschaft von Sprachen und Technologien einfügt. Der folgende Abschnitt schildert die Vorzüge von Groovy und wie sie Ihnen das Leben erleichtern können, ob Sie ein Java-Programmierer, ein Skript-Fan oder ein agiler Entwickler sind.

Wir sind der festen Überzeugung, dass es nur eine einzige Möglichkeit gibt, eine Programmiersprache zu erlernen: durch Ausprobieren. Wir stellen eine Reihe von Skripten vor, um den Compiler, den Interpreter und die Shells zu demonstrieren, führen dann einige Plugins für gebräuchliche IDEs auf, und verraten danach, wo Sie die neuesten Informationen über Groovy finden können.

Wenn Sie dieses Kapitel durchgelesen haben, verfügen Sie über das Basiswissen, um zu verstehen, was Groovy ist und wie Sie damit experimentieren können.

Wir, die Autoren, Gutachter und Editoren, wünschen Ihnen viel Spaß beim Programmieren mit Groovy und bei der Verwendung dieses Buchs als Anleitung und Referenz.

1.1 Die Groovy-Story

Anlässlich der GroovyOne 2004 – eines Treffens von Groovy-Entwicklern in London – berichtete James Strachan in einer Grundsatzrede, wie er auf die Idee kam, Groovy zu erfinden.

Vor einiger Zeit warteten er und seine Frau auf ein sich verspätendes Flugzeug. Während sie beim Shopping war, besuchte er ein Internet-Café und fasste spontan den Entschluss, auf die Website von Python zu gehen, um Python zu lernen. Diese Beschäftigung zog ihn mehr und mehr in ihren Bann, denn als gewiefter Java-Programmierer erkannte er schnell, dass in Java viele der interessanten und nützlichen Fähigkeiten von Python fehlten, wie zum Beispiel die native Sprachunterstützung für gebräuchliche Datentypen in einer ausdrucksstarken Syntax und, noch wichtiger, dynamisches Verhalten. So entstand die Idee, diese Fähigkeiten in Java einzubringen.

Daraus ergaben sich die Hauptprinzipien für die Entwicklung von Groovy: Es sollte eine reichhaltige, Java-freundliche Sprache entstehen, die die attraktiven Vorteile dynamischer Sprachen auf einer robusten und gut unterstützten Plattform zur Verfügung stellt.

Abbildung 1.1 zeigt, wie diese einzigartige Kombination Groovy in der vielfältigen Welt der Sprachen für die Java-Plattform positioniert.[1] Wir möchten niemanden beleidigen, indem wir genau angeben, wo eine bestimmte Sprache unserer Meinung nach einzuordnen ist. Doch was die Position von Groovy betrifft, sind wir uns sicher.

Manche Sprachen verfügen vielleicht über etwas mehr Funktionen als Groovy, und andere können von sich behaupten, die bessere Java-Integration zu bieten. Wenn Sie jedoch beide

[1] Auf http://www.robert-tolksdorf.de/vmlanguages.html sind fast 200 (!) Sprachen für die Java Virtual Machine aufgeführt.

Aspekte gemeinsam betrachten, kann zurzeit keine Sprache Groovy das Wasser reichen. Nirgendwo ist eine bessere Kombination von Java-Freundlichkeit *plus* einer vollständigen Bandbreite moderner Sprachfähigkeiten zu finden.

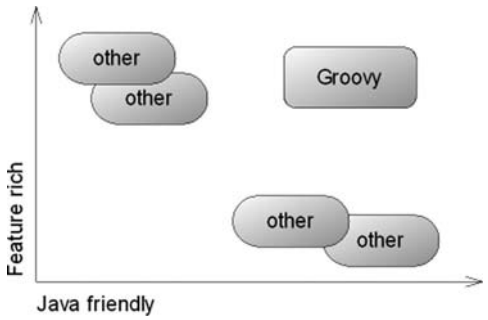

Abbildung 1.1 Die Landschaft der JVM-Sprachen. Groovy ist reichhaltig *und* Java-freundlich, und beides in hohem Maße, ohne das eine für das andere zu opfern.

Nachdem nun einige Ziele von Groovy bekannt sind, wollen wir betrachten, was es *ist*.

1.1.1 Was ist Groovy?

Die Website von Groovy (*http://groovy.codehaus.org*) gibt eine der besten Definitionen von Groovy: „Groovy ist eine agile, dynamische Sprache für die Java-Plattform mit vielen Fähigkeiten, die von Sprachen wie Python, Ruby und Smalltalk inspiriert sind, wodurch diese den Java-Entwicklern in einer Java-ähnlichen Syntax zugänglich werden."

Groovy wird oft als Skriptsprache bezeichnet und eignet sich tatsächlich sehr gut für Skripte. Doch es wäre ein Fehler, Groovy ausschließlich unter diesem Etikett abzuhandeln. Groovy lässt sich in Java-Bytecode vorkompilieren, in Java-Anwendungen integrieren, es kann Webanwendungen bereitstellen, mehr Kontrolle in Build-Dateien ermöglichen, und sogar selbst die Grundlage ganzer Anwendungen sein. Groovy ist so flexibel, dass es in keine Schublade passt.

Eines können wir mit Gewissheit über Groovy sagen: Es ist eng mit der Java-Plattform verbunden. Das gilt sowohl für die Implementierung (viele Teile von Groovy sind in Java geschrieben, der Rest in Groovy selbst), als auch für die Interaktion. Wenn Sie in Groovy programmieren, schreiben Sie in vieler Hinsicht eine spezielle Art von Java-Code. Alle Mächtigkeit der Java-Plattform, einschließlich der vielen verfügbaren Bibliotheken, steht als Rüstzeug bereit.

Ist Groovy deshalb nur eine Schicht syntaktischer Zuckerguss? Keinesfalls. Zwar *könnte* alles, was Sie mit Groovy tun, auch mit Java getan werden, aber es wäre Irrsinn, den Java-Code zu schreiben, der erforderlich wäre, um die Zaubertricks von Groovy nachzubilden. Groovy leistet viel Arbeit hinter den Kulissen, um seine Agilität und Dynamik zu erreichen. Stellen Sie sich bitte bei der Lektüre dieses Buches immer wieder vor, was alles erforderlich wäre, um mit Java die Effekte von Groovy nachzuahmen. Viele Groovy-Fähigkeiten, die auf den ersten Blick ungewöhnlich erscheinen – Logik auf natürliche Weise in Objekten kapseln, Hierarchien mit kaum mehr als dem *Minimum* an Programmierarbeit erstellen, das für die

Verarbeitung der Daten unerlässlich ist, Datenbankabfragen in der normalen Anwendungssprache ausdrücken, bevor sie in SQL übersetzt werden, das Laufzeitverhalten einzelner Objekte nach ihrer Erzeugung noch ändern – sind alles Aufgaben, die Java gar nicht leisten kann. Vielleicht gefällt Ihnen das Bild von Groovy als „bunter Sprache“, im Gegensatz zu dem eher einfarbigen Java: Das Wunder besteht darin, dass die farbigen Bilder aus Massen von sorgfältig gestalteten schwarz-weißen Punkten entstehen.

Wir werden nun genauer betrachten, was Groovy so attraktiv macht. Als Erstes werden wir sehen, wie Groovy und Java Hand in Hand arbeiten.

1.1.2 Gutes Zusammenspiel mit Java: nahtlose Integration

Java-Freundlichkeit bedeutet zweierlei: nahtlose Integration in die Laufzeitumgebung von Java (Java Runtime Environment) und eine auf Java abgestimmte Syntax.

Nahtlose Integration

Abbildung 1.2 zeigt den Integrationsaspekt von Groovy: Es läuft innerhalb der Java Virtual Machine und nutzt die Java-Bibliotheken (beides zusammen wird als Java Runtime Environment oder *JRE* bezeichnet). Groovy ist nur eine neue Art, *normale* Java-Klassen anzulegen. Aus der Perspektive der Laufzeit *ist* Groovy Java mit einer zusätzlichen Jar-Datei als Abhängigkeit.

