Alexandru Jecan

Die Modularität von Java 9

Projekt Jigsaw und skalierbare Java-Anwendungen

Springer Vieweg

Die Modularität von Java 9

Projekt Jigsaw und skalierbare Java-Anwendungen

Alexandru Jecan

Springer Vieweg

Die Modularität von Java 9: Projekt Jigsaw und skalierbare Java-Anwendungen

Alexandru Jecan
Munich, Deutschland

ISBN 978-3-662-68876-2 ISBN 978-3-662-68877-9 (eBook)
https://doi.org/10.1007/978-3-662-68877-9

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://portal.dnb.de abrufbar.

Dieses Buch ist eine Übersetzung des Originals in Englisch „Java 9 Modularity Revealed“ von Alexandru Jecan, publiziert durch APress Media, LLC im Jahr 2017. Die Übersetzung erfolgte mit Hilfe von künstlicher Intelligenz (maschinelle Übersetzung). Eine anschließende Überarbeitung im Satzbetrieb erfolgte vor allem in inhaltlicher Hinsicht, so dass sich das Buch stilistisch anders lesen wird als eine herkömmliche Übersetzung. Springer Nature arbeitet kontinuierlich an der Weiterentwicklung von Werkzeugen für die Produktion von Büchern und an den damit verbundenen Technologien zur Unterstützung der Autoren.

Planung/Lektorat: Axel Garbers
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft APress Media, LLC und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: 1 New York Plaza, New York, NY 10004, U.S.A.

An meine Frau, Diana, die mich jeden Tag in all meinen Bemühungen unterstützt und ermutigt. An meine Eltern, Alexandrina und Eugen, die mir seit meiner Kindheit eine sehr gute Ausbildung ermöglicht haben. Danke, ich liebe euch.

Inhaltsverzeichnis

Über den Autor

Alexandru Jecan ist ein leitender Software-Ingenieur, Berater, Autor, Trainer, Redner und Tech-Unternehmer, der derzeit in Berlin, Deutschland, lebt. Er hat einen Abschluss in Informatik von der Technischen Universität Cluj-Napoca, Rumänien, erworben.

Alexandru bietet professionelle Inhouse-Schulungen zu verschiedenen Softwaretechnologien in ganz Deutschland an. Seine Spezialgebiete sind: Big Data, Datenanalyse, künstliche Intelligenz, maschinelles Lernen, Back-End-Softwareentwicklung, Front-End-Softwareentwicklung, Datenbankentwicklung, Mikroservices und Devops.

Er spricht auf Technologiekonferenzen und Benutzergruppen, sowohl in Europa als auch in den Vereinigten Staaten, zu verschiedenen Themen im Zusammenhang mit Softwareentwicklung und Softwaretechnologien.

In seiner Freizeit liest Alexandru gerne viel und verbringt Zeit mit seiner Familie und seinen Töchtern Melissa und Mia. Alexandru ist ein begeisterter Leser, er liest viele Bücher und Zeitschriften in den Bereichen Informationstechnologie, Wirtschaft, Geschäft und Aktienmärkte. Sie können Alexandru auf Twitter unter @alexandrujecan folgen oder ihm eine E-Mail schicken an `alexandrujecan@gmail.com`.

Über den technischen Gutachter

Josh Juneau hat seit den frühen Tagen von Java EE Software und Unternehmensanwendungen entwickelt. Anwendungsentwicklung und Datenbankentwicklung waren seit Beginn seiner Karriere sein Schwerpunkt. Er wurde ein Oracle-Datenbankadministrator und übernahm die PL/SQL-Sprache für administrative Aufgaben und die Entwicklung von Anwendungen für die Oracle-Datenbank. In dem Bestreben, komplexere Lösungen zu erstellen, begann er, Java in seine PL/SQL-Anwendungen zu integrieren und entwickelte später eigenständige und Webanwendungen mit Java. Josh schrieb seine frühen Java-Webanwendungen unter Verwendung von JDBC und Servlets oder JSP, um mit Backend-Datenbanken zu arbeiten. Später begann er, Frameworks in seine Unternehmenslösungen zu integrieren, wie Java EE und JBoss Seam. Heute entwickelt er hauptsächlich Unternehmensweb-Lösungen unter Verwendung von Java EE und anderen Technologien. Er arbeitete auch mit alternativen Sprachen, wie Jython und Groovy, in einigen seiner Projekte.

