Java Challenge

Michael Inden

Fit für das Jobinterview und die Praxis –
mit mehr als 100 Aufgaben
und Musterlösungen

dpunkt.verlag

Dipl.-Inform. Michael Inden ist Oracle-zertifizierter Java-Entwickler. Nach seinem Studium in Oldenburg hat er bei diversen internationalen Firmen in verschiedenen Rollen etwa als Softwareentwickler, -architekt, Consultant, Teamleiter sowie Trainer gearbeitet. Zurzeit ist er als CTO und Leiter Academy in Zürich tätig.

Michael Inden hat über zwanzig Jahre Berufserfahrung beim Entwurf komplexer Softwaresysteme gesammelt, an diversen Fortbildungen und mehreren Java-One-Konferenzen teilgenommen. Sein besonderes Interesse gilt dem Design qualitativ hochwertiger Applikationen mit ergonomischen GUIs sowie dem Coaching. Sein Wissen gibt er gerne als Trainer in internen und externen Schulungen und auf Konferenzen weiter, etwa bei der Java User Group Switzerland, bei der JAX/W-JAX, ch.open und den IT-Tagen.

Michael Inden

Java Challenge

Fit für das Job-Interview und die Praxis – mit mehr als 100 Aufgaben und Musterlösungen

dpunkt.verlag

Michael Inden
michael_inden@hotmail.com

Lektorat: Michael Barabas
Copy-Editing: Ursula Zimpfer, Herrenberg
Satz: Michael Inden
Herstellung: Stefanie Weidner
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-86490-756-2
PDF 978-3-96910-028-8
ePub 978-3-96910-029-5
mobi 978-3-96910-030-1

1. Auflage 2020

Wieblinger Weg 17
69123 Heidelberg

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: hallo@dpunkt.de.

5 4 3 2 1 0

Inhaltsverzeichnis

Vorwort

Zunächst einmal bedanke ich mich bei Ihnen, dass Sie sich für dieses Buch entschieden haben. Hierin finden Sie eine Vielzahl an Übungsaufgaben zu den unterschiedlichsten Themengebieten, die kurzweilige Unterhaltung durch Lösen und Implementieren der Aufgaben bieten und Sie so entweder auf Bewerbungsgespräche einstimmen oder aber einfach Ihre Problemlösungsfähigkeiten verbessern.

Übung macht den Meister

Wir alle kennen das Sprichwort: »Übung macht den Meister.« Im Handwerk und in diversen Bereichen des realen Lebens wird viel geübt und der Ernstfall ist eher selten, etwa im Sport, bei Musikern und in anderen Bereichen. Merkwürdigerweise ist dies bei uns Softwareentwicklern oftmals deutlich anders. Wir entwickeln eigentlich fast die gesamte Zeit und widmen uns dem Üben und Lernen bzw. Einstudieren eher selten, teilweise gar nicht. Wie kommt das?

Vermutlich liegt das neben dem in der Regel vorherrschenden Zeitdruck auch daran, dass nicht so viel geeignetes Übungsmaterial zur Verfügung steht – es gibt zwar Lehrbücher zu Algorithmen sowie Bücher zu Coding, aber meistens sind diese entweder zu theoretisch oder zu Sourcecode-lastig und beinhalten zu wenig Erklärungen der Lösungswege. Das will dieses Buch ändern.

Wieso dieses Buch?

Wie kam ich dazu, dieses Buchprojekt in Angriff zu nehmen? Das hat mehrere Gründe. Zum einen wurde ich immer wieder per Mail oder persönlich von Teilnehmern meiner Workshops gefragt, ob es nicht ein Übungsbuch als Ergänzung zu meinem Buch »Der Weg zum Java-Profi« [2] geben würde. Dadurch kam die erste Idee auf.

Wirklich ausgelöst hat das Ganze, dass ein Recruiter von Google, mit einer Jobanfrage recht überraschend auf mich zukam. Als Vorbereitung für die dann bevorstehenden Jobinterviews und zur Auffrischung meiner Kenntnisse machte ich mich auf die Suche nach geeigneter Lektüre und entwickelte selbst schon einige Übungsaufgaben. Dabei entdeckte ich das großartige, aber auch teilweise recht anspruchsvolle Buch »Cracking the coding interview« von Gayle Laakmann McDowell [5], das mich weiter inspirierte, sodass ich ein paar der dort vorgestellten Ideen aufgreife.

An wen richtet sich dieses Buch?

Dieses Buch ist kein Buch für Programmierneulinge, sondern richtet sich an Leser, die bereits recht gutes Java-Know-how besitzen und dieses mithilfe von Übungen weiter vertiefen wollen. Anhand kleiner Programmieraufgaben erweitern Sie auf unterhaltsame Weise Ihr Wissen rund um Java, Algorithmen und gutes OO-Design.

Dieses Buch richtet sich im Speziellen an zwei Zielgruppen:

1. Zum einen sind dies engagierte Hobbyprogrammierer und Informatikstudierende, aber auch Berufseinsteiger, die Java als Sprache schon ganz gut beherrschen, und nun ihr Wissen anhand von Übungen vertiefen wollen.
2. Zum anderen ist das Buch für erfahrene Softwareentwickler und -architekten bestimmt, die ihr Wissen ergänzen oder auffrischen wollen, um einige althergebrachte Denkmuster aufzubrechen und neue Ideen zu entwickeln. Insbesondere zur Lösungsfindung und zu Algorithmen und Datenstrukturen sollte es das eine oder andere Aha-Erlebnis geben.