Abbildung 1.2 Groovy und Java passen zusammen wie Nut und Feder

Folglich ist es kein Problem, Java aus Groovy heraus aufzurufen. Beim Programmieren in Groovy werden Sie dies andauernd tun, ohne es auch nur zu bemerken. Jeder Typ von Groovy ist ein Subtyp von `java.lang.Object`. Jedes Objekt von Groovy ist eine ganz normale Instanz eines Typs. Ein Datum in Groovy *ist* ein `java.util.Date`, und so weiter.

Die Integration in umgekehrter Richtung ist ebenso einfach. Angenommen, eine Groovy-Klasse namens `MyGroovyClass` wird in eine *.class-Datei kompiliert und in den Klassenpfad gelegt. Diese Groovy-Klasse könnten Sie dann in einer Java-Klasse folgendermaßen verwenden:

```
new MyGroovyClass();    // Erzeugung aus Java
```

Mit anderen Worten: Eine Groovy-Klasse wird genau wie eine Java-Klasse instanziert, denn schließlich *ist* sie ja auch eine Java-Klasse. Danach können Sie Methoden auf der Instanz aufrufen, die Referenz als Argument an Methoden übergeben und so weiter, während sich

die JVM in völliger Unkenntnis der Tatsache befindet, dass der Code in Groovy geschrieben wurde.

Ausrichtung der Syntax

Die zweite Dimension der Java-Freundlichkeit von Groovy ist seine Syntaxausrichtung. Um zu demonstrieren, was diese Ausrichtung bedeuten *sollte*, wollen wir die verschiedenen Mechanismen vergleichen, mit denen das heutige Datum in Java, Groovy und Ruby abgerufen wird:

```
import java.util.*;           // Java
Date today = new Date();      // Java

today = new Date()            // ein Groovy-Skript

require 'date'                 # Ruby
today = Date.new               # Ruby
```

Die Lösung von Groovy ist kurz, präzise und kompakter als normales Java. Groovy muss weder das Package `java.util` importieren noch den Typ `Date` angeben; ja mehr noch, Groovy benötigt nicht einmal Semikola, wenn es den Code auch ohne sie verstehen kann – und obwohl es kompakter ist, ist Groovy für einen Java-Programmierer völlig verständlich.

Die Ruby-Lösung wird aufgelistet, um zu veranschaulichen, was Groovy vermeidet: ein anderes Konzept für Packages (`require`), eine andere Kommentarsyntax und eine andere Syntax für die Objekterzeugung. Ruby ist in sich zwar stimmig (und vielleicht sogar konsistenter als Java), aber es passt nicht so gut wie Groovy zur Syntax und Architektur von Java.

Nun wissen Sie, was Java-Freundlichkeit in Bezug auf die Integration und Anpassung der Syntax bedeutet. Doch wie steht es mit der Reichhaltigkeit?

1.1.3 Power in Ihrem Code: eine reichhaltige Sprache

Alle Fähigkeiten von Groovy aufzulisten wäre ein bisschen so, als wolle man alle möglichen Bewegungen eines Tänzers aufzählen. Zwar ist jede Fähigkeit an sich bereits wichtig, doch die Strahlkraft von Groovy ergibt sich aus dem Zusammenspiel. Zusätzlich zu denen von Java besitzt Groovy drei Typen von Merkmalen: Sprachfähigkeiten, Groovy-spezifische Bibliotheken und Ergänzungen zu den bestehenden Standardklassen von Java (GDK). Abbildung 1.3 zeigt einige dieser Merkmale und wie sie zusammenpassen. Die schattierten Kreise zeigen, wie die Merkmale einander benutzen. So stützen sich beispielsweise einige der Bibliotheksfunktionen stark auf Sprachfähigkeiten. Idiomatischer Groovy-Code verwendet nur selten eine Fähigkeit isoliert, sondern normalerweise mehrere im Verbund, wie die Töne in einem Akkord.

Leider lassen sich viele dieser Fähigkeiten nicht mit wenigen Worten erklären. So sind zum Beispiel *Closures,*in Groovy ein Sprachkonzept von unschätzbarem Wert, doch das Wort an sich ist nichtssagend. Wir werden jetzt noch nicht in die Einzelheiten gehen, doch die folgenden Beispiele sollen Ihnen einen ersten Eindruck vermitteln.

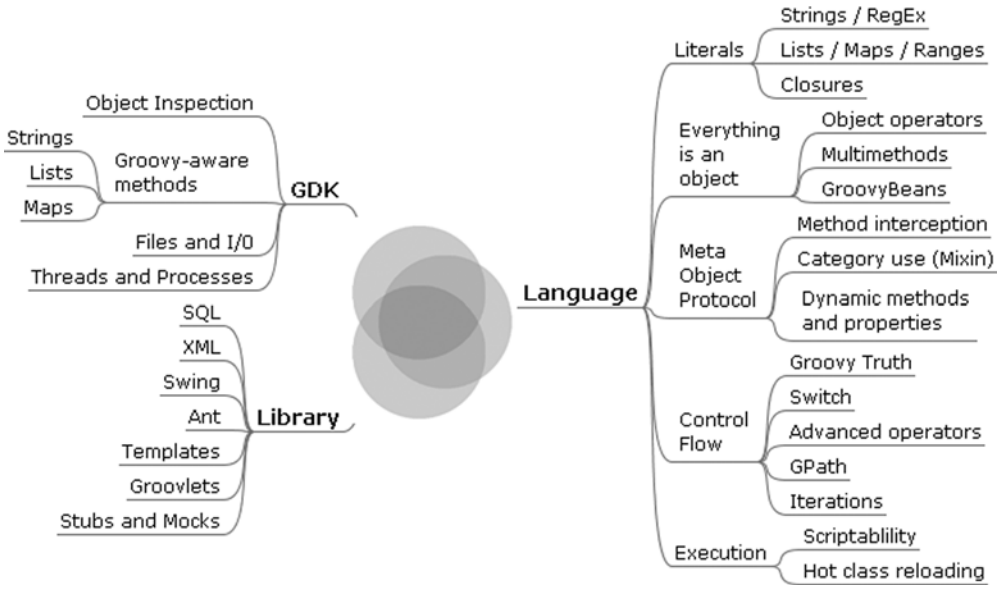

Abbildung 1.3 Viele zusätzliche Bibliotheken und JDK-Verbesserungen in Groovy bauen auf den neuen Sprachfähigkeiten auf. Die Kombination der drei bildet einen „Sweet Spot“ für klaren und mächtigen Code.

Eine Datei auflisten: Closures und Neuerungen in I/O

Closures sind Codeblöcke, die als Objekte erster Güte behandelt werden können: Man kann sie als Referenzen übergeben, speichern, zu beliebiger Zeit ausführen und so weiter. In Java werden oft die anonymen inneren Klassen auf diese Weise genutzt, insbesondere mit Adapterklassen, doch die Syntax innerer Klassen ist hässlich, und ihre Fähigkeiten in Bezug auf Datenzugriff und -änderung sind beschränkt.

In Groovy ist der Umgang mit Dateien durch das Hinzufügen mehrerer neuer Methoden zu den Klassen des Packages `java.io` viel einfacher. Ein gutes Beispiel dafür ist die Methode `File.eachLine`: Wie oft mussten Sie nicht schon eine Datei zeilenweise lesen, und danach wieder schließen? Diese Aufgabe kommt so häufig vor, dass sie keinerlei Schwierigkeiten bereiten sollte, und in Groovy tut sie es auch nicht.

Fügen wir nun die beiden Fähigkeiten zusammen, um ein vollständiges Programm zu schreiben, das eine Datei mit Zeilennummern auflistet:

```
def number=0
new File ('test.groovy').eachLine { line ->
    number++
    println "$number: $line"
}
```

Die Closure in den geschweiften Klammern wird für jede Zeile ein einziges Mal ausgeführt, und zwar durch Vermittlung der neuen `eachLine`-Methode der Klasse `File`.

Ausgabe einer Liste: Collection-Literale und vereinfachter Zugriff auf Eigenschaften

`java.util.List` und `java.util.Map` sind wohl die meistgenutzten Interfaces in Java, doch die Sprache bietet wenig Unterstützung für sie. In Groovy können Sie Listen- und Map-Literale mit derselben Leichtigkeit wie String- oder Zahlenliterale verwenden, und haben darüber hinaus viel mehr Methoden in den Collection-Klassen zur Verfügung.

