
mitp
Elisabeth Jung
Java
Übungsbuch
Aufgaben mit vollständigen Lösungen
Für die Versionen Java 8 bis Java 17

Elisabeth Jung

Java Übungsbuch

Für die Versionen Java 8 bis Java 17

Aufgaben mit vollständigen Lösungen

mitp

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 978-3-7475-0450-5
1. Auflage 2021

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Sabine Schulz
Sprachkorrektorat: Petra Heubach-Erdmann
Coverbild: © maxec / stock.adobe.com
Satz: III-Satz, www.drei-satz.de

Inhaltsverzeichnis

Einleitung

Das Java Übungsbuch: Für die Versionen Java 8 bis Java 17 ist wie alle meine Übungsbücher aus der Erkenntnis entstanden, dass zu umfangreiche Beispiele mit komplizierten Algorithmen beim Lernen von Java am Anfang keine echte Hilfe bieten. Darum liegt der Schwerpunkt des Buches nicht auf der Umsetzung von komplizierten Vorgängen, sondern konzentriert sich stattdessen darauf, die in der Dokumentation nicht immer verständlich formulierten Erläuterungen zu Java-Klassen und -Interfaces mit einfachen Beispielen zu erklären und gleichzeitig die zugrunde liegenden Konzepte zu erörtern.

Das Java Übungsbuch: Für die Versionen Java 8 bis Java 17 wendet sich in erster Linie an Lehrer, Schüler und Studenten als Begleitliteratur zum Lernen der Programmiersprache Java, ist aber auch zum Selbststudium für alle Interessenten an dem Erlernen der Programmiersprache geeignet.

Durch die Einfachheit und Vollständigkeit der Aufgabenlösungen sowie die unterschiedlichen Lösungsmöglichkeiten erhält der Leser ein fundiertes Verständnis für die Aufgabenstellungen und deren Lösungen.

Durch das Lösen von Aufgaben soll der in Referenz- und Lehrbüchern von Java angebotene Stoff vertieft werden, und die dabei erzielten Ergebnisse können anhand der Lösungsvorschläge überprüft werden. Die Beispiele im Buch sind eher selten von zu komplexer Natur, sodass der eigentliche Zweck nicht in den Hintergrund tritt, und alle beschriebenen Themen können tiefgehend und präzise damit eingeübt werden.

Vorkenntnisse

Es ist Voraussetzung, dass der Leser zusätzlich mit einem Lehrbuch zu Java arbeitet bzw. bereits damit gearbeitet hat. Die grundlegenden Erläuterungen zu Java in diesem Buch können lediglich als Wiederholung des bereits vorhandenen Wissens dienen, reichen aber nicht aus, um die Sprache Java erst neu zu lernen.

Als weitere Voraussetzung gelten Grundlagen im Bereich der Programmierung und im Umgang mit dem Betriebssystem. Ein paralleler Zugriff auf die Java-Online-Dokumentation kann Hilfe zu den Java-Standard-Klassen bieten.

Aufbau des Buches

Jedes Kapitel beginnt mit einer kurzen und knappen Wiederholung des Stoffes, der in den Übungsaufgaben dieses Kapitels verwendet wird. Danach folgen alle Aufgabenstellungen der Übungen. Am Ende des Kapitels stehen gesammelt die Lösungen der Übungsaufgaben mit Kommentaren, Erläuterungen und Hinweisen.

Die Aufgaben haben unterschiedliche Schwierigkeitsgrade. Dieser wird im Aufgabenkopf durch ein bis drei Sternchen gekennzeichnet:

ein Sternchen für besonders einfache Aufgaben, die auch von Anfängern leicht bewältigt werden können

zwei Sternchen für etwas kompliziertere Aufgaben, die einen durchschnittlichen Aufwand benötigen

drei Sternchen für Aufgaben, die sich an geübte Programmierer richten und einen wesentlich höheren Aufwand oder die Kenntnis von speziellen Details erfordern

Die Programme aus früheren Übungen werden teilweise in späteren Übungen gebraucht und es wird auch immer wieder auf theoretische Zusammenhänge zurückgekommen oder hingewiesen.

Die Lösungsvorschläge haben umfangreiche Kommentare, sodass ein Verständnis für die durchgeführte Aufgabe auch daraus abgeleitet werden kann und dadurch jede einzelne Aufgabe im Gesamtkontext unabhängig erscheint.