Generell verwende ich die maskuline Form, um den Text leichter lesbar zu halten. Natürlich beziehe ich damit alle weiblichen Leserinnen mit ein und freue mich über diese ganz besonders.

Was vermittelt dieses Buch?

Dieses Buch enthält einen bunten Mix an Übungsaufgaben zu verschiedenen Themengebieten. Mitunter gibt es auch einige Knobelaufgaben, die zwar nicht direkt für die Praxis wichtig sind, aber indirekt doch, weil Sie Ihre Fähigkeiten zur Kreativität und zur Lösungsfindung verbessern.

Neben Übungsaufgaben und dokumentierten Lösungen war es mir wichtig, dass jeder im Buch behandelte Themenbereich mit einer kurzen Einführung startet, damit auch diejenigen Leser abgeholt werden, die in einigen Gebieten vielleicht noch nicht so viel Know-how aufgebaut haben. Damit können Sie sich dann an die Aufgaben bis etwa zum mittleren Schwierigkeitsgrad wagen. In jedem Themengebiet finden sich immer auch einige leichtere Aufgaben zum Einstieg. Mit etwas Übung sollten Sie sich auch an etwas schwierigere Probleme wagen. Mitunter gibt es herausfordernde Knacknüsse, an denen sich dann Experten versuchen können oder solche, die es werden wollen.

Tipps und Hinweise aus der Praxis

Dieses Buch ist mit diversen Praxistipps gespickt. In diesen werden interessante Hintergrundinformationen präsentiert oder es wird auf Fallstricke hingewiesen.

Tipp: Praxistipp

In derart formatierten Kästen finden sich im späteren Verlauf des Buchs immer wieder einige wissenswerte Tipps und ergänzende Hinweise zum eigentlichen Text.

Schwierigkeitsgrad im Überblick

Für ein ausgewogenes, ansprechendes Übungsbuch bedarf es selbstverständlich einer Vielzahl an Aufgaben verschiedener Schwierigkeitsstufen, die Ihnen als Leser die Möglichkeit bieten, sich schrittweise zu steigern und Ihre Kenntnisse auszubauen. Dabei setze ich zwar ein gutes Java-Grundwissen voraus, allerdings erfordern die Lösungen niemals ganz tiefes Wissen über ein Themengebiet oder ganz besondere Sprachfeatures.

Damit der Schwierigkeitsgrad einfach und direkt ersichtlich ist, habe ich die von anderen Bereichen bekannte Sternekategorisierung genutzt, deren Bedeutung in diesem Kontext in nachfolgender Tabelle etwas genauer erläutert wird.

Sterne (Bedeutung)	Einschätzung	Zeitaufwand
★☆☆☆☆ (sehr leicht)	Die Aufgaben sollten ohne große Vorkenntnisse mit einfachem Java-Wissen in wenigen Minuten lösbar sein.	< 15 min
★★☆☆☆ (leicht)	Die Aufgaben erfordern ein wenig Nachdenken, sind aber dann direkt zu lösen.	< 30 min
★★★☆☆ (mittel)	Die Aufgaben sind mit etwas Nachdenken, ein wenig Strategie und manchmal durch die Betrachtung verschiedener Rahmenbedingungen gut schaffbar.	~ 30 – 45 min
★★★★☆ (schwierig)	Erprobte Problemlösungsstrategien, gutes Wissen zu Datenstrukturen und fundierte Java-Kenntnisse sind zur Lösung nötig.	~ 45 – 90 min
★★★★★ (sehr schwierig)	Die Aufgaben sind wirklich knifflig und schwierig zu lösen. Das sind erst dann Kandidaten, nachdem die anderen Aufgaben Ihnen keine größeren Schwierigkeiten mehr bereiten.	> 60 min

Dies sind jeweils nur Einschätzungen von meiner Seite und eher grobe Einordnungen. Bedenken Sie bitte, dass die von jedem Einzelnen wahrgenommene Schwierigkeit auch sehr von seinem Background und Wissensstand abhängt. Ich habe schon erlebt, dass sich Kollegen mit Aufgaben schwergetan haben, die ich als recht einfach empfand. Aber auch das Gegenteil kenne ich: Während andere eine Aufgabe anscheinend spielend lösen, ist man selbst am Verzweifeln, weil der Groschen einfach nicht fällt. Manchmal hilft dann eine Pause mit einem Kaffee oder ein kleiner Spaziergang. Lassen Sie sich auf keinen Fall demotivieren – jeder hat irgendwann mit irgendeiner Art von Aufgabe zu kämpfen.

Hinweis: Mögliche Alternativen zu den Musterlösungen

Beachten Sie bitte, dass es für Problemstellungen nahezu immer einige Varianten gibt, die für Sie vielleicht sogar eingängiger sind. Deswegen werde ich ab und an als Denkanstoß interessante Alternativen zur (Muster-)Lösung präsentieren.