Im Laufe der Jahre hat Josh mit vielen verschiedenen Programmiersprachen experimentiert, einschließlich alternativen Sprachen für die JVM. Im Jahr 2006 begann Josh, Zeit für das Jython-Projekt als Redakteur und Herausgeber des *Jython Monthly* Newsletters zu widmen. Ende 2008 startete er einen Podcast, der sich der Jython-Programmiersprache widmete. Josh war der Hauptautor von *The Definitive Guide to Jython,Oracle PL/SQL Recipes,* und *Java 7 Recipes,* und alleiniger Autor von *Java EE 7 Recipes* und *Einführung in Java EE 7* – alle veröffentlicht von Apress. Er arbeitet als Anwendungsentwickler und Systemanalyst am Fermi National Accelerator Laboratory und schreibt technische Artikel für Oracle und OTN. Er war Mitglied der Expertengruppen JSR 372 und JSR 378 und ist ein aktives Mitglied der Java-Community und hilft, die Bemühungen des Chicago Java User Group's Adopt-a-JSR zu leiten.

Wenn er nicht gerade programmiert oder schreibt, verbringt Josh gerne Zeit mit seiner wunderbaren Frau und seinen fünf Kindern, insbesondere beim Schwimmen, Angeln, Ballspielen und Filme schauen. Um mehr von Josh zu hören, folgen Sie seinem Blog unter http://jj-blogger.blogspot.com. Sie können ihm auch auf Twitter unter @ javajuneau folgen.

Danksagung

Ich möchte meiner Familie und meiner Frau, Diana, dafür danken, dass sie mich unterstützt, ermutigt und verstanden haben während der langen Nächte und Wochenenden, die ich damit verbracht habe, an diesem Buch zu schreiben.

Ich möchte meinen Eltern, Alexandrina und Eugen, dafür danken, dass sie mir seit meiner Kindheit eine sehr gute Ausbildung ermöglicht haben. Danke, dass ihr in meine Bildung investiert und mir seit meinen ersten Jahren Informatik- und Fremdsprachenkurse ermöglicht habt. Danke, dass ihr mich sehr gut erzogen habt.

Ich möchte dem gesamten Team von Apress für ihre sehr gute und professionelle Arbeit danken. Dank an meine koordinierende Redakteurin, Jill Balzano, und meinen Akquisitionsredakteur, Jonathan Gennick, für ihr Vertrauen in mich und ihre wertvollen Ratschläge und Unterstützung auf der schwierigen Reise des Schreibens dieses Buches. Ich danke Ihnen auch für Ihre Geduld und dafür, dass Sie mich während des gesamten Schreibprozesses ermutigt haben. Ich danke meinem technischen Gutachter, Josh Juneau, für die sehr hilfreichen und nützlichen Bewertungen. Vielen Dank, Apress-Team, für die hervorragende Arbeit!

Alexandru Jecan

Einführung

Die Programmiersprache Java, eingeführt im Jahr 1995, hat eine sehr erfolgreiche Geschichte. Sie hat sich seit ihrer Geburt ständig weiterentwickelt und ist eine der beliebtesten Programmiersprachen der Welt geworden. Jede neue Version von Java hat neue Funktionen hinzugefügt – klein, mittel und groß.

Java 9 ist endlich da! Es ist für die Veröffentlichung im September 2017 geplant, mehr als drei Jahre nach der Veröffentlichung von Java 8.

Probleme mit dem monolithischen JDK

Die Veröffentlichung von Java 8 im März 2014 brachte sehr wichtige Funktionen für die Java-Plattform wie Lambdas und die Stream-API, die von Entwicklern definitiv benötigt wurden. Dennoch wurden einige bekannte Schwächen der Plattform in Java 8 noch nicht angegangen: das riesige monolithische Java Development Kit (JDK) und der Klassenpfad. Diese Probleme werden in Java 9 durch Project Jigsaw angegangen.

Das wichtigste Merkmal von Java 9 ist mit Abstand das neue modulare System, das es eingeführt hat. Andere neue Funktionen wurden in Java 9 eingeführt, aber sie sind nicht der Schwerpunkt dieses Buches. Dieses Buch behandelt das neue modulare System, das in Java 9 eingeführt wurde. Das große, monolithische und unteilbare JDK war lange Zeit problematisch. Es ist schwierig, es auf kleinen Geräten zu installieren, weil viele dazu nicht genug Speicher haben. In vielen Fällen sind eine große Anzahl von Klassen, die das JDK bilden, nicht erforderlich, weil die Anwendung sie möglicherweise nicht benötigt. CORBA ist beispielsweise immer noch Teil des JDK, wird aber heute in realen Projekten kaum noch verwendet. Es macht keinen Sinn, das gesamte JDK zu verwenden, wenn nur ein Teil oder ein kleiner Teil davon benötigt wird. Die in Java 8 eingeführten Compact Profiles erkannten die Probleme, die durch das riesige JDK verursacht wurden, und versuchten, sie zu lösen, aber nur in geringem Maße. Die drei Compact Profiles enthalten immer noch viele Bibliotheken, die ein Entwickler tatsächlich nicht benötigen könnte. Es musste einen besseren Weg geben, das gesamte JDK aufzuteilen und ein viel kleineres

benutzerdefiniertes JDK als Laufzeitbild zu erstellen, das nur die benötigten Bibliotheken und nichts mehr enthält. Project Jigsaw ist dieser Weg, wie wir im Laufe dieses Buches sehen werden.