In ähnlicher Weise sind zwar die JavaBean-Konventionen für Eigenschaften in Java fast allgegenwärtig, werden aber von der Sprache nicht genutzt. Groovy vereinfacht den Zugriff auf Eigenschaften, sodass der Code viel leichter lesbar wird.

Das folgende Beispiel nutzt diese beiden Fähigkeiten, um für jede Klasse aus einer Klassenliste das dazugehörige Package auszugeben. Beachten Sie, dass das Wort *Package* in Anführungszeichen gesetzt werden muss, da es ein Schlüsselwort ist, aber auch als Name einer Property verwendet werden kann. In Java könnte man zwar in einer ähnlichen Anfangszeile ein Array deklarieren, doch hier verwenden wir eine richtige Liste, mit Elementen, die sich ohne zusätzliche Arbeit hinzufügen oder entfernen lassen:

```
def classes = [String, List, File]
for (clazz in classes)
{
    println clazz.'package'.name
}
```

In Groovy könnten Sie sogar solche banalen `for`-Schleifen vermeiden, indem Sie Property-Zugriff auf eine Liste anwenden. Das Resultat ist eine Liste der Properties. Mit dieser Funktion lässt sich das Äquivalent des obigen Codes wie folgt formulieren:

```
println( [String, List, File].'package'.name )
```

Dies erzeugt die Ausgabe:

```
["java.lang", "java.util", "java.io"]
```

Ziemlich cool, oder?

XML-Behandlung nach Groovy-Art: GPath mit dynamischen Eigenschaften

Ob beim Lesen oder Schreiben, der Umgang mit XML macht in Java viel Arbeit. Alternativen zum W3C DOM machen Ihnen zwar das Leben leichter, aber Java selbst steuert als Sprache nichts dazu bei, es kann sich nicht an Ihre Bedürfnisse anpassen. In Groovy können sich Klassen so verhalten, als hätten sie zur Laufzeit Properties, selbst wenn die Namen dieser Properties beim Kompilieren der Klasse noch nicht bekannt sind. `GPath` baut auf dieser Fähigkeit auf und ermöglicht eine nahtlose, XPath-ähnliche Navigation in XML-Dokumenten.

Angenommen, Sie haben eine Datei namens customers.xml, die folgendermaßen aussieht:

```
<?xml version="1.0" ?>
<customers>
  <corporate>
    <customer name="Bill Gates"        company="Microsoft" />
    <customer name="Steve Jobs"        company="Apple" />
    <customer name="Jonathan Schwartz" company="Sun" />
  </corporate>
```

```
  <consumer>
    <customer name="John Doe" />
    <customer name="Jane Doe" />
  </consumer>
</customers>
```

Sie können alle Firmenkunden mit Namen und Firma ausgeben, ohne mehr als den folgenden Code zu benötigen. (Die Datei schon gleich mit dem Groovy-`Builder` zu generieren, wäre ebenfalls beträchtlich einfacher, als sie in Java zu erstellen.)

```
def customers = new XmlSlurper().parse(new File('customers.xml'))
for (customer in customers.corporate.customer)
{
    println "${customer.@name} arbeitet für ${customer.@company}"
}
```

Schon bei der Demonstration nur einiger weniger Fähigkeiten von Groovy haben Sie in den vorangegangenen Beispielen auch andere Funktionen gesehen, zum Beispiel String-Interpolation mit `GString`, einfachere `for`-Schleifen, optionale Typisierung sowie optionale Anweisungsbegrenzer und Klammern, um nur einige zu nennen. Die Funktionen arbeiten so gut zusammen und sind so eingängig, dass Sie kaum merken, dass Sie sie verwenden.

Auch wenn die Java-Freundlichkeit und die Reichhaltigkeit die Hauptantriebskräfte für Groovy sind, gibt es noch weitere bemerkenswerte Aspekte. Bisher haben wir uns auf die harten technischen Fakten von Groovy konzentriert, aber eine Sprache braucht mehr als das, um Erfolg zu haben: Sie muss *attraktiv* sein. In der Welt der Programmiersprachen kann niemand erwarten, dass ihm die Welt zu Füßen liegt, nur weil er eine bessere Lösung gefunden hat. Die Sprache muss sowohl Entwickler als auch Manager ansprechen, und jeden auf seine Art.

1.1.4 Community-gesteuert, aber von Unternehmen gestützt

Manch einer findet es beruhigend zu wissen, dass seine Investition in eine Sprache dadurch gesichert ist, dass sie als Standard anerkannt wird. Durch diese Zusicherung hebt sich Groovy ab, denn seit der Annahme des JSR-241 ist Groovy die zweite Standardsprache für die Java-Plattform (die erste ist Java selbst).

Ein zweites Kriterium ist die Benutzerbasis: Je mehr Nutzer es gibt, umso größer ist die Chance für guten Support und nachhaltige Entwicklung. Groovy verfügt über eine ausreichend große Benutzerbasis. Ein guter Indikator dafür ist die Aktivität auf den Mailinglisten und die Anzahl der Groovy-Projekte (siehe *http://groovy.codehaus.org/Related+Projects*).

Doch Attraktivität ist mehr als nur Strategie. Über das Messbare hinaus muss die Sprache ein gutes Gefühl vermitteln, das dafür sorgt, dass Programmieren Spaß macht.

Die Entwickler von Groovy kennen dieses Gefühl und achten sorgfältig darauf, wenn sie über Sprachfähigkeiten entscheiden. Dass die Sprache so heißt, hat schließlich einen Grund.

Groovy

„Eine Situation oder Aktivität, die man genießt oder für die man prädestiniert ist („er fand seinen Groove als Bassist in einem Trio“). Eine sehr angenehme Erfahrung; Spaß haben (Musik hören, die „groovy“ ist). Freudige Erregung verspüren. Harmonisch reagieren oder interagieren.“ [übersetzt aus Leo]

Kürzlich sagte jemand, Groovy verbinde „den Stil von Java mit dem Feeling von Ruby“. Besser kann man es gar nicht beschreiben. Wenn Sie mit Groovy arbeiten, ist die Sprache Ihr Partner, und kein Gegner, mit dem Sie kämpfen müssen, um Ihre klaren Gedanken irgendwie für den Computer verständlich zu machen.

Nun ist es zwar schön und gut, sich „groovy“ zu fühlen, aber deswegen müssen Sie immer noch Ihre Rechnungen bezahlen. Im nächsten Abschnitt werden Sie einige der praktischen Vorzüge kennen lernen, die Groovy für Ihr Berufsleben bringt.

1.2 Was Groovy für Sie tun kann

Je nach Vorwissen und Erfahrung interessieren sich verschiedene Leser wahrscheinlich für verschiedene Fähigkeiten von Groovy. Kaum jemand wird jeden einzelnen Aspekt von Groovy für die alltägliche Arbeit benötigen, ebenso wie niemand jeden Tag das gesamte Mammut-Framework der Java-Standardbibliotheken braucht.

Dieser Abschnitt präsentiert interessante Merkmale von Groovy und ihre Anwendungsgebiete für Java-Kenner, Skriptprogrammierer und pragmatische, extreme sowie agile Programmierer. Uns ist bewusst, dass Entwickler an ihren Arbeitsstellen meist mehrere Rollen spielen, und nicht selten abwechselnd in unterschiedliche Identitäten schlüpfen müssen. Es ist hilfreich zu sehen, wie Sie von Groovy in den typischen Situationen, die diese verschiedenen Rollen mit sich bringen, profitieren können.

1.2.1 Groovy für Java-Profis

Wenn Sie sich als Java-Experten betrachten, haben Sie wahrscheinlich viele Jahre Erfahrung in der Java-Programmierung. Sie kennen alle wichtigen Teile des Java Runtime APIs und wahrscheinlich auch die APIs vieler anderer Java-Packages.

Doch wenn Sie ehrlich sind, können Sie dieses Wissen nicht immer nutzen. Nehmen wir zum Beispiel ist die alltägliche Aufgabe, alle Dateien unterhalb des aktuellen Verzeichnisses rekursiv zu durchsuchen. Wenn es Ihnen ebenso geht wie uns, ist das Programmieren einer derartigen Ad-hoc-Aufgabe in Java einfach zu anstrengend.