Aufbau dieses Buchs

Nachdem Sie eine grobe Vorstellung über den Inhalt dieses Buchs haben, möchte ich die Themen der einzelnen Kapitel kurz vorstellen.

Wie bereits angedeutet, sind die Übungsaufgaben thematisch gruppiert. Dabei bilden die sechs Kapitel nach der Einleitung die Grundlage und die darauffolgenden drei Kapitel behandeln fortgeschrittenere Themengebiete.

Kapitel 1 – Einleitung Dieses Kapitel beschreibt den grundsätzlichen Aufbau der folgenden Kapitel mit den Abschnitten Einführung, Aufgaben und Lösungen. Zudem wird ein Grundgerüst für die oftmals zur Prüfung der Lösungen genutzten Unit Tests vorgestellt. Abschließend gebe ich Hinweise zum Ausprobieren der Beispiele und Lösungen.

Kapitel 2 – Mathematische Aufgaben Das zweite Kapitel widmet sich mathematischen Operationen sowie Aufgaben zu Primzahlen und dem römischen Zahlensystem. Darüber hinaus präsentiere ich ein paar Ideen zu Zahlenspielereien.

Kapitel 3 – Rekursion Rekursion ist ein wichtiger Basisbaustein bei der Formulierung von Algorithmen. Dieses Kapitel gibt einen kurzen Einstieg und die diversen Übungsaufgaben sollten dabei helfen, Rekursion zu verstehen.

Kapitel 4 – Strings Strings sind bekanntermaßen Zeichenketten, die eine Vielzahl an Methoden bieten. Ein solides Verständnis ist elementar wichtig, da nahezu kein Programm ohne Strings auskommt. Deswegen werden wir in diesem Kapitel die Verarbeitung von Zeichenketten anhand verschiedener Übungsaufgaben kennenlernen.

Kapitel 5 – Arrays Neben Strings sind Arrays ebenfalls Grundbausteine beim Programmieren. Arrays sind – wie Sie wissen – einfache Datenstrukturen zur Speicherung von Werten, allerdings ohne allzu viel Komfort. In der Praxis empfiehlt sich deswegen oftmals, auf die Datenstrukturen des Collections-Frameworks zurückzugreifen. Diese behandelt dann Kapitel 7 eingehend.

Kapitel 6 – Datumsverarbeitung Mit Java 8 wurde das JDK um einige Funktionalitäten zur Datumsverarbeitung erweitert. Diese sollte jeder Java-Entwickler kennen. Die Übungsaufgaben verschaffen Ihnen einen guten Einstieg in die Thematik, sodass der Transfer in die Praxis leichtfallen sollte.

Kapitel 7 – Basisdatenstrukturen: Listen, Sets und Maps Im Collections-Framework werden Listen, Mengen und Schlüssel-Wert-Abbildungen durch verschiedene Containerklassen realisiert. Für den Programmieralltag ist ein sicherer Einsatz von großem Vorteil, was durch die Übungsaufgaben trainiert wird.

Kapitel 8 – Rekursion Advanced Kapitel 3 hat das Thema Rekursion einleitend behandelt. In diesem Kapitel beschäftigen wir uns mit einigen fortgeschritteneren Aspekten rund um das Thema Rekursion. Wir starten mit der Optimierungstechnik namens Memoization. Im Anschluss schauen wir uns Backtracking als eine Problemlösungsstrategie an, die auf Versuch-und-Irrtum beruht und mögliche Lösungswege durchprobiert. Damit lassen sich diverse Algorithmen ziemlich verständlich und elegant halten.

Kapitel 9 – Bäume Baumstrukturen spielen in der Informatik sowohl in der Theorie als auch der Praxis eine wichtige Rolle. In vielen Anwendungskontexten lassen sich Bäume gewinnbringend einsetzen, etwa für die Verwaltung eines Dateisystems, die Darstellung eines Projekts mit Teilprojekten und Aufgabenpaketen oder eines Buchs mit Kapiteln, Unterkapiteln und Abschnitten.

Kapitel 10 – Suchen und Sortieren Suchen und Sortieren sind zwei elementare Themen der Informatik im Bereich der Algorithmen und Datenstrukturen. Das Collections-Framework setzt beide um und nimmt einem dadurch viel Arbeit ab. Jedoch lohnt sich auch ein Blick hinter die Kulissen, etwa auf verschiedene Sortierverfahren und deren spezifische Stärken und Schwächen.

Kapitel 11 – Schlusswort und ergänzende Literatur In diesem Kapitel fasse ich das Buch zusammen und gebe vor allem einen Ausblick auf ergänzende Literatur. Um Ihr Können zu erweitern, ist neben dem Programmiertraining auch das Studium von weiteren Büchern empfehlenswert. Eine Auswahl an hilfreichen Titeln bildet den Abschluss des Hauptteils dieses Buchs.

Anhang A – Schnelleinstieg JShell In diesem Buch werden diverse Beispiele direkt auf der Konsole ausprobiert. Der Grund ist vor allem, dass Java seit Version 9 die interaktive Kommandozeilenapplikation JShell als REPL (Read Evaluate Print Loop) bietet, die wir in diesem Anhang kurz kennenlernen wollen.