In den Kapiteln 1, 2 und 3 liegt das Hauptmerkmal auf den Eigenheiten der objektorientierten Programmierung mit Java. Durch eine Vielzahl von Beispielen wird gezeigt, was die Java-Standard-Klassen und Interfaces an Funktionalitäten bieten und wie diese sinnvoll in die Definition von eigenen Klassen eingebettet werden können. Diese Kapitel enthalten zusätzlich Informationen zur Reflection-API von Java, der Definition von Annotationen und inneren Klassen sowie Neuerungen aus den Versionen 8 bis 13, die sich auf die neue Date&Time-API, Textblöcke, Compact Strings und die Weiterentwicklung von Interfaces beziehen. Mit Java 8 wurden sogenannte Default-Methoden eingeführt. Diese werden in der Literatur auch als »virtual extension«- bzw. »defender«-Methoden bezeichnet und Schnittstellen, die über derartige Methoden verfügen, als erweiterte Schnittstellen. Damit können Interfaces zusätzlich zu abstrakten Methoden konkrete Methoden in Form von Standard-

Implementierungen definieren und in Java wird die Mehrfachvererbung von Funktionalität ermöglicht. Neben Default-Methoden können Interfaces in Java nun auch statische Methoden enthalten. Anders als die statischen Methoden von Klassen werden diese jedoch nicht von abgeleiteten Typen geerbt.

Kapitel 4 beschäftigt sich im Detail mit Generics und dem Collection Framework mit all seinen generischen Klassen und Interfaces sowie mit der Definition von Enumerationen. Die Typinferenz für Methoden und beim Erzeugen von generischen Typen (der Diamond-Operator) sowie das Subtyping von parametrisierten und Wildcard-parametrisierten Typen sind ebenfalls Gegenstand der Themen aus diesem Kapitel.

Kapitel 5 erläutert das Exception-Handling.

Kapitel 6 beschäftigt sich mit den neuen Sprachmitteln von Java 8, Lambdas und Streams sowie mit weiteren Neuerungen aus den Versionen 8 bis 14, wie Switch-Expressions und Local Variable Type Inference.

Mit der Java-Version 8 haben sich ganz neue Betrachtungsweisen und Programmiertechniken in der Entwicklung von Applikationen mit Java eröffnet. Eine der wichtigsten Neuerungen in Java 8 sind neue Sprachmittel, die sogenannten Lambda-Ausdrücke, eine Art anonyme Methoden, die auf funktionalen Interfaces basieren. Diese besitzen jedoch eine viel kompaktere Syntax als Methoden. Das resultiert daraus, dass in ihrer Benutzung auf Namen, Modifikatoren, Rückgabetyp, `throws`-Klausel und in vielen Fällen auch auf Parameter verzichtet werden kann. Mit ihnen kann Funktionalität ausgeführt, gespeichert und übergeben werden, wie dies bisher nur von Instanzen in Java bekannt war.

Damit verbundene Themen wie die Gegenüberstellung zu anonymen Klassen, Syntax und Semantik, Behandlung von Exceptions, Scoping und Variable Capture, Methoden- und Konstruktor-Referenzen werden in den ersten Unterkapiteln des 6. Kapitels dieses Buches beschrieben und anhand von vielen Beispielen erläutert.

Des Weiteren finden Sie hier die Beschreibung aller neuen funktionalen Interfaces und deren Methoden. Die nachfolgenden Unterkapitel beschäftigen sich im Detail mit der Definition und Nutzung von Streams. Ein Stream besteht aus einer Folge von Werten (in der Literatur wird auch von Sequenzen von Elementen gesprochen), die nur teilweise von mehreren in einer Pipeline dazwischenliegenden Operationen ausgewertet und durch eine abschließende Operation bereitgestellt werden. Diese Operationen werden in Java als Methodenaufrufe formuliert, die Funktionalität in Form von Lambdas und Methoden-Referenzen entgegennehmen können und diese auf alle Elemente der Folge anwenden.

Mit einer Vielzahl von Aufgaben basierend auf Lambdas, Streams und Kollektoren (in denen Stream-Elemente angesammelt und reduziert werden können) werden die neuen Techniken angewandt und alle neuen Begriffe erklärt.

Kapitel 7 präsentiert das neue Java-Modulsystem. Mit dem neuen Modulsystem wurde Java selbst modular gemacht und es können eigene Applikationen und Bibliotheken modularisiert werden.