In Groovy können Sie hingegen, wie in diesem Buch noch zu sehen sein wird, rasch die Konsole öffnen und mit folgendem Code

```
groovy -e "new File('.').eachFileRecurse { println it }"
```

alle Dateinamen rekursiv ausgeben.

Selbst wenn Java eine `eachFileRecurse`-Methode und ein passendes `FileListener`-Interface hätte, müssten Sie immer noch explizit eine Klasse erzeugen, eine `main`-Methode deklarieren, den Code als Datei speichern und ihn kompilieren; erst danach könnten Sie ihn ausführen. Schauen wir uns zum Vergleich einmal an, wie der Java-Code aussähe, wenn es eine geeignete `eachFileRecurse`-Methode darin gäbe:

```
public class ListFiles {                          // JAVA !!
    public static void main(String[] args) {
        new java.io.File(".").eachFileRecurse( <┐  Wenn Java
            new FileListener() {                │  dies hätte
                public void onFile (File file) {
                    System.out.println(file.toString());
                }
            }
        );
    }
}
```

Beachten Sie, wie der Zweck des Codes (nämlich alle Dateien auszugeben) durch den sperrigen Code verschleiert wird, den Java benötigt, um zu guter Letzt zu einem vollständigen Programm zu kommen.

Neben der Verfügbarkeit der Kommandozeile und der Schönheit des Codes bietet Groovy auch dynamisches Verhalten für Java-Anwendungen, zum Beispiel durch das Ausdrücken von Geschäftsregeln, das Ermöglichen von intelligenten Konfigurationen oder sogar durch das Implementieren von *domänenspezifischen Sprachen*.

Sie haben die Möglichkeiten, statische oder dynamische Typen zu verwenden und mit vorkompiliertem Code oder einfachem Groovy-Quellcode zu arbeiten, den Sie nach Bedarf kompilieren. Als Entwickler können Sie entscheiden, wo und wann Sie Ihre Lösung „festklopfen" möchten und wo sie flexibel sein soll – in Groovy haben Sie die Wahl.

Dies sollte Ihnen genügend Sicherheit geben, um Groovy bedenkenlos in Ihre Projekte zu integrieren und von seinen Fähigkeiten zu profitieren.

1.2.2 Groovy für Skriptprogrammierer

Als Skriptprogrammierer haben Sie vielleicht schon in Perl, Ruby, Python oder anderen dynamischen (nicht für das Skripting geschaffenen) Sprachen wie Smalltalk, Lisp oder Dylan programmiert.

Doch der Marktanteil der Java-Plattform ist nicht zu leugnen, und Leute wie Sie arbeiten häufig auch mit Java, um sich ihre Brötchen zu verdienen. Firmenkunden betreiben oft eine Java-Standardplattform (z.B. J2EE), und erlauben nichts anderes als Java für die Entwicklung und Produktionsumgebung. Da Sie keine Chance sehen, hier Ihre superschlanke Skriptlösung einzufädeln, müssen Sie in den sauren Apfel beißen, die Ärmel hochkrempeln und sich durch endlosen Java-Code pflügen, während Sie andauernd denken: „Wenn ich doch nur [*hier Ihre Sprache einsetzen*] hätte, dann könnte ich diese ganze Methode durch eine einzige Zeile ersetzen!" Wir gestehen, dass auch wir diese Frustration bereits durchgemacht haben.

Hier schafft Groovy Erleichterung und gibt Ihnen die Freude am Programmieren zurück, indem es fortgeschrittene Sprachfähigkeiten dort zur Verfügung stellt, wo sie gebraucht werden: in der alltäglichen Arbeit. In Groovy können Sie Methoden *auf allem und jedem* aufrufen, Codeblöcke zur sofortigen oder späteren Ausführung übergeben, den bestehenden Bibliothekscode durch eigene Spezialsemantik ergänzen und eine Unmenge weiterer Fähigkeiten nutzen. So können Sie sich klar ausdrücken und mit wenig Code Wunder wirken.

Setzen Sie einfach die Datei *groovy-all-*.jar* in den Klassenpfad Ihres Projekts, und schon sind Sie am Ziel.

Heutzutage ist Softwareentwicklung nicht mehr das Werk Einzelner, und Ihre Teamkollegen (und Ihr Chef) möchten wissen, was Sie mit Groovy tun und was Groovy überhaupt ist. Dieses Buch ist eine Quelle, die Sie an andere weitergeben können, damit auch diese daraus lernen. (Wenn Sie den Gedanken nicht ertragen, sich von diesem schönen Buch zu trennen, können Sie den Kollegen natürlich auch empfehlen, es selbst anzuschaffen. Das wäre uns nur recht.)

1.2.3 Groovy für pragmatische, extreme und agile Programmierer

Wenn Sie in diese Kategorie gehören, verfügen Sie wahrscheinlich bereits über ein überquellendes Bücherregal, eine riesige Zettelkartei voller Aufgaben und eine automatisierte Testsuite, die jeden Moment auf Rot umspringen kann. Der Release der nächsten Iteration steht unmittelbar bevor und Sie haben kaum die Zeit, über Groovy nachzudenken. Die bloße Erwähnung dieses Wortes lässt Ihre Kollegen bereits an Ihrem Geisteszustand zweifeln.

Wenn wir eines über pragmatische, extreme oder agile Programmierung wissen, dann dies: Dass Sie von Zeit zu Zeit einen Schritt zurücktreten, zur Ruhe kommen und einschätzen müssen, ob Ihre Werkzeuge noch genügend *Biss* haben. Trotz ewig enger Termine müssen Sie Ihre *Messer regelmäßig wetzen*. Für Software bedeutet das, dass Sie das nötige Wissen und die Ressourcen besitzen und für jede Aufgabe die richtigen Verfahren, Tools und Technologien nutzen.

Groovy wird in Ihrem Werkzeugkasten unschätzbar für alle Automatisierungen sein, die Sie in Ihren Projekten vornehmen müssen, angefangen von einem einfachen Build-Automatismus über die laufende Integration und Berichterstattung bis hin zur automatisierten Dokumentation, Lieferung und Installation. Groovys Automatisierungsunterstützung stärkt die Mächtigkeit vorhandener Lösungen wie Ant und Maven, stellt aber zugleich eine einfache und klare Sprache zu ihrer Steuerung zur Verfügung. Groovy hilft sogar beim Testen, sowohl auf Unit Test-Ebene als auch auf funktionaler Ebene, damit auch wir Testfanatiker uns so richtig zu Hause fühlen.

Kaum eine andere Programmierschule ist so rigoros und aufmerksam in Bezug auf selbsterklärenden Code, der seine Absichten klar erkennen lässt. Wir haben ein beinahe körperliches Bedürfnis, Doppelungen zu vermeiden und einfachere Lösungen zu erstreben. Dabei kann Groovy außerordentlich hilfreich sein.

Vor Groovy benutzte ich (Dierk) andere Skriptsprachen (vorzugsweise Ruby), um Entwurfsideen zu skizzieren, mit einem Programmierexperiment (einem so genannten *Spike*) die Machbarkeit einer Lösung zu testen und einen funktionalen Prototyp auszuführen. Der Nachteil dabei: Ich war nie sicher, ob das, was ich da schrieb, auch in Java funktionieren würde. Ja schlimmer noch, am Ende musste ich das Ganze portieren oder von vorne anfangen. Mit Groovy kann ich auch die Entwurfsarbeit *direkt* auf meiner Zielplattform erledigen.

Beispiel

Vor Kurzem führten Guillaume und ich einen Spike zur *Primfaktorzerlegung*[2] durch. Wir begannen mit einer kurzen Groovy-Lösung, die den Job zwar sauber, aber nicht effizient erledigte. Mit den Interception-Funktionen von Groovy führten wir einen Unit Test der Lösung durch und zählten die Operationen. Da der Code sauber war, war es eine Kleinigkeit, die Lösung zu optimieren und die Zahl der Operationen zu reduzieren. In Java wäre das Optimierungspotenzial bei weitem nicht so leicht erkennbar gewesen. Das Endresultat kann von Java ohne Umstände übernommen werden, und obwohl wir immer noch die Möglichkeit haben, die optimierte Lösung in reines Java zu portieren, was eine zusätzliche Performance-Steigerung bewirken würde, können wir diese Entscheidung so lange aufschieben, bis sich die Notwendigkeit ergibt.