Anhang B – Schnelleinsteg JUnit Zum Prüfen kleinerer Programmbausteine haben sich Unit Tests bewährt. Mit JUnit 5 ist das Ganze insbesondere beim Formulieren von Testfällen für mehrere Eingabekombinationen ziemlich komfortabel. Weil viele der hier im Buch erstellten Lösungen mit Unit Tests geprüft werden, gibt dieser Anhang einen Einstieg in die Thematik.

Anhang C – Schnelleinstieg O-Notation In diesem Buch verwende ich manchmal zur Abschätzung des Laufzeitverhaltens und zur Einordnung der Komplexität von Algorithmen die sogenannte O-Notation. Dieser Anhang stellt Wesentliches dazu vor.

Konventionen und ausführbare Programme

Verwendete Zeichensätze

Im gesamten Text gelten folgende Konventionen bezüglich der Schriftart: Der normale Text erscheint in der vorliegenden Schriftart. Dabei werden wichtige Textpassagen *kursiv* oder ***kursiv und fett*** markiert. Englische Fachbegriffe werden eingedeutscht groß geschrieben. Zusammensetzungen aus englischen und deutschen (oder eingedeutschten) Begriffen werden mit Bindestrich verbunden, z. B. Plugin-Manager. Sourcecode-Listings sind in der Schrift `courier` gesetzt, um zu verdeutlichen, dass dieser Text einen Ausschnitt aus einem Java-Programm wiedergibt. Auch im normalen Text werden Klassen, Methoden, Konstanten und Übergabeparameter in dieser Schriftart dargestellt.

Verwendete Abkürzungen

Im Buch verwende ich die in der nachfolgenden Tabelle aufgelisteten Abkürzungen. Weitere Abkürzungen werden im laufenden Text in Klammern nach ihrer ersten Definition aufgeführt und anschließend bei Bedarf genutzt.

Abkürzung	Bedeutung
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
(G)UI	(Graphical) User Interface
IDE	Integrated Development Environment
JDK	Java Development Kit
JLS	Java Language Specification
JRE	Java Runtime Environment
JSR	Java Specification Request
JVM	Java Virtual Machine

Verwendete Java-Version(en)

Nahezu alle Programme nutzen Java 11 als Basis, da dies die letzte LTS-Version (Long Term Support) ist, die mittlerweile eine gute Verbreitung besitzt. Die neueren Java-Versionen bringen einige hilfreiche Syntaxänderungen und API-Erweiterungen, die jedoch zur Lösung der Aufgaben nicht von Bedeutung sind – außer für ganz spezielle Aufgaben zu `switch` mit Java 14.

Verwendete Klassen aus dem JDK

Werden Klassen des JDKs zum ersten Mal im Text erwähnt, so wird deren voll qualifizierter Name, d. h. inklusive der Package-Struktur, angegeben: Für die Klasse `String` würde dann etwa `java.lang.String` notiert. Dies erleichtert ein Auffinden im JDK. Im darauffolgenden Text wird zur besseren Lesbarkeit auf diese Angabe verzichtet und nur der Klassenname genannt. Zudem sind aus Platzgründen in den Listings nur selten `import`-Anweisungen abgebildet.

Im Text beschriebene Methodenaufrufe enthalten in der Regel die Typen der Übergabeparameter, etwa `substring(int, int)`. Sind die Parameter in einem Kontext nicht entscheidend, wird auf deren Angabe aus Gründen der besseren Lesbarkeit verzichtet oder aber durch die Zeichenfolge `...` abgekürzt.

Download, Sourcecode und ausführbare Programme

Der Sourcecode der Beispiele steht auf der Webseite

`www.dpunkt.de/java-challenge`

zum Download bereit und ist in ein Eclipse-Projekt integriert. Weil dies ein Buch zum Mitmachen ist, sind einige der Programme mithilfe von Gradle-Tasks ausführbar. Deren Name wird in Kapitälchenschrift, etwa LOCALEEXAMPLE, angegeben – alternativ ist natürlich eine Ausführung in der IDE bzw. als Unit Test möglich.

Viele Codeschnipsel lassen sich aber auch hervorragend in der JShell ausprobieren. Um das zu gewährleisten, sind mitunter bereits entwickelte Methoden an geeigneter Stelle nochmals abgebildet.

Nacharbeiten nach Projekt-Import Nach dem erstmaligen Import müssen die Abhängigkeiten auf die externen Bibliotheken im Eclipse-Projekt mit dem Kommando `gradle cleanEclipse eclipse` neu initialisiert und auf Ihren Rechner aktualisiert werden.

Danksagung

Ein Fachbuch zu schreiben ist eine schöne, aber arbeitsreiche und langwierige Aufgabe. Alleine kann man dies kaum bewältigen. Daher möchte ich mich an dieser Stelle bei allen bedanken, die direkt oder indirekt zum Gelingen des Buchs beigetragen haben. Insbesondere konnte ich bei der Erstellung des Manuskripts auf ein starkes Team an Korrekturlesern zurückgreifen. Es ist hilfreich, von den unterschiedlichen Sichtweisen und Erfahrungen profitieren zu dürfen.