Große, monolithische Softwareanwendungen weisen eine Reihe von Nachteilen auf. Ihre Wartung ist schwierig und teuer, und eine kleine Änderung kann zu einem großen Aufwand führen. Bei großen Projekten ist Modularität entscheidend, da sie durch den von ihr bereitgestellten losen Kopplungsmechanismus eine einfache Wartung des Quellcodes ermöglicht, indem ihre Komplexität reduziert wird.

Probleme mit dem Klassenpfad

Die Probleme im Zusammenhang mit dem Klassenpfad existieren in Java seit der Geburt der Sprache. Die Java Virtual Machine (JVM) weiß nicht, dass ein Java Archive (JAR) auf dem Klassenpfad von einem anderen JAR abhängt. Sie lädt einfach eine Gruppe von JARs, überprüft aber nicht deren Abhängigkeiten. Fehlt eine JAR, bricht die JVM die Ausführung zur Laufzeit ab. Die JARs können keine Konzepte im Zusammenhang mit Zugänglichkeit definieren. Sie definieren keine Zugänglichkeitsbeschränkungen wie öffentlich oder privat. Der gesamte Inhalt aller JARs auf dem Klassenpfad ist vollständig sichtbar für alle anderen JARs auf dem Klassenpfad. Es gibt keine Möglichkeit zu erklären, dass einige Klassen in einer JAR privat sind. Alle Klassen und Methoden sind öffentlich im Zusammenhang mit dem Klassenpfad, was manchmal zu einem Problem führt, das als *JAR Hell* bezeichnet wird. Konflikte zwischen Versionen von JARs können entstehen, insbesondere wenn zwei verschiedene Versionen der gleichen Bibliothek benötigt werden. Das Laden der Klassen vom Klassenpfad ist ein langsamer Prozess, da die JVM nicht genau weiß, wo die Klasse sich befindet und daher alle vorhandenen Dateien vom Klassenpfad überprüfen muss. Jigsaw behebt diese Schwachstelle. Durch die Nutzung zuverlässiger Konfiguration werden Modulgrenzen durchgesetzt und die JVM kennt die benötigten Abhängigkeiten. Dies hat einen positiven Einfluss auf die Leistung. Java 9 definiert das Konzept des *Modulpfads* und ermöglicht es Ihnen, eine Bibliothek als JAR-Datei auf dem Klassenpfad zu haben oder die gleiche Bibliothek als Modul auf dem Modulpfad zu haben. Das bedeutet, niemand ist gezwungen, alle ihre Bibliotheken in Module umzuwandeln, wenn sie auf Java 9 umsteigen. Die Bibliotheken können weiterhin auf

dem Klassenpfad verwendet werden, auch in Java 9. Dies ist ein großer Vorteil, da Java 9 einen reibungslosen Übergang von Bibliotheken ermöglicht.

Der in Java 9 eingeführte Modulpfad tendiert dazu, die Probleme zu lösen, die durch den Klassenpfad verursacht werden. Er kann den Klassenpfad vollständig ersetzen oder kann interagieren und zusammen mit dem Klassenpfad arbeiten.

Modularität ist wichtig, weil sie eine wartbare Codebasis für die Zukunft bietet. Wir sollten modulare Programmierung verwenden, wenn wir den Aufwand für Design, Entwicklung und Testen trennen möchten. Modulare Programmierung beschleunigt die Entwicklung und erleichtert das Debuggen von Anwendungen, indem sie deren Komplexität reduziert.

Übersicht der Kapitel

Das erste Kapitel dieses Buches beschreibt die Konzepte, die das Fundament einer modularen Anwendung bilden: hohe Modulkohäsion, starke Kapselung, geringe Modulkopplung und explizite Schnittstellen. Es veranschaulicht auch einige der wichtigsten Prinzipien der modularen Programmierung wie Kontinuität, Verständlichkeit, Wiederverwendbarkeit, Kombinierbarkeit und Zerlegbarkeit.

Sie fragen sich vielleicht, warum wir nicht OSGi anstelle von Jigsaw verwenden sollten. Der Grund ist, dass OSGi nicht verwendet werden kann, um das JDK zu modularisieren, da es auf der Plattform Java Development Kit aufgebaut ist. Jigsaw ist nicht *auf der Plattform* aufgebaut, sondern direkt *im Kern davon.* Dies ermöglicht es Jigsaw, die Struktur des JDK vollständig zu verändern. Die Hauptunterschiede zwischen Jigsaw und OSGi werden in Kap. 2 beschrieben.