Java 9 führt das Modul als eine neue Programmkomponente ein. Das Erzeugen von Modulen und deren Abhängigkeiten führen dazu, dass der Zugriffsschutz in Java 9 restriktiver ist. Das Anlegen der erforderlichen Verzeichnisstrukturen für modulbasierte Applikationen, das Packaging von Modul-Code sowie die Implementierung von Services werden ebenfalls im Detail erklärt. Eine Vielzahl von Applikationen mit ausführlichen `.cmd`-Dateien für deren Ausführung ergänzen die theoretischen Erläuterungen aus diesem Kapitel.

In Kapitel 8 werden die Weiterentwicklungen aus der Programmiersprache mit den Versionen 14 bis 17 erläutert. Dazu gehören die Einführung von Records und Sealed Classes sowie das Pattern Matching.

Records wurden in der Version 14 entworfen, um die Wiederholungen von repetitivem Code in Datenklassen zu unterdrücken. Sie überlassen dem Compiler eine korrekte Generierung der Methoden `equals()`, `hashCode()`, `toString()` (die in Klassen, um eine Wertegleichheit von Objekten zu ermöglichen, überschrieben werden müssen) und von Zugriffsmethoden.

`record`-Klassen helfen in Kombination mit den in Java 15 neu eingeführten `sealed`-Klassen und -Interfaces, die auch mit Java 16 im Preview-Status bleiben und mit Java 17 finalisiert werden, die funktionalen Features von Java zu erweitern, insbesondere das Pattern Matching und in naher Zukunft die Destrukturierung von Objekten.

Sealed Classes und Interfaces sind Java-Datentypen, für die die Definition von Subtypen reduziert wird. Sie können nur von den in ihrer Deklaration angegebenen Typen erweitert bzw. implementiert werden.

Auch wenn es keine direkten Abhängigkeiten zwischen den Previews aus den JEPs 395, 394, 409, 406 und 405 gibt, die die Einführung dieser neuen Java-Datentypen sowie das Pattern Matching in Java beschreiben, so sind die von diesen vorgeschlagenen neuen Java-Features, wie mit vielen Beispielen in den Kapiteln 8 und 9 illustriert wird, sehr gut zusammen einsetzbar und im weitesten Sinne auch dafür gedacht.

Das Pattern Matching wurde in Java ursprünglich für den Abgleich von regulären Ausdrücken mit einem Text eingesetzt und für einen Vergleich von Typen im Zusammenhang mit dem `instanceof`-Operator und `switch` weiterentwickelt.

Der `instanceof`-Operator wurde erweitert, sodass anstelle eines Typ-Tests ein Musterabgleich-Test (»type test pattern«) durchgeführt wird. Dieser prüft die Übereinstimmung eines Zielobjekts mit einem vorgegebenen Mustertyp und erweist sich als sehr nützlich beim Schreiben von `equals()`-Methoden.

Mit Java 17 sind rund um das Pattern Matching weitere Funktionen im Zusammenhang mit Switch Statements und Switch Expressions realisiert worden. Damit werden die Restriktionen für den Typ des Ausdrucks, der im `switch` übergeben wird, weitestgehend aufgehoben. Bei einem klassischen `switch` waren zugelassen: ganzzahlige primitive Typen (`char`, `byte`, `short`, `int`) und die dazugehörigen Wrapper-Typen (`Character`, `Byte`, `Short`, `Integer`) sowie `String` und `enum`-Konstanten.

Diese Auswahl wurde nun auf ganzzahlige primitive Typen und beliebige Referenztypen erweitert, sodass `class`-, `enum`-, `record`- und `array`-Typen zugelassen sind, die zusammen mit einem `null`-`case`-Label und einem `default`-Label die Angaben in den `switch`-`case`-Labels ausmachen können.

Die Destrukturierung von Objekten wird zusammen mit Record Patterns und Array Patterns (JEP 405) die Entwickler von nachfolgenden Java-Versionen weiter beschäftigen.

Neu in dieser Auflage des Buches sind Tests mit JUnit 5 und Gradle, die in Kapitel 9 beispielhaft präsentiert werden.

JUnit 5 kann von der Website `https://junit.org/junit5/` unter »Latest Release« (aktuelle Version zum Zeitpunkt der Redaktion dieses Buches waren: Jupiter v5.7.1, Vintage v5.7.1, Platform v1.7.1) heruntergeladen werden.

Zum Testen von Applikationen werden, wie auch in den bevorstehenden Versionen von JUnit üblich, sogenannte Testklassen geschrieben. Sie beinhalten Methoden, die Testfälle beschreiben, den Rückgabetyp `void` aufweisen und durch Annotationen gekennzeichnet sind.