Zunächst einmal möchte ich mich bei Michael Kulla, der als Trainer für Java SE und Java EE bekannt ist, für sein mehrmaliges, gründliches Review vieler Kapitel, die fundierten Anmerkungen und den tollen Einsatz ganz herzlich bedanken. Ebenfalls bin ich Prof. Dr. Dominik Gruntz für eine Vielzahl an Verbesserungsvorschlägen sehr dankbar. Zudem erhielt ich die eine oder andere hilfreiche Anregung von Jean-Claude Brantschen, Prof. Dr. Carsten Kern und Christian Heitzmann. Wieder einmal hat auch Ralph Willenborg ganz genau gelesen und so diverse Tippfehler gefunden. Vielen Dank dafür!

Ebenso geht ein Dankeschön an das Team des dpunkt.verlags (Dr. Michael Barabas, Martin Wohlrab, Anja Weimer und Birgit Bäuerlein) für die tolle Zusammenarbeit. Außerdem möchte ich mich bei Torsten Horn für die fundierte fachliche Durchsicht sowie bei Ursula Zimpfer für ihre Adleraugen beim Copy-Editing bedanken.

Abschließend geht ein lieber Dank an meine Frau Lilija für ihr Verständnis und die Unterstützung, vor allem auch für etliche Stupser, um auf das Fahrrad zu steigen und eine Runde zu drehen, anstatt nur am Buch zu arbeiten.

Anregungen und Kritik

Trotz großer Sorgfalt und mehrfachen Korrekturlesens lassen sich missverständliche Formulierungen oder sogar Fehler leider nicht vollständig ausschließen. Falls Ihnen etwas Derartiges auffallen sollte, so zögern Sie bitte nicht, mir dies mitzuteilen. Gerne nehme ich auch Anregungen oder Verbesserungsvorschläge entgegen. Kontaktieren Sie mich bitte per Mail unter:

`michael_inden@hotmail.com`

Zürich, im Juni 2020
Michael Inden

1 Einleitung

Herzlich willkommen zu diesem Übungsbuch! Bevor Sie loslegen, möchte ich kurz darstellen, was Sie bei der Lektüre erwartet.

Dieses Buch behandelt verschiedene praxisrelevante Themengebiete und deckt diese durch Übungsaufgaben unterschiedlicher Schwierigkeitsstufen ab. Die Übungsaufgaben sind (größtenteils) voneinander unabhängig und können je nach Lust und Laune oder Interesse in beliebiger Reihenfolge gelöst werden.

Neben den Aufgaben finden sich die jeweiligen Lösungen inklusive einer kurzen Beschreibung des zur Lösung verwendeten Algorithmus sowie dem eigentlichen, an wesentlichen Stellen kommentierten Sourcecode.

1.1 Aufbau der Kapitel

Jedes Kapitel ist strukturell gleich aufgebaut, sodass Sie sich schnell zurechtfinden werden.

Einführung

Ein Kapitel beginnt jeweils mit einer Einführung in die jeweilige Thematik, um auch diejenigen Leser abzuholen, die mit dem Themengebiet vielleicht noch nicht so vertraut sind, oder aber, um Sie auf die nachfolgenden Aufgaben entsprechend einzustimmen.

Aufgaben

Danach schließt sich ein Block mit Übungsaufgaben und folgender Struktur an:

Aufgabenstellung Jede einzelne Übungsaufgabe besitzt zunächst eine Aufgabenstellung. Dort werden in wenigen Sätzen die zu realisierenden Funktionalitäten beschrieben. Oftmals wird auch schon eine mögliche Methodensignatur als Anhaltspunkt zur Lösung angegeben.

Beispiele Ergänzend finden sich fast immer Beispiele zur Verdeutlichung mit Eingaben und erwarteten Ergebnissen. Nur für einige recht einfache Aufgaben, die vor allem zum Kennenlernen eines APIs dienen, wird mitunter auf Beispiele verzichtet.

Oftmals werden in einer Tabelle verschiedene Wertebelegungen von Eingabeparameter(n) sowie das erwartete Ergebnis dargestellt, etwa wie folgt:

Eingabe A	Eingabe B	Ergebnis
[1, 2, 4, 7, 8]	[2, 3, 7, 9]	[2, 7]

Für die Angaben gelten folgende Notationsformen:

- "AB" – steht für textuelle Angaben
- true / false – repräsentieren boolesche Werte
- 123 – Zahlenangaben
- [value1, value2, ...] – steht für Collections wie Sets oder Listen, aber auch Arrays
- { key1 : value1, key2 : value3, ... } – beschreibt Maps

Lösungen

Auch der Teil der Lösungen besitzt die nachfolgend beschriebene Struktur.

Aufgabenstellung und Beispiele Zunächst finden wir nochmals die Aufgabenstellung, sodass wir nicht ständig zwischen Aufgaben und Lösungen hin- und herblättern müssen, sondern das Ganze in sich abgeschlossen ist.

Algorithmus Danach folgt eine Beschreibung des gewählten Algorithmus zur Lösung. Aus Gründen der Didaktik zeige ich bewusst auch einmal einen Irrweg oder eine nicht so optimale Lösung, um daran dann Fallstricke aufzudecken und iterativ zu einer Verbesserung zu kommen. Tatsächlich ist die eine oder andere Brute-Force-Lösung manchmal sogar schon brauchbar, bietet aber Optimierungspotenziale. Exemplarisch werde ich immer wieder entsprechende, mitunter verblüffend einfache, aber oft auch sehr wirksame Verbesserungen vorstellen.