Vor JDK 9 gab es keine Möglichkeit, Module zu verwalten. Hier kommt Project Jigsaw ins Spiel. Es führt ein brandneues modulares System in das JDK ein und ermöglicht es daher, Anwendungen auf dem Gerüst einer modularen Architektur zu erstellen. Es bringt Flexibilität, Wartbarkeit, Sicherheit und Skalierbarkeit auf die Java-Plattform. Es führt lose gekoppelte Module ein, indem es die Abhängigkeiten zwischen den Modulen klar definiert.

Project Jigsaw gruppiert den Quellcode der Java-Plattform in Module und bietet ein neues System zur Implementierung von Modulen als Teil der Plattform. Es wendet das

modulare System auf die Plattform und auf das JDK selbst an und bietet Programmierern[1] die Möglichkeit, Programme mit dem modularen System auf dem JDK zu schreiben.

Das Hauptziel von Project Jigsaw besteht darin, das JDK und die Laufzeit zu modularisieren. Eine neue Komponente namens *Modul* wird eingeführt. Kap. 4 erklärt, was ein Modul ist und wie man ein Modul in Java 9 definiert. In diesem Kapitel wird auch untersucht, wie man die Abhängigkeiten eines Moduls deklariert, wie man es kapselt und wie man die interne Implementierung eines Moduls verbirgt. Sie werden lernen, was ein unbenanntes Modul, ein benanntes Modul, ein automatisches Modul und ein offenes Modul sind. Das Kapitel führt den Begriff des Modulpfads ein und zeigt, wie man einen Modulgraphen ohne Zyklen erstellt.

Bezüglich der Moduldeklarationen zeige ich praktische Beispiele unter Verwendung jeder der fünf in Jigsaw eingeführten Klauseln: die `requires`-Klausel, die `exports`-Klausel, die `uses`-Klausel, die `provides`-Klausel und die `opens`-Klausel.

Das Ziel des Java-Modulsystems besteht darin, *starke Kapselung* und *zuverlässige Konfiguration* bereitzustellen. Kap. 2 erklärt, was diese Begriffe bedeuten und wie sie durch das Modulsystem in Java 9 erreicht werden.

Der Klassenpfad wird teilweise durch Module und den Modulpfad ersetzt. Die klassenpfadspezifischen bekannten `ClassNotFound`-Ausnahmen werden auf dem Modulpfad vermieden, da der Compiler nun auf Basis der Moduldefinitionen testen kann, ob alle für den Betrieb der Anwendung benötigten Module verfügbar sind. Wenn sie nicht gefunden werden, kompiliert er die Anwendung nicht.

Zugänglichkeit ist ein wichtiger Teil des gesamten Ökosystems und wird in diesem Buch durchgehend behandelt. Das grundlegende Konzept, wie die Zugänglichkeit erreicht wird, ändert sich grundlegend in Java 9. Der öffentliche Zugriffsbezeichner, den wir alle kennen, bedeutet nicht mehr *überall und für jeden zugänglich.* Zusätzliche Zugänglichkeitsebenen wurden in JDK 9 hinzugefügt, die den bestehenden Zugänglichkeitsmechanismus erweitern, indem sie die Zugänglichkeit auf Modulebene definieren. Konzepte wie *direkte* und *implizite Lesbarkeit,* Voraussetzungen für die Definition von

[1] Anmerkung zur Übersetzung: Bei der Übersetzung von im Englischen nicht nach Geschlecht differenzierten Personenbezeichnungen wie „Programmierer“ u. Ä. wurde im Deutschen meistens die männliche Form verwendet, um den Text kürzer und besser lesbar zu machen. Selbstverständlich sind damit Personen jeden Geschlechts gemeint.

Zugänglichkeit, werden ebenfalls in diesem Buch skizziert. Sie werden sehen, wie die Zugänglichkeit durch den Compiler, durch die virtuelle Maschine oder durch die Verwendung von Core-Reflection durchgesetzt wird.

Wir werden uns das neue Konzept der *modularen JAR-Dateien* ansehen und den großen Vorteil, den es bringt: die Möglichkeit, eine Bibliothek mit JDK 9+ zu kompilieren, sie auf dem Modulpfad für JDK 9+ zu verwenden und sie mit JDK8 oder früher zu kompilieren und sie auf dem Klassenpfad zu verwenden. Da der Klassenpfad weiterhin verwendet werden kann, ist die Migration von Bibliotheken zu JDK 9 reibungslos. Selbst wenn die Bibliotheken einen Modulbeschreiber enthalten und als „Module“ behandelt werden sollen, funktionieren sie immer noch auf früheren Versionen von JDK 9, weil ihr Modulbeschreiber auf dem Klassenpfad nicht berücksichtigt wird. Durch die Verwendung von modularen JARs haben Entwickler die Freiheit zu entscheiden, ob sie zur Modulplattform wechseln möchten oder nicht.

Wir werden die Unterschiede zwischen regulären und modularen JARs anhand einiger Beispiele hervorheben und das neue Format für Dateien beschreiben, das in Java 9 eingeführt wurde, genannt JMOD, das dem Format von JAR-Dateien sehr ähnlich ist. Wir werden das neue JMOD-Tool durchgehen und seine Verwendung im Detail beschreiben.