Das nahtlose Zusammenspiel von Groovy und Java eröffnet zwei Dimensionen der Code-Optimierung: Java für Code zu verwenden, der für die Laufzeitperformance optimiert werden muss, und Groovy für Code, der möglichst flexibel und lesbar sein soll.

Neben diesen greifbaren Vorteilen ist es ein Wert an sich, Groovy zu lernen. Es wird Ihnen neue Lösungen aufzeigen und beim Entwickeln von Software, gleichgültig in welcher Sprache, neue Konzepte zu erkennen helfen.

Unabhängig davon, welcher Programmierertyp Sie sind, hoffen wir, dass Sie nun gerne etwas Groovy-Code zwischen die Finger bekommen möchten. Wenn Sie sich nicht bremsen können, schauen Sie doch in Kapitel 2 bereits einige echte Groovy-Beispiele an.

1.3 Groovy ausführen

Zuerst möchten wir einige der Tools einführen, die Sie benötigen, um Groovy auszuführen und optional auch zu kompilieren. Wenn Sie diese beim Lesen bereits ausprobieren möchten, müssen Sie natürlich zuvor Groovy installieren. Anhang A führt Sie durch den Installationsprozess.

Zum Ausführen von Groovy-Code und -Skripten sind drei Befehle notwendig, wie in Tabelle 1.1 gezeigt. Die drei verschiedenen Mechanismen, Groovy auszuführen, werden in den folgenden Abschnitten mit Beispielen und Screenshots gezeigt. Groovy kann auch wie jedes normale Java-Programm „laufen“, wie Sie in Abschnitt 1.4.2 sehen werde, und darüber hinaus gibt es eine spezielle Ant-Integration, die in Abschnitt 1.4.3 erläutert wird.

In Kapitel 11 werden Sie mehrere Möglichkeiten kennen lernen, um Groovy in Java-Programme zu integrieren.

[2] Jede Ordinalzahl N lässt sich eindeutig in ihre Primfaktoren zerlegen: N = p1*p2*p3. Das Problem der Zerlegung ist bekanntermaßen schwierig. Seine Komplexität schützt einige kryptographische Algorithmen, wie den bekannten Rivest-Shamir-Adleman (RSA)-Algorithmus.

Tabelle 1.1 Befehle zum Ausführen von Groovy

Befehl	Wirkung
`groovysh`	Startet die Kommandozeilenshell `groovysh`, in der Groovy-Code interaktiv ausgeführt werden kann. Indem Sie Anweisungen oder ganze Skripte Zeile für Zeile in die Shell eingeben und den Befehl `go` erteilen, wird der Code „on the fly“ ausgeführt.
`groovyConsole`	Startet eine Benutzeroberfläche, auf der Groovy-Code interaktiv ausgeführt werden kann. Zudem lädt `groovyConsole` Groovy-Skriptdateien und führt sie aus.
`groovy`	Startet den Interpreter, der Groovy-Skripte ausführt. Einzeilige Groovy-Skripte können als Kommandozeilenargumente angegeben werden.

1.3.1 „Hello World“ auf Groovysh

Schauen wir uns als Erstes `groovysh` an, da die Shell so ein praktisches Tool für Groovy-Experimente ist. Es ist ganz leicht, Groovy darin zu bearbeiten und iterativ auszuführen. Dadurch lässt sich ohne Scriptingaufwand erkennen, wie Groovy funktioniert.

Um die Shell zu starten, führen Sie `groovysh` (UNIX) oder `groovysh.bat` (Windows) auf der Kommandozeile aus. Dann sollte folgender Prompt angezeigt werden:

```
Lets get Groovy!
================
Version: 1.0-RC-01-SNAPSHOT JVM: 1.4.2_05-b04
Type 'exit' to terminate the shell
Type 'help' for command help
Type 'go' to execute the statements

groovy>
```

Das traditionelle „Hello World!“-Programm kann in Groovy mit einer einzigen Zeile formuliert und dann in `groovysh` mit dem Befehl `go` ausgeführt werden:

```
groovy> "Hello, World!"
groovy> go

===> Hello, World!
```

Der Befehl `go` ist einer der wenigen, den die Shell kennt. Die übrigen können Sie anzeigen, indem Sie `help` auf der Kommandozeile eingeben:

```
groovy> help
Available commands (must be entered without extraneous characters):
exit/quit      - terminates processing
help           - displays this help text
discard        - discards the current statement
display        - displays the current statement
explain        - explains the parsing of the current statement (currently
                 disabled)
execute/go     - temporary command to cause statement execution
binding        - shows the binding used by this interactive shell
discardclasses - discards all former unbound class definitions
inspect        - opens ObjectBrowser on expression returned from
                 previous "go"
```

Die Befehle `go` und `execute` sind äquivalent. Der Befehl `discard` lässt Groovy die zuletzt eingegebene Zeile vergessen, was bei der Eingabe langer Skripte nützlich ist, da dieser Befehl das Löschen kleiner Codeabschnitte ermöglicht, ohne dass ein komplettes Skript von Anfang an neu geschrieben werden muss. Betrachten wir nun die übrigen Befehle.

Display

Der Befehl `display` zeigt die letzte Anweisung an, die keine Befehlsanweisung war:

```
groovy> display
1> "Hello World!"
```

Binding

Der Befehl `binding` zeigt Variablen an, die in einer `groovysh`-Session verwendet wurden. Wir haben in unserem einfachen Beispiel keine Variablen genutzt, werden aber zu Demonstrationszwecken „Hello World!“ durch die Variable `greeting` ersetzen, die an die Stelle der ausgegebenen Meldung tritt:

```
groovy> greeting = "Hello"
groovy> "${greeting}, World!"
groovy> go

===> Hello, World!

groovy> binding
Available variables in the current binding
greeting = Hello
```

Der Befehl `binding` ist nützlich, wenn Sie in einer längeren `groovysh`-Session stecken und nicht mehr genau wissen, welche Variablen mit welchen aktuellen Werten gerade in Gebrauch sind.

Um die Bindung zu löschen, schließen Sie die Shell und starten eine neue.

Inspect

Der Befehl `inspect` öffnet den *Groovy Object Browser* auf dem zuletzt ausgewerteten Ausdruck. Dieser Browser ist eine Swing-Benutzeroberfläche, auf der Sie mithilfe des Groovy-GDK das native Java API eines Objekts und zusätzliche Fähigkeiten, die ihm zur Verfügung stehen, durchsuchen können. Abbildung 1.4 zeigt, wie der Object Browser eine Instanz von `String` untersucht. Sie enthält Informationen über die Klasse `String` im Header und darüber hinaus zwei Tabellen mit den verfügbaren Methoden und Feldern.

Betrachten Sie die zweite und dritte Zeile. Eine Methode namens `center` steht für ein `String`-Objekt zur Verfügung. Sie nimmt den Parameter `Number` (zweite Zeile) und einen optionalen `String`-Parameter entgegen (dritte Zeile), und hat den Rückgabetyp `String`. Groovy definierte diese neue öffentliche Methode für die Klasse `String`.

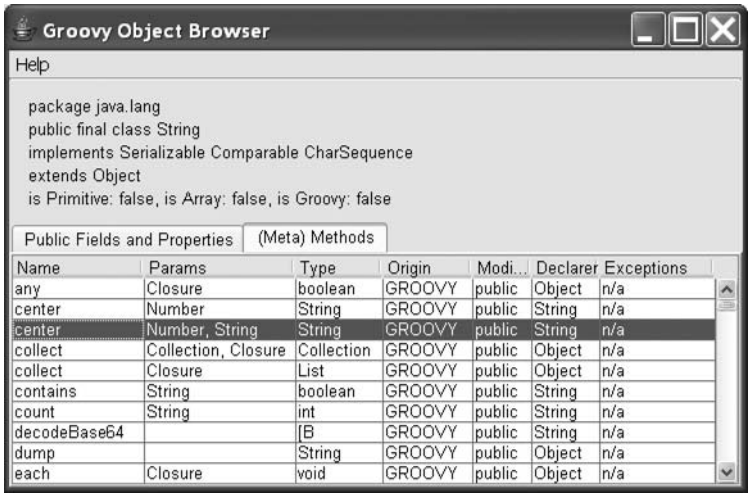

Abbildung 1.4 Wird der Groovy Object Browser auf einem Objekt vom Typ `String` geöffnet, zeigt er die Tabelle der im Bytecode verfügbaren Methoden und registrierten Metamethoden an.