Prüfung Teilweise sind die Aufgaben recht leicht oder dienen nur dem Kennenlernen von Syntax oder API-Funktionalität. Dafür scheint es mir oftmals ausreichend, ein paar Aufrufe direkt in der JShell auszuführen. Deshalb verzichte ich hierfür auf Unit Tests. Gleiches gilt auch, wenn wir bevorzugt eine grafische Aufbereitung einer Lösung, etwa die Darstellung eines Sudoku-Spielfelds zur Kontrolle nutzen und der korrespondierende Unit Test vermutlich schwieriger verständlich wäre.

Je komplizierter allerdings die Algorithmen werden, desto mehr lauern auch Fehlerquellen, wie falsche Indexwerte, eine versehentliche oder unterbliebene Negation oder ein übersehener Randfall. Deswegen bietet es sich an, Funktionalitäten mithilfe von Unit Tests zu überprüfen – in diesem Buch kann das aus Platzgründen natürlich nur exemplarisch für wichtige Eingaben geschehen. Insgesamt existieren jedoch über 90 Unit Tests mit rund 750 Testfällen. Ein ziemlich guter Anfang. Trotzdem sollte in der Praxis das Netz an Unit Tests und Testfällen wenn möglich noch umfangreicher sein.

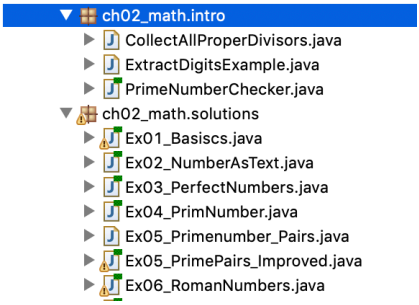
1.2 Grundgerüst des Eclipse-Projekts

Auch das mitgelieferte Eclipse-Projekt orientiert sich in seinem Aufbau an demjenigen des Buchs und bietet für die Kapitel mit Übungsaufgaben jeweils ein eigenes Package pro Kapitel, z. B. `ch02_math` oder `ch08_recursion_advanced`. Dabei weiche ich ausnahmsweise von der Namenskonvention für Packages ab, weil ich die Unterstriche in diesem Fall für eine lesbare Notation halte.

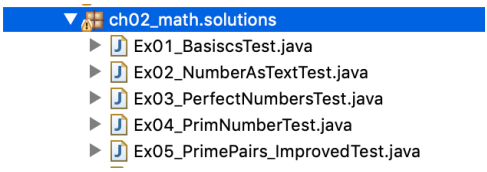
Einige der Sourcecode-Schnipsel aus den jeweiligen Einführungen finden sich in einem Subpackage `intro`. Die bereitgestellten (Muster-)Lösungen werden in jeweils eigenen Subpackages namens `solutions` gesammelt und die Klassen sind gemäß Aufgabenstellung wie folgt benannt: `Ex<Nr>_<Aufgabenstellung>`.

Das gesamte Projekt folgt dem Maven-Standardverzeichnisaufbau und somit finden siche die Sourcen unter `src/main/java` und die Tests unter `src/test/java`.

Sourcen – `src/main/java` Nachfolgend ist ein Ausschnitt für das Kapitel 2 gezeigt:



Test-Klassen – `src/test/java` Exemplarisch hier einige dazugehörige Tests:



Utility-Klassen Alle in den jeweiligen Kapiteln entwickelten nützlichen Utility-Methoden sind im bereitgestellten Eclipse-Projekt in Form von Utility-Klassen enthalten. Beispielsweise implementieren wir in Kapitel 5 einige hilfreiche Methoden, unter anderem `swap()` und `find()` (alle in Abschnitt 5.1.1). Diese kombinieren wir dann in einer Klasse `ArrayUtils`, die in einem eigenen Subpackage `util` liegt – für das Kapitel zu Arrays im Subpackage `ch05_arrays.util`. Gleiches gilt für die anderen Kapitel und Themengebiete.

1.3 Grundgerüst für die Unit Tests

Um den Rahmen des Buchs nicht zu sprengen, zeigen die abgebildeten Unit Tests jeweils nur die Testmethoden, jedoch nicht die Testklasse und die Imports. Damit Sie ein Grundgerüst haben, in das Sie die Testmethoden einfügen können sowie als Ausgangspunkt für eigene Experimente, ist nachfolgend eine typische Testklasse gezeigt:

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.time.LocalDate;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;
import org.junit.jupiter.params.provider.MethodSource;

public class SomeUnitTests
{
    @ParameterizedTest(name = "value at pos {index} ==> {0} should be perfect")
    @ValueSource(ints = { 6, 28, 496, 8128 } )
    void testIsPerfectNumberSimple(int value)
    {
        assertTrue(Ex03_PerfectNumbers.isPerfectNumberSimple(value));
    }