Die Kompilierung mehrerer Module mit der Option `--module-source-path` wird anhand einiger erläuternder Beispiele veranschaulicht. Wir werden auch die Verbesserungen, die dem JAR-Tool hinzugefügt wurden, beschreiben und zeigen, wie man es verwendet, um alles als modulares JAR oder als JMOD-Datei zu verpacken.

Wir werden sehen, wie man die kompilierten Klassen und die `module-info.class` mit dem Java Launcher ausführt. Neue Optionen wie `--module-path` und `-m`, die in JDK9 eingeführt wurden, werden gründlich behandelt. Wenn man versucht, eine Anwendung mit der Option -m auszuführen, die dem Launcher mitteilt, wo das Hauptmodul ist, wird eine Auflösung ausgelöst. Wir werden alle Schritte, die bei der Ausführung einer Java-Modularanwendung beteiligt sind, detailliert beschreiben, einschließlich der Auslösung der Auflösung und der Erzeugung des Moduldiagramms. Wir werden uns auch spezielle Fälle ansehen, wie wenn ein Modul fehlt und der Start fehlschlägt, und wir werden einige Lösungen dafür vorstellen, wie die neu eingeführte Java-Launcher-Option `--show-module-resolution`.

Wir werden auch modulare JARs auf den Klassenpfad setzen und sehen, wie man sie erfolgreich ausführt. Das ist sehr wichtig: Wir werden erklären, wie man den Klassenpfad

und den Modulpfad beim Ausführen mit dem Java-Starter mischt. Dafür werden wir den neu eingeführten Befehlszeilenoption `--add-modules` nutzen.

Kap. 3 beschreibt das JEP 200, genannt das Modulare JDK. Dies ist das JEP, das das JDK in eine Reihe von Modulen aufteilt. Wir werden die neue Struktur des JDK zusammen mit seinen Modulen betrachten. Wir werden über Plattformmodule sprechen und den Modulgraphen zeigen, der die neue modulare Struktur des JDK darstellt. Wir werden den Graphen untersuchen, zeigen, wie die Module voneinander abhängen, und lernen, wie man alle Module aus dem System mit der Befehlszeilenoption `--list-modules` auflistet. Wir werden auch die Begriffe Standardmodul und Nicht-Standardmodul erklären.

Es liegt außerhalb des Rahmens dieses Buches, jedes Modul im Detail durchzugehen. Sie können eine umfassende Liste aller vorhandenen Module auf der Open-JDK-Website unter `http://cr.openjdk.java.net/~mr/jigsaw/ea/module-summary.html` finden.

Das JEP 260, Kapseln der meisten internen APIs, wird auch in diesem Buch behandelt. Um Inkompatibilitäten zu verwalten, wurden alle nicht kritischen internen APIs standardmäßig gekapselt. Darüber hinaus wurden alle kritischen internen APIs, für die unterstützter Ersatz in JDK 8 existiert, gekapselt. Andere kritische interne APIs wurden nicht gekapselt. Da sie in JDK 9 als veraltet gekennzeichnet wurden, wird ein Workaround über einen Befehlszeilenflag bereitgestellt.

Ein Linker und eine neue Phase namens *Link-Zeit* wurden in Java 9 eingeführt. Diese Phase ist optional und wird nach der Kompilierzeit, aber vor der Laufzeit ausgeführt. Sie setzt im Grunde die Module zusammen, um ein Laufzeitbild zu erstellen. Laufzeitbilder ermöglichen es uns, benutzerdefinierte JDKs zu erstellen, die nur die minimal notwendige Anzahl von Modulen enthalten, um unsere Anwendung auszuführen. Die kleinstmögliche Laufzeit würde ein einziges Modul, das Modul java.base, enthalten. Laufzeitbilder ermöglichen es uns, das JDK zu verkleinern oder es basierend auf unseren Bedürfnissen zu vergrößern. Sie sind ein Ersatz für das Laufzeit rt.jar.

Der Linker stellt eine neue Phase des Entwicklungslebenszyklus dar. Er verbessert die Leistung, indem er nur die minimalen Module auswählt, die er benötigt, um den Code erfolgreich zu kompilieren, und bietet viele Optimierungsoptionen für die Zukunft.

Jigsaw ermöglicht es, separate Module als Teil der JDK-Plattforminstallation zu installieren. Es erlaubt uns auch, dynamisch weitere zusätzliche Module in das JDK-Laufzeitbild einzufügen. Wir werden über die Änderungen in Bezug auf die binäre Struktur

der JRE und der JDK sowie über die neue Struktur des Legacy-JDK-Bildes sprechen. Sie werden mehr über all diese neuen Konzepte in den Kap. 5 und 7 erfahren.