Wenn Sie nicht völlig anders als wir gestrickt sind, können Sie es nun kaum noch erwarten, dieses neue Wissen über `groovysh` auszuprobieren und Folgendes einzugeben:

```
groovy> 'test'.center 20, '-'
groovy> go

===> --------test--------
```

Das ist fast so gut wie IDE-Support!

Um das Browsen zu vereinfachen, können Sie Spalten sortieren, indem Sie auf die Überschriften klicken und die Sortierreihenfolge mit einem zweiten Klick umkehren. Sie können nach mehreren Kriterien sortieren, indem Sie die Spaltenüberschriften der Reihe nach anklicken und durch Ziehen mit der Maus neu anordnen.

Zukünftige Versionen des Groovy Object Browsers werden vielleicht noch raffiniertere Fähigkeiten zur Verfügung stellen.

1.3.2 Die groovyConsole

Die `groovyConsole` ist eine Swing-Oberfläche, die als minimalistischer interaktiver Groovy-Interpreter fungiert. Sie unterstützt zwar nicht die Kommandozeilenoptionen von `groovysh`, aber sie verfügt über ein FILE-Menü, mit dem Sie Groovy-Skripte laden, erstellen und speichern können. Interessanterweise ist `groovyConsole` in Groovy geschrieben. Ihre Implementierung ist ein gutes Beispiel für die Anwendung von Builders, die in Kapitel 7 erläutert werden.

Die `groovyConsole` nimmt keine Argumente entgegen und startet ein zweigeteiltes Fenster, wie es in Abbildung 1.5 gezeigt wird. Im oberen Bereich nimmt die Konsole Tastatureingaben entgegen. Um ein Skript auszuführen, geben Sie entweder Strg+R oder Strg+Enter ein, oder Sie verwenden den RUN-Befehl aus dem Menü ACTION, um das Skript auszuführen. Ist

ein Teil des Skriptcodes markiert, wird nur dieser Teil ausgeführt. Diese Eigenschaft ist nützlich für das einfache Debugging oder *Single-Stepping*, da Sie nacheinander eine oder mehrere Zeilen markieren können.

Das FILE-Menü von `groovyConsole` verfügt über die Befehle NEW, OPEN, SAVE und EXIT. Der Befehl NEW öffnet ein neues `groovyConsole`-Fenster. Mit OPEN können Sie zu einem Groovy-Skript im Dateisystem navigieren und dieses im Bearbeitungsbereich öffnen und ausführen. Das SAVE-Kommando speichert den im Bearbeitungsfenster geöffneten Text in einer Datei. EXIT beendet die `groovyConsole`.

Der in Abbildung 1.4 gezeigte Groovy Object Browser steht auch in der `groovyConsole` zur Verfügung und operiert ebenfalls auf dem zuletzt ausgewerteten Ausdruck. Um den Browser zu öffnen, drücken Sie auf Strg+I (für *Inspect*) oder wählen den Befehl INSPECT aus dem Menü ACTIONS.

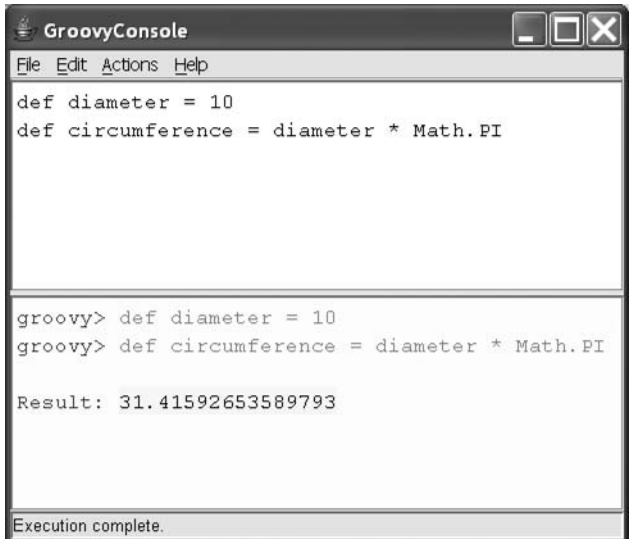

Abbildung 1.5 Die `groovyConsole` mit einem einfachen Skript, das aus dem Durchmesser den Umfang eines Kreises berechnet. Die Ausgabe steht im Ausgabefenster.

Soweit zur `groovyConsole`. Ob Sie lieber mit `groovysh` oder mit `groovyConsole` arbeiten, ist Ihre persönliche Entscheidung. Skriptprogrammierer, die ihre Arbeit auf der Kommandozeile erledigen, bevorzugen die Shell.

Unser Tipp

Ich (Dierk) habe mich umgewöhnt und verwende jetzt häufiger die Konsole, weil mir das Bearbeitungsfenster durch die Möglichkeit von Ausschneiden und Einfügen Tipparbeit erspart.

Wenn nichts Gegenteiliges gesagt wird, können Sie jedes Codebeispiel aus diesem Buch direkt in `groovysh` oder die `groovyConsole` übertragen und dort ausführen. Je öfter Sie das tun, umso schneller bekommen Sie ein Gefühl für die Sprache.

1.3.3 Der Befehl groovy

Der Befehl `groovy` führt in Groovy Programme und Skripte aus. So zeigt zum Beispiel das Listing 1.1 das Groovy-Programm, das die ersten 10 Zahlen der obligatorischen Fibonacci[3]-Zahlenfolge ausgibt. In der Fibonacci-Zahlenfolge sind die ersten beiden Zahlen 1 und 1, und jede nachfolgende Zahl ist die Summe ihrer beiden Vorgänger.

Wenn Sie das ausprobieren möchten, kopieren Sie den Code in eine Datei und speichern ihn unter dem Namen *Fibonacci.groovy*. Die Dateierweiterung spielt für die `groovy`-Executable keine große Rolle, doch es entspricht der Konvention, Namen von Groovy-Skripten mit der Erweiterung *.groovy* zu versehen. Ein Vorteil dieser Erweiterung ist es, dass Sie sie auf der Kommandozeile bei der Angabe des Skriptnamens weglassen und statt `groovy MyScript.groovy` einfach `groovy MyScript` ausführen können.

Listing 1.1 Fibonacci.groovy

```
current = 1
next    = 1
10.times {                    // 10-mal rechnen
   print current + ' '
   newCurrent = next
   next = next + current
   current = newCurrent
}
println ''
```

Führen Sie diese Datei als Groovy-Programm aus, indem Sie dem Befehl `groovy` den Dateinamen übergeben. Dann müsste folgende Ausgabe erscheinen:

```
> groovy Fibonacci
1 1 2 3 5 8 13 21 34 55
```

Der Befehl `groovy` hat viele Zusatzoptionen, die für das Scripting auf der Kommandozeile praktisch sind. Ausdrücke können Sie zum Beispiel ausführen, indem Sie `groovy -e "println 1+1"` eingeben. Dann wird auf der Konsole `2` ausgegeben. In Abschnitt 12.3 wird die ganze Palette der Optionen mit vielen Beispielen vorgestellt.

Im vorliegenden Abschnitt ging es um den Support, den Groovy für das Ad-hoc-Scripting bietet, aber das ist noch längst nicht alles. Im nächsten Abschnitt erfahren Sie, wie Groovy sich in den Zyklus von Programmieren-Kompilieren-Ausführen einfügt.

[3] Leonardo Pisano (1170..1250), genannt Fibonacci, war ein Mathematiker aus Pisa (heute eine Stadt in Italien). Er führte seine Zahlenreihe ein, um das Wachstum einer isolierten Population von Kaninchen zu beschreiben. Vom biologischen Standpunkt aus mag das zweifelhaft sein, doch in den Künsten und Wissenschaften spielt seine Zahlenfolge eine große Rolle. Wenn Sie mehr darüber wissen möchten, abonnieren Sie *Fibonacci Quarterly*.