    @ParameterizedTest
    @CsvSource({"2017-01-01, 2018-01-01, 53", "2019-01-01, 2019-02-07, 5"})
    void testAllSundaysBetween(LocalDate start, LocalDate end, int expected)
    {
        var result = Ex09_CountSundaysExample.allSundaysBetween(start, end);

        assertEquals(expected, result.count());
    }

    @ParameterizedTest(name = "calcPrimes({0}) = {1}")
    @MethodSource("argumentProvider")
    void testCalcPrimesBelow(int n, List<Integer> expected)
    {
        List<Integer> result = Ex04_PrimNumber.calcPrimesBelow(n);

        assertEquals(expected, result);
    }

    // Parameter sind Listen von Werten => Stream<Arguments>
    static Stream<Arguments> argumentProvider()
    {
        return Stream.of(Arguments.of(2, List.of(2)),
                         Arguments.of(3, List.of(2, 3)),
                         Arguments.of(10, List.of(2, 3, 5, 7)),
                         Arguments.of(15, List.of(2, 3, 5, 7, 11, 13)));
    }
}
```

Neben den Imports und den ausgiebig genutzten parametrisierten Tests, die das Prüfen mehrerer Wertkombinationen auf einfache Weise erlauben, ist hier die Bereitstellung der Testeingaben über `@CsvSource` und `@MethodSource` in Kombination mit einem `Stream<Arguments>` gezeigt. Für Details schauen Sie bitte in Anhang B.

1.4 Anmerkung zum Programmierstil

In diesem Abschnitt möchte ich noch vorab etwas zum Programmierstil sagen, weil in Diskussionen ab und an einmal die Frage aufkam, ob man gewisse Dinge nicht kompakter gestalten sollte.

Gedanken zur Sourcecode-Kompaktheit

In der Regel sind mir beim Programmieren und insbesondere für die Implementierungen in diesem Buch vor allem eine leichte Nachvollziehbarkeit sowie eine übersichtliche Strukturierung und damit später eine vereinfachte Wartbarkeit und Veränderbarkeit wichtig. Deshalb sind die gezeigten Implementierungen möglichst verständlich programmiert und vermeiden etwa den ?-Operator für komplexere Ausdrücke. Dadurch ist vielleicht nicht jedes Konstrukt maximal kompakt, dafür aber in der Regel gut verständlich. Diesem Aspekt möchte ich in diesem Buch den Vorrang geben. Auch in der Praxis kann man damit oftmals besser leben als mit einer schlechten Wartbarkeit, dafür aber einer kompakteren Programmierung.

Beispiel 1

Schauen wir uns zur Verdeutlichung ein kleines Beispiel an. Zunächst betrachten wir die lesbare, gut verständliche Variante zum Umdrehen des Inhalts eines Strings, die zudem sehr schön die beiden wichtigen Elemente des rekursiven Abbruchs und Abstiegs verdeutlicht:

```
static String reverseString(final String input)
{
    // rekursiver Abbruch
    if (input.length() <= 1)
        return input;

    final char firstChar = input.charAt(0);
    final String remaining = input.substring(1);

    // rekursiver Abstieg
    return reverseString(remaining) + firstChar;
}
```

Die folgende deutliche kompaktere Variante bietet diese Vorteile nicht:

```
static String reverseStringShort(final String input)
{
    return input.length() <= 1 ? input :
           reverseStringShort(input.substring(1)) + input.charAt(0);
}
```

Überlegen Sie kurz, in welcher der beiden Methoden Sie sich sicher fühlen, eine nachträgliche Änderung vorzunehmen. Und wie sieht es aus, wenn Sie noch Unit Tests ergänzen wollen: Wie finden Sie passende Wertebelegungen und Prüfungen?

Außerdem sollte man bedenken, dass die obere Variante entweder bereits während der Kompilierung (Umwandlung in den Bytecode) oder später während der Ausführung und Optimierung automatisch in etwas Ähnliches wie die untere Variante konvertiert wird – eben mit dem Vorteil der besseren Lesbarkeit beim Programmieren.

Beispiel 2

Lassen Sie mich noch ein weiteres Beispiel anbringen, um meine Aussage zu verdeutlichen. Nehmen wir folgende Methode `countSubstrings()`, die die Anzahl der Vorkommen eines Strings in einem anderen zählt und für die beiden Eingaben "halloha" und "ha" das Ergebnis 2 liefert.

Zunächst implementieren wir das einigermaßen geradeheraus wie folgt:

```
static int countSubstrings(final String input, final String valueToFind)
{
    // rekursiver Abbruch
    if (input.length() < valueToFind.length())
        return 0;

    int count;
    String remaining;

    // startet der Text mit der Suchzeichenfolge?
    if (input.startsWith(valueToFind))
    {
        // Treffer: Setze die Suche nach dem gefundenen
        // Begriff nach der Fundstelle fort
        remaining = input.substring(valueToFind.length());
        count = 1;
    }
    else
    {
        // entferne erstes Zeichen und suche erneut
        remaining = input.substring(1);
        count = 0;
    }

    // rekursiver Abstieg
    return countSubstrings(remaining, valueToFind) + count;
}
```

Schauen wir uns an, wie man dies kompakt zu realisieren versuchen könnte:

```
static int countSubstringsShort(final String input, final String valueToFind)
{
    return input.length() < valueToFind.length() ? 0 :
           (input.startsWith(valueToFind) ? 1 : 0) +
           countSubstringsShort(input.substring(1), valueToFind);
}
```

Würden Sie lieber in dieser Methode oder der zuvor gezeigten ändern?

Übrigens: Die untere enthält noch eine subtile funktionale Abweichung! Bei den Eingaben von "XXXX" und "XX" »konsumiert« die erste Variante immer die Zeichen und findet zwei Vorkommen. Die untere bewegt sich aber jeweils nur um ein Zeichen weiter und findet somit drei Vorkommen.

Und weiter: Die Integration der oben realisierten Funktionalität des Weiterschiebens um den gesamten Suchstring in die zweite Variante wird zu immer undurchsichtigerem Sourcecode führen. Dagegen kann man oben das Weiterschieben um nur ein Zeichen einfach durch Anpassen des oberen `substring(valueToFind.length())`-Aufrufs umsetzen und diese Funktionalität dann sogar aus dem `if` herausziehen.

Gedanken zu `final` und `var`

Normalerweise bevorzuge ich, unveränderliche Variablen als `final` zu markieren. In diesem Buch verzichte ich mitunter darauf, vor allem in Unit Tests, um diese so kurz wie möglich zu halten. Ein weiterer Grund ist, dass die JShell das Schlüsselwort `final` nicht überall unterstützt, glücklicherweise aber an den wichtigen Stellen, nämlich für Parameter und lokale Variablen.

Local Variable Type Inference – `var` Seit Java 10 existiert die sogenannte Local Variable Type Inference, besser bekannt als `var`. Diese erlaubt es, auf die explizite Typangabe auf der linken Seite einer Variablendefinition zu verzichten, sofern sich der konkrete Typ für eine lokale Variable anhand der Definition auf der rechten Seite der Zuweisung vom Compiler ermitteln lässt:

```
var name = "Peter";                // var => String
var chars = name.toCharArray();    // var => char[]

var mike = new Person("Mike", 47); // var => Person
var hash = mike.hashCode();        // var => int
```

Insbesondere im Zusammenhang mit generischen Containern spielt die Local Variable Type Inference ihre Vorteile aus:

```
// var => ArrayList<String>
var names = new ArrayList<String>();
names.add("Tim");
names.add("Tom");
names.add("Jerry");

// var => Map<String, Long>
var personAgeMapping = Map.of("Tim", 47L, "Tom", 12L,
                              "Michael", 47L, "Max", 25L);
```

Konvention: `var` falls lesbarer Sofern die Verständlichkeit darunter nicht leidet, werde ich `var` an geeigneter Stelle verwenden, um den Sourcecode kürzer und klarer zu halten. Ist jedoch eine Typangabe für das Nachvollziehen von größerer Wichtigkeit, bevorzuge ich den konkreten Typ und vermeide `var` – die Grenzen sind aber fließend.

Konvention: `final` oder `var` Eine weitere Anmerkung noch: Zwar kann man `final` und `var` kombinieren, ich finde dies jedoch stilistisch nicht schön und verwende entweder das eine oder das andere.

Anmerkungen zu Methodensichtbarkeiten

Vielleicht fragen Sie sich, wieso die vorgestellten Methoden nicht als `public` markiert sind. Die in diesem Buch vorgestellten Methoden sind oftmals ohne Sichtbarkeitsmodifier dargestellt, da sie vor allem in demjenigen Package und Kontext aufgerufen werden, wo sie definiert sind. Dadurch sind sie sowohl für die begleitenden Unit Tests als auch für Experimente in der JShell problemlos aufrufbar. Manchmal extrahiere ich aus der Implementierung der Übungsaufgabe zur besseren Strukturierung und Lesbarkeit einige Hilfsmethoden. Diese sind dann in der Regel `private`, um diesen Umstand entsprechend auszudrücken.

Tatsächlich fasse ich die wichtigsten Hilfsmethoden in speziellen Utility-Klassen zusammen, wo diese für andere Packages natürlich `public` und `static` sind, um den Zugriff zu ermöglichen.

Blockkommentare in Listings

Beachten Sie bitte, dass sich in den Listings diverse Blockkommentare finden, die der Orientierung und dem besseren Verständnis dienen. In der Praxis sollte man derartige Kommentierungen mit Bedacht einsetzen und lieber einzelne Sourcecode-Abschnitte in Methoden auslagern. Für die Beispiele des Buchs dienen diese Kommentare aber als Anhaltspunkte, weil die eingeführten oder dargestellten Sachverhalte für Sie als Leser vermutlich noch neu und ungewohnt sind.

```
// Prozess erzeugen
final String command = "sleep 60s";
final String commandWin = "cmd timeout 60";
final Process sleeper = Runtime.getRuntime().exec(command);
//    ...

// Process => ProcessHandle
final ProcessHandle sleeperHandle = ProcessHandle.of(sleeper.pid()).
                                    orElseThrow(IllegalStateException::new);
//    ...
```

Gedanken zur Formattierung

Die in den Listings genutzte Formatierung weicht leicht von den Coding Conventions von Oracle[1] ab. Ich orientiere mich an denjenigen von Scott Ambler[2], der insbesondere (öffnende) Klammern in jeweils eigenen Zeilen vorschlägt. Dazu habe ich ein spezielles Format namens `Michaelis_CodeFormat` erstellt. Dieses ist im Projekt-Download integriert.

[1] http://www.oracle.com/technetwork/java/codeconv-138413.html
[2] http://www.ambysoft.com/essays/javaCodingStandards.html