Im Kap. 6 lernen Sie, was *Dienste* sind, und wir werden anhand von Beispielen die Begriffe Dienstschnittstelle und Dienstanbieter erläutern. Wir zeigen, wie man Dienstanbieter in Modulen definiert und wie man sie für andere Module verfügbar macht.

Im Kap. 8 sehen Sie, wie man Anwendungen und Bibliotheken auf eine reibungslose Weise zu Modulen migriert. Wir werden beschreiben, wie man eine Anwendung mit der Top-down-Migrationsstrategie auf Java 9 migriert. Dafür werden wir uns ein konkretes Beispiel ansehen, wie man eine Anwendung, die einige Drittanbieter-JARs enthält, zu Modulen migriert. Wir werden sehen, welche Art von Anwendungen das Risiko eines Ausfalls beim Wechsel zu JDK 9 darstellen. Wir werden nützliche Lösungen geben, um dies zu vermeiden, wie zum Beispiel das Durchsuchen des Codes nach Abhängigkeiten, das Vermeiden von Paketteilungen und das Überprüfen der Nutzung der internen APIs. Wenn Sie bereits auf JDK 9 umgestiegen sind, empfehlen wir, zuerst zu versuchen, Ihre Anwendungen mit dem neuen JDK auszuführen, um zu sehen, ob es Ihren Code bricht. Wir werden das JDeps-Tool und seine Verwendung zur Überprüfung Ihres Codes und zur Suche nach JDK-internen APIs behandeln. Wir werden das Plugin Maven JDeps diskutieren, das die Integration des JDeps-Tools mit dem Build-Tool Maven darstellt. Wir werden auch über die Auswirkungen und Konsequenzen der Entfernung von rt.jar und tools.jar aus der JRE sprechen.

Kap. 9 behandelt die neue API, die in JDK 9 für die Arbeit mit Modulen eingeführt wurde. Wir werden sehen, wie man grundlegende Operationen mit Modulen durchführt.

Kap. 10 geht auf einige fortgeschrittene Themen ein, zum Beispiel Schichten, den Klassenlademechanismus in JDK 9, Mehrversionen-JAR-Dateien, das JMOD-Tool und aufrüstbare Module. Wir werden das Konzept der Schichten beschreiben, die eine Gruppe von Klassenladern sind, die dazu dienen, Klassen aus einem Modulgraphen zu laden. Wir werden uns die Boot-Schicht ansehen und den Zusammenhang zu den sogenannten wohlgeformten Graphen.

Kap. 11 spricht darüber, wie man verschiedene Szenarien für das Testen modularer Anwendungen handhabt. Drei Szenarien werden behandelt: Junit-Testklassen und zu testende Objekte befinden sich in verschiedenen Modulen, Junit-Testklassen und zu testende Objekte befinden sich im selben Modul und nur zu testende Objekte befinden sich in einem Modul. Wir werden zeigen, wie man ein Modul patcht und wie man Maven zur Erleichterung des Testens verwendet.

Im Kap. 12 lernen Sie, wie Jigsaw mit Build-Tools wie Maven interagiert und welche Art von Unterstützung die beliebtesten IDEs für Project Jigsaw bieten.

Wie Sie sehen können, ist dieses Buch in 12 Kapitel gegliedert. Kap. 1 behandelt modulare Programmierkonzepte. Kap. 2–9 bieten Ihnen eine sehr starke Grundlage für Project Jigsaw. Kap. 10 beschreibt einige fortgeschrittene Funktionen, die Ihnen helfen, einige komplexe Themen zu Jigsaw zu verstehen. Kap. 11 zeigt, wie man modulare Anwendungen mit Junit testet. Kap. 12 lehrt den Einsatz von Jigsaw zusammen mit Build-Tools und integrierten Entwicklungsumgebungen (IDEs).

Jedes Thema sollte leicht zu finden sein. Wir empfehlen, die Kapitel der Reihe nach zu lesen, um alle Themen zu verstehen. Einige Beispiele bauen auf Konzepten auf, die in vorherigen Kapiteln erklärt wurden.

Wer sollte dieses Buch lesen

Dieses Buch richtet sich an jeden, der sich mit dem neuen Modularitätssystem vertraut machen möchte, das in Java 9 eingeführt wurde. Es bietet eine solide Grundlage für jeden, der die Kernkonzepte sowie die fortgeschrittenen Konzepte der Modularität von Java 9 verstehen möchte. Die Beispiele sind so konzipiert, dass sie Ihnen helfen, alle im Project Jigsaw eingeführten Begriffe tiefgehend zu verstehen. Wir haben versucht, so weit wie möglich, eine Vielzahl von Beispielen für die meisten der im gesamten Buch diskutierten theoretischen Konzepte zu geben.

Ob Sie bereits Erfahrung mit modularen Systemen haben oder nicht, dieses Buch ist für Sie.