1.4 Groovy kompilieren und ausführen

Bisher haben wir Groovy nur im *Direktmodus* ausgeführt, in dem der Code direkt umgesetzt wird, ohne zuvor ausführbare Dateien zu produzieren, aber nun werden Sie eine zweite Art des Umgangs mit Groovy kennen lernen: Sie können es auch in Java-Bytecode kompilieren und als normale Java-Anwendung in einer Java Virtual Machine (JVM) ausführen. Dies ist der *vorkompilierte* Modus. In beiden Modi wird Groovy letzten Endes in einer JVM ausgeführt, und beide kompilieren den Groovy-Code in Java-Bytecode. Der Hauptunterschied ist der *Zeitpunkt* der Kompilierung und die Frage, ob die resultierenden Klassen im Arbeitsspeicher verwendet oder auf der Festplatte gespeichert werden.

1.4.1 Groovy mit groovyc kompilieren

Das Kompilieren von Groovy ist einfach, da Groovy mit einem Compiler namens `groovyc` ausgestattet ist, der für jede Groovy-Quelldatei mindestens eine Klassendatei generiert. So können wir zum Beispiel die Datei *Fibonacci.groovy* aus dem vorigen Abschnitt in normalen Java-Bytecode kompilieren, indem wir `groovyc` folgendermaßen auf der Skriptdatei ausführen:

```
> groovyc -d classes Fibonacci.groovy
```

In unserem Fall gibt der Groovy-Compiler zwei Java-Klassendateien in ein Verzeichnis namens *classes* aus, wie wir ihm mit dem Flag `-d` befohlen haben. Wenn das mit `-d` angegebene Verzeichnis nicht existiert, wird es angelegt. Wird der Compiler ausgeführt, wird der Name jeder generierten Klassendatei auf der Konsole ausgegeben.

Für jedes Skript erzeugt `groovyc` eine Klasse, die `groovy.lang.Script` erweitert und eine `main`-Methode enthält, damit `java` sie ausführen kann, wobei die kompilierte Klasse genauso wie das Skript heißt, das kompiliert wird.

Je nach dem Skriptcode können viele Klassen generiert werden, aber das braucht uns nicht zu kümmern, da es Sache der Java-Plattform ist. Im Wesentlichen funktioniert `groovyc` in derselben Weise, wie `javac` geschachtelte Klassen kompiliert.

Hinweis

Das Fibonacci-Skript enthält das Konstrukt `10.times{}`, das `groovyc` eine Klasse vom Typ *closure* generieren lässt, die alles, was in den geschweiften Klammern steht, implementiert. Diese Klasse ist in die Fibonacci-Klasse eingeschachtelt. Über Closures werden Sie in Kapitel 5 mehr erfahren. Wenn es Ihnen zurzeit noch verwirrend vorkommt, können Sie es vorläufig ignorieren.

Die Klassendateien werden wie in Tabelle 1.2 gezeigt auf Implementierungen abgebildet, und auch der Zweck der jeweiligen Klasse wird in Tabelle 1.2 erklärt.

Da wir nun ein kompiliertes Programm haben, wollen wir sehen, wie es ausgeführt wird.

Tabelle 1.2 Diese Klassen generiert `groovyc` für die Datei *Fibonacci.groovy*

Klassendatei	Unterklasse von ...	Zweck
`Fibonacci.class`	`groovy.lang.Script`	Enthält eine `main`-Methode, die mit dem Befehl java ausgeführt werden kann.
`Fibonacci$_run_closure1.class`	`groovy.lang.Closure`	Nimmt auf, was 10 *times* (also zehnmal) getan werden muss. Sie können dies ruhig ignorieren.

1.4.2 Ein kompiliertes Groovy-Skript mit Java ausführen

Ein kompiliertes Groovy-Programm wird genau wie ein kompiliertes Java-Programm ausgeführt, nur mit einer zusätzlichen Anforderung: Sie müssen in den Klassenpfad Ihrer JVM die einbettungsfähige Datei *groovy-all*.jar* aufnehmen, die dafür sorgt, dass alle Abhängigkeiten zwischen Groovy und Dritten zur Laufzeit automatisch aufgelöst werden. Auch das Verzeichnis, in dem Ihr kompiliertes Programm liegt, müssen Sie unbedingt in den Klassenpfad schreiben. Dann führen Sie das Programm wie jedes andere Java-Programm mit dem Befehl `java` aus.[4]

```
> java -cp %GROOVY_HOME%/embeddable/groovy-all-1.0.jar;classes Fibonacci
1 1 2 3 5 8 13 21 34 55
```

Beachten Sie, dass die Dateierweiterung *.class* für die Klasse `main` nicht angegeben werden sollte, wenn `java` ausgeführt wird.

Das mag nach einer Menge Arbeit aussehen, wenn Sie es gewohnt sind, Java-Code mit Ant auf Knopfdruck zu erstellen und auszuführen. Wir stimmen Ihnen zu, und deshalb haben die Entwickler von Groovy dafür gesorgt, dass Sie dies alles ganz einfach mit einem Ant-Skript erledigen können.

1.4.3 Kompilieren und Ausführen mit Ant

Groovy wird mit einer Ant-Task geliefert, die im `groovyc`-Compiler in eine Ant-Build-Skript ausgeführt wird. Hierzu müssen Sie Ant installieren,[5] am besten in der Version 1.6.2 oder höher.

Listing 1.2 zeigt ein Ant-Build-Skript, mit dem das Skript *Fibonacci.groovy* als Java-Bytecode erstellt und ausgeführt wird.

Listing 1.2 build.xml kompiliert ein Groovy-Programm als Java-Bytecode und führt es aus

```
<project name="fibonacci-build" default="run">

  <property environment="env"/>

  <path id="groovy.classpath">
    <fileset dir="${env.GROOVY_HOME}/embeddable/"/>
  </path>
```

❶ **Pfad-definition**

[4] Die hier gezeigte Kommandozeile gilt für Windows-Shells. Auf Linux/Solaris/UNIX/Cygwin hieße es:

```
java -cp $GROOVY_HOME/embeddable/groovy-all-1.0.jar:classes Fibonacci
```

[5] Groovy enthält eine eigene Kopie der Jar-Dateien von Ant, die man ebenfalls für diesen Zweck benutzen könnte, aber mit einer Standalone-Installation von Ant ist der Vorgang leichter zu erklären.

```
  <taskdef name ="groovyc"
    classname   ="org.codehaus.groovy.ant.Groovyc"
    classpathref="groovy.classpath"/>

  <target name="compile"
    description="compile groovy to bytecode">
    <mkdir  dir="classes"/>
    <groovyc
      destdir="classes"
      srcdir="."
      includes="Fibonacci.groovy"
      classpathref="groovy.classpath">
    </groovyc>
  </target>

  <target name="run" depends="compile"
    description="run the compiled class">
    <java classname="Fibonacci">
      <classpath refid="groovy.classpath"/>
      <classpath location="classes"/>
    </java>
  </target>
</project>
```

❷ Task-Def

❸ Compile-Ziel

❹ Run-Ziel

Speichern Sie diese Datei unter dem Namen *build.xml* in Ihrem aktuellen Verzeichnis, das auch das Skript *Fibonacci.groovy* enthalten sollte, und geben Sie `ant` an der Eingabeaufforderung ein.

Der Build beginnt beim ❹ `run`-Ziel, das von dem ❸ `compile`-Ziel abhängig ist und dieses folglich zuerst aufruft. Das `compile`-Ziel benutzt die `groovyc`-Task. Um Ant diese Task verständlich zu machen, ist die ❷ `taskdef` da. Sie findet die Implementierung der `groovyc`-Task im `groovy.classpath` in der ❶ Pfaddefinition.

Wird im ❸ `compile`-Ziel alles erfolgreich kompiliert, ruft das ❹ `run`-Ziel die `java`-Task auf den kompilierten Klassen auf.

Sie werden dann eine Ausgabe wie diese sehen:

```
> ant
Buildfile: build.xml

compile:
    [mkdir] Created dir: ...\classes
  [groovyc] Compiling 1 source file to ...\classes
run:
     [java] 1 1 2 3 5 8 13 21 34 55

  BUILD SUCCESSFUL
Total time: 2 seconds
```

Wenn Sie `ant` ein zweites Mal ausführen, erscheint keine Compiler-Ausgabe, denn die `groovyc`-Task ist klug genug, den Code *nur wenn nötig* zu kompilieren. Also müssen Sie für eine *saubere* Kompilierung das Zielverzeichnis vor dem Kompilieren löschen.