Das Buch kann nicht *alles* über die Modularität von Java 9 abdecken. Die Java-9-Modularität ist ein sehr großes und komplexes Thema, und es wäre nicht möglich, jeden Aspekt davon in einem Buch dieser Größe zu behandeln. Das Buch geht jedoch durch alle Kernbereiche der Java-9-Modularität und berührt einige fortgeschrittene Themen. Durch das Lesen dieses Buches werden Sie nicht nur die Konzepte hinter der Java-9-Modularität verstehen, sondern auch in der Lage sein, sie auf Ihre täglichen Projekte anzuwenden.

Wir empfehlen Ihnen, die Beispiele aus diesem Buch selbst auszuprobieren, um sich mit Project Jigsaw vertraut zu machen.

Was Sie lernen werden

Dieses Buch zielt darauf ab, umfassende Informationen über das in Java 9 eingeführte neue modulare System zu liefern. Ein gut strukturiertes Tutorial wird im gesamten Buch durchgeführt, indem theoretische Konzepte mit praktischen Beispielen kombiniert werden.

Das Erlernen der Modularität mit Java 9 wird Ihnen helfen, Ihre Technologiekarriere zu verbessern und Ihnen sehr wertvolle technische Fähigkeiten zu verleihen.

Sobald Sie dieses Buch gelesen haben, werden Sie in der Lage sein, skalierbare und modulare Java-9-Anwendungen zu entwickeln und bestehende Java-Anwendungen auf Java 9 zu migrieren.

Hier sind einige der wichtigsten Dinge, die Sie in diesem Buch lernen werden:

- Was Modularität im Allgemeinen ist und welche Vorteile sie bringt
- Was ist die Modularität von Java 9 und was sind ihre Ziele
- Wie das neue Layout von JDK und JRE aussieht
- Was starke Kapselung und zuverlässige Konfiguration sind und wie man sie anwendet und davon profitiert
- Welche JDK-internen APIs in Java 9 gekapselt wurden und welche zugänglich geblieben sind
- Wie das JDK in eine Gruppe von Modulen unterteilt wurde
- Was die neuen Zugänglichkeitsregeln in JDK 9 sind
- Wie man ein Modul zusammen mit seinen Abhängigkeiten definiert
- Wie man eine modulare JAR-Datei und eine JMOD-Datei erstellt
- Wie man eine modulare Java-Anwendung mit Jigsaw kompiliert, packt und ausführt
- Wie man das JDeps-Tool zur Überprüfung des Codes, zur Suche nach Abhängigkeiten zwischen den Bibliotheken und zur Erstellung von Moduldeskriptoren nutzt
- Wie man eine Anwendung auf das modulare System migriert

- Wie man Migrationsprobleme wie gekapselte JDK-interne APIs, nicht aufgelöste Module, geteilte Pakete, zyklische Abhängigkeiten und mehr löst
- Wie man eine Top-down-Migration durchführt
- Wie man Dienste für Module definiert, konfiguriert und verwendet
- Wie man den Modulpfad und den Klassenpfad kombiniert, um Rückwärtskompatibilität zu gewährleisten
- Wie man benutzerdefinierte modulare Laufzeit-Images mit dem Jlink-Linking-Tool erstellt
- Wie man eine Schicht in Java 9 definiert und verwendet
- Wie man mit der neuen Modul-API Operationen mit einem Modul durchführt
- Wie man qualifizierte Exporte verwendet
- Wie man die Wartbarkeit und Leistung von Java-Anwendungen verbessern kann
- Wie man das Unit-Testing einer modularen Anwendung handhabt
- Wie man ein Modul patcht
- Wie man Jigsaw mit verschiedenen Build-Tools wie Maven integriert
- Wie man überprüft, ob eine Java-Anwendung mit JDK 9 kompatibel ist
- Wie man eine Java-Anwendung mit JDK 9 kompatibel macht
- Wie man die Laufzeitkompatibilität beim Wechsel zu Java 9 sicherstellt
- Wie man die besten Designmuster auswählt, wenn man eine Java-Anwendung modularisiert

Fehlerkorrekturen

Jeder, der an der Veröffentlichung dieses Buches beteiligt ist, ist stark daran interessiert, ein fehlerfreies Buch bereitzustellen. Deshalb wird das Erratum dieses Buches kontinuierlich aktualisiert, sobald auch nur ein kleines Problem gefunden wird. Sie können Errata unter www.apress.com/us/services/errata einreichen.

Den Autor kontaktieren

Der Quellcode für dieses Buch kann durch Klicken auf die Schaltfläche „Quellcode herunterladen“ auf seiner Produktseite bei apress.com abgerufen werden. Sie finden Sie unter www.apress.com/9781484227121.

Herunterladen des Codes

Der Quellcode aus diesem Buch kann auf GitHub gefunden werden. Sie können ihn auch von der Produktseite des Buches herunterladen unter www.apress.com/us/book/9781484227121.