Die Ant-Task `groovyc` kennt viele Optionen, von denen die meisten denen der Ant-Task `javac` gleichen. Die Optionen `srcdir` und `destdir` sind obligatorisch.

Es kann praktisch sein, mit `groovyc` zu kompilieren, wenn Sie Groovy in Java-Projekte integrieren, die Ant (oder Maven) zur Build-Automatisierung verwenden. In Kapitel 14 erfahren Sie mehr darüber.

1.5 IDE- und Editor-Unterstützung für Groovy

Wenn Sie häufig in Groovy programmieren möchten, sollten Sie nach Groovy-Unterstützung für Ihre IDE oder den Editor Ihrer Wahl suchen. Manche Editoren unterstützen einstweilen nur Syntax-Highlighting für Groovy, doch selbst dies kann praktisch sein und die Arbeit mit Groovy bequemer machen. In den folgenden Abschnitten werden einige häufig benutzte IDEs und Text-Editoren für Groovy aufgeführt.

Dieser Abschnitt wird wahrscheinlich bei Drucklegung schon wieder überholt sein. Achten Sie auf Updates Ihrer bevorzugten IDE, da schon in naher Zukunft die wichtigsten Java-IDEs mit Groovy-Unterstützung ausgestattet werden. Sun Microsystems hat kürzlich Groovy-Unterstützung für sein NetBeans-Projekt *Coyote* angekündigt (*https://coyote.dev.java.net/*), was insofern interessant ist, als dies der erste IDE-Support für Groovy ist, der vom IDE-Hersteller selbst angeboten wird.

1.5.1 Das Plugin IntelliJ IDEA

Die Groovy-Community arbeitet zurzeit an einem Open-Source-Plugin namens GroovyJ. Mit diesem Plugin und den integrierten Funktionen von IDEA genießt ein Groovy-Programmierer folgende Vorteile:

- Einfaches Syntax-Highlighting nach Benutzervorgaben: GroovyJ nutzt zurzeit den Syntax-Highlighter von Java 5, der einen Großteil der Groovy-Syntax abdeckt. Die Version 1.0 wird die komplette Groovy-Syntax einbeziehen und über das COLORS & FONTS-Panel wie bei der Java-Syntax auch Anpassungen ermöglichen.
- Code-Vervollständigung: Bisher ist diese auf die Wortvervollständigung beschränkt, indem nur die Wortvervollständigung von IDEA mithilfe eines On-the-Fly-Wörterbuchs für den gerade benutzten Editor nutzbar gemacht wird.
- Enge Integration in die IDEA-Konfiguration und -Ausgabedarstellung von *compile*, *run*, *build* und *make*.
- Viele fortgeschrittene Editor-Aktionen, die wie in Java benutzt werden können.
- Effizienter Lookup aller dazugehörigen Java-Klassen im Projekt oder in abhängigen Bibliotheken.
- Effiziente Navigation zwischen den Dateien, einschließlich der *.groovy*-Dateien.
- Ein Icon für den Dateityp *Groovy*.

GroovyJ hat eine viel versprechende Zukunft, die stark von der IDEA-Implementierung des *Program Structure Interface (PSI)* für die Programmiersprache Groovy abhängt. Zu diesem Zweck wird IDEA die Grammatikdatei von Groovy spezialisieren und einen speziellen Parser generieren. Da IDEA alle fortgeschrittenen Merkmale (wie zum Beispiel Refactoring-

Unterstützung, Inspections, Navigation, Intentions usw.) auf dem PSI aufbaut, ist es wohl nur eine Frage der Zeit, bis diese Fähigkeiten auch in Groovy verfügbar sind.

GroovyJ ist ein interessantes Projekt unter der umsichtigen Leitung von Franck Rasolo. Dieses Plugin ist momentan eines der fortschrittlichsten, das für Groovy zur Verfügung steht. Mehr Information finden Sie unter *http://groovy.codehaus.org/GroovyJ+Status*.

1.5.2 Das Eclipse-Plugin

Das Groovy-Plugin für Eclipse erfordert Eclipse 3.1.1 oder höher. Es funktioniert auch mit von Eclipse 3.x abgeleiteten Tools wie dem Rational Application Developer und Rational Software Architect von IBM Rational. Zurzeit unterstützt das Groovy-Eclipse-Plugin die folgenden Fähigkeiten:

- Syntax-Highlighting für Groovy-Dateien
- Ein Groovy-Dateisymbol (Icon) für Groovy-Dateien in den Ansichten PACKAGE EXPLORER und RESOURCES
- Ausführen von Groovy-Skripten innerhalb der IDE
- Automatischer Build von Groovy-Dateien
- Debugger-Integration

Das Eclipse-Plugin für Groovy können Sie von *http://groovy.codehaus.org/Eclipse+Plugin* herunterladen.

1.5.3 Groovy-Unterstützung in anderen Editoren

Auch wenn sie nicht von sich behaupten, vollständige Entwicklungsumgebungen zu sein, bieten viele Editoren Unterstützung für Programmiersprachen im Allgemeinen und Groovy im Besonderen.

UltraEdit kann leicht angepasst werden, um Syntax-Highlighting für Groovy zu bieten und Skripte vom Editor aus zu starten oder zu kompilieren. Ausgaben werden an ein integriertes Ausgabefenster gesendet. Ein kleiner Kasten ermöglicht es, in der Datei zu Klassen- und Methodendeklarationen zu springen, und unterstützt intelligentes Einrücken und Klammernabgleich für Groovy. Abgesehen vom Groovy-Support ist er ein reichhaltiger, schnell startender Editor für alle Zwecke. Einzelheiten finden Sie unter *http://groovy.codehaus.org/UltraEdit+Plugin*.

Das *JEdit*-Plugin für Groovy unterstützt die Ausführung von Groovy-Skripten und Programmstücken gleich im Editor. Eine Konfiguration für Syntax-Highlighting ist separat erhältlich. Mehr darüber unter *http://groovy.codehaus.org/JEdit+Plugin*.

Konfigurationsdateien für das Syntax-Highlighting in TextPad, Emacs, Vim und mehreren anderen Editoren finden Sie auf der Groovy-Website unter *http://groovy.codehaus.org/Other+Plugins*.

Unser Tipp

Zum Programmieren kleiner Ad-hoc-Skripte mit Groovy verwende ich (Dierk) UltraEdit für Windows und Vim für Linux. Für größere Projekte benutze ich IntelliJ IDEA mit dem GroovyJ-Plugin.

In dem Maße, wie Groovy ausreift und von Java-Programmierern angenommen wird, wird weitere Unterstützung in Java-IDEs hinzukommen, mit Funktionen wie Debugging, Unit Tests und dynamischem Vervollständigen von Code.

1.6 Zusammenfassung

Wir hoffen, dass Sie nun überzeugt sind, dass Sie Groovy wirklich wollen. Als moderne Sprache, die auf dem soliden Fundament von Java aufbaut und von Sun unterstützt wird, hat Groovy fast jedem etwas zu bieten, egal in welcher Weise er mit der Java-Plattform arbeitet.

Mit einer klaren Vorstellung davon, weshalb Groovy entwickelt wurde und welche Ziele sein Design bestimmen, sollten Sie nun auch erkennen können, wo sich die Merkmale, die in den nächsten Kapiteln eingeführt werden, in das Gesamtbild einfügen. Denken Sie dabei immer an die Prinzipien Java-Integration und Reichhaltigkeit, die häufige Aufgaben vereinfachen und den Code ausdrucksstärker werden lassen.

Sobald Sie Groovy installiert haben, können Sie es sowohl direkt als Skript ausführen als auch in Klassen kompilieren. Wenn Sie vor Energie platzen, haben Sie vielleicht auch schon ein Groovy-Plugin für Ihre bevorzugte IDE installiert. Nach diesen Vorbereitungen sind Sie bereit, mehr von der Sprache selbst zu sehen (und auszuprobieren!). Im nächsten Kapitel nehmen wir Sie mit auf einen Kurztrip durch die Fähigkeiten von Groovy, damit Sie mehr Gefühl für den Zuschnitt dieser Sprache bekommen, ehe im Rest von Teil 1 die einzelnen Elemente genauer erklärt werden.