KAPITEL 1

Konzepte der modularen Programmierung

Sie hatten fast sicher schon einmal mit Komplexität in Ihren Softwareprojekten zu tun. Das Komplexitätsniveau einer Softwareanwendung ist normalerweise niedrig, wenn die Entwicklung beginnt, aber nach einer Weile beginnt die Komplexität aufgrund der Änderungen, die an der Plattform vorgenommen wurden, zu steigen. Die Komplexität steigt ständig, da neue Funktionen hinzugefügt und die bestehende Funktionalität angepasst werden. Je mehr Änderungen und Anpassungen vorgenommen werden, desto komplexer wird das System – es kann so komplex werden, dass es für einen neuen Entwickler schwierig wird, das Projekt zu übernehmen und alle seine inneren Abläufe zu verstehen. Und wenn die Dokumentation der Systemsoftware nicht gut genug ist, dann wird das Verständnis des Systems noch schwieriger. Ein hohes Maß an Komplexität erfordert mehr Energie, Ressourcen und Zeit, um die innere Struktur der Anwendung zu verstehen.

Was führt dazu, dass Softwaresysteme so schwer zu warten sind? Die Antwort hängt damit zusammen, dass es viele bestehende Abhängigkeiten im gesamten Code gibt. Das passiert, wenn ein Codeabschnitt von vielen anderen Codeabschnitten abhängt, und das kann viele Probleme verursachen. Die Verbesserung eines solchen Systems wird schmerzhaft, weil eine Änderung an einer Stelle viele andere Teile der Anwendung beeinflussen kann. Durch die Modifikation einer Anwendung in vielen verschiedenen Bereichen steigt das Risiko, neue Fehler einzuführen. Darüber hinaus wird es sehr schwierig, ein zufriedenstellendes Maß an Wiederverwendung zu erreichen. Die Software hat so viele Abhängigkeiten, dass die einfache Wiederverwendung einer Komponente zeitlich kostspielig sein kann und die Komplexität sogar *weiter* erhöhen könnte. Dies behindert auch den Wunsch, das System zu verbessern. Wenn man ein System mit vielen Abhängigkeiten hat, wird die Hinzufügung neuer Funktionen zum

A. Jecan, *Die Modularität von Java 9,* https://doi.org/10.1007/978-3-662-68877-9_1

Albtraum. Darüber hinaus wird auch der Testprozess schwieriger, weil das Testen einzelner Komponenten fast unmöglich zu erreichen ist. Für Sie als Entwickler ist es aufgrund seiner Komplexität schwierig, jeden Teil des Systems zu verstehen. Da regelmäßig neue Funktionen hinzugefügt werden und das Softwaresystem sich weiterentwickelt, kann es eine Herausforderung sein, mit den Änderungen Schritt zu halten. Um die negativen Auswirkungen steigender Komplexität zu mildern und zu reduzieren, ist die Wartung des Systems obligatorisch, obwohl die Wartung selbst in Bezug auf Zeit, Aufwand und Kosten anspruchsvoll wird.

Was brauchen wir, um diese Probleme loszuwerden? Die Antwort ist Modularität.

Allgemeine Aspekte der Modularität

Modularität spezifiziert die Wechselbeziehung und Kommunikation zwischen den Teilen, die ein Softwaresystem bilden. *Modulare Programmierung* definiert ein Konzept namens Modul. *Module* sind Softwarekomponenten, die Daten und Funktionen enthalten. Integriert mit anderen Modulen, bilden sie zusammen ein einheitliches Softwaresystem. Modulare Programmierung bietet eine Technik zur Zerlegung eines gesamten Systems in unabhängige Softwaremodule. Modularität spielt eine entscheidende Rolle in der modernen Softwarearchitektur. Sie teilt ein großes Softwaresystem in separate Einheiten auf und hilft, die Komplexität von Softwareanwendungen zu reduzieren und gleichzeitig den Entwicklungsaufwand zu verringern.

Das Ziel der Modularität ist es, neue Einheiten zu definieren, die leicht zu verstehen und zu verwenden sind. Modulare Programmierung ist ein Stil zur Entwicklung von Softwareanwendungen, indem die Funktionalität in verschiedene Module aufgeteilt wird – Softwareeinheiten, die Geschäftslogik enthalten und die Aufgabe haben, eine spezifische Funktionalität zu implementieren. Modularität ermöglicht eine klare Trennung der Anliegen und gewährleistet Spezialisierung. Sie verbirgt auch die Implementierungsdetails des Moduls. Modularität ist ein wichtiger Teil der agilen Softwareentwicklung, weil sie es uns ermöglicht, Module zu ändern oder zu refaktorisieren, ohne andere Module zu brechen.

Zwei der wichtigsten Aspekte der Modularität sind Wartbarkeit und Wiederverwendbarkeit, die beide große Vorteile bringen.