JUnit 5 führt darüber hinaus das Konzept eines ConsoleLaunchers ein, der benutzt werden kann, um Tests zu entwickeln, zu filtern und durchzuführen.

Um Ihnen ein gutes Verständnis für Details zu ermöglichen, wähle ich in diesem Buch die Ausführung über die Kommandozeile, die der ConsoleLauncher in diesem Fall ermöglicht.

Sicherlich sind Build-Tools wie Gradle und IDEs wie Eclipse, Intellij IDEA oder Maven eine große Hilfe nicht nur bei der Ausführung von JUnit-Tests, sondern generell in der Programmierung mit Java. Die Angabe von Details in diesem Zusammenhang würde den Rahmen dieses Buches jedoch sprengen.

In einem Unterkapitel in Kapitel 9 erfolgt eine kurze Beschreibung von Gradle und der Ausführung von Tests mit diesem Tool. Weil es gerade im Zusammenhang mit JUnit-Tests dem Anwender viel Kopfzerbrechen und Arbeit erspart, präsentiere ich es als Alternative zum ConsoleLauncher für die Durchführung von JUnit-Tests für die Java-Applikationen.

Eine neue Gradle-Version kann von der Website `https://gradle.org/releases/` heruntergeladen werden. Zum Zeitpunkt der Buchredaktion war die Version `v7.0.1` aktuell.

Weil der Schwerpunkt des Buches nicht auf der Umsetzung von aufwendigen Algorithmen liegen soll, verwende ich einfache Beispiele mit Zahlen, Buchstaben, Wörtern, Büchern, Wochentagen, geometrischen Figuren etc. und teilweise auch mit ganz abstrakten Klassennamen wie `Klasse1`, `Klasse2`, `KlasseA`, `KlasseB` etc.

An dieser Stelle möchte ich auf das dem Buch zugrunde liegende Konzept hinweisen, dass parallel zu einfachen Aufgaben, die zu allen eingeführten Definitionen und Begriffen gebracht werden, auch Aufgaben von einem höheren Schwierigkeits-

grad präsentiert werden. Dabei werden anhand von inhaltlichen Zusammenhängen zwischen den Beispielen viele Basiskonzepte von Java erläutert.

Ich habe generell versucht, keine Begriffe, Klassen und Komponenten zu benutzen, die nicht schon in vorangehenden Beispielen und Kapiteln definiert oder erläutert wurden. In den wenigen Fällen, wo es sich nicht vermieden ließ, wird darauf hingewiesen und auf die entsprechenden Stellen verwiesen.

Das Buch soll möglichst parallel zu einer Vielzahl von Java-Lehrbüchern eingesetzt werden können und einen Beitrag dazu leisten, die große Fülle von Informationen, die auf uns über die API-Dokumentation zukommt, besser einzuordnen und korrekt anwenden zu können.

Schrecken Sie nicht davor zurück, von Anfang an (gerade bei den schwierigeren Aufgaben) die Anforderungen aus dem Aufgabentext mit den Ergebnissen aus dem Lösungsvorschlag zu vergleichen (zumindest anfangs und vielleicht auch nur teilweise). Nahe an der Programmiersprache formuliert, sollen diese Aufgaben in erster Linie dazu dienen, das Programmieren mit Java zu erlernen, ohne sich gleichzeitig auf aufwendige Algorithmen zu konzentrieren. Es ist ja auch mit ein Grund, warum die Bücher vollständige Lösungsvorschläge beinhalten. Sie sind gerade dafür gedacht, die Theorie besser zu verstehen, aber auch gleichzeitig mit Beispielen einzuüben.

Andererseits verpflichten diese Bücher nicht, selbst auf die gleiche Lösung zu kommen, und enthalten nur Vorschläge zu den Lösungen von Aufgaben.

Anhand eines umfangreichen Index können Sie beim selbstständigen Programmieren im Buch nachschlagen, wenn Sie nach einem Lösungsansatz oder Hilfe beim Beseitigen von Fehlern suchen sollten.

Benötigte Software

Das aktuelle Java Development Kit der Java Standard Edition können Sie sich kostenlos von der Java-Homepage von Oracle `http://www.oracle.com/technetwork/java/javase/downloads/index.html` herunterladen. Das JDK umfasst sowohl die Software zur Programmerstellung als auch das JRE (Java Runtime Environment) für die Programmausführung.

Grafische Entwicklungsoberflächen sind, wie bereits erwähnt, keine Voraussetzung und auch nicht Bestandteil dieses Buches. Die Programme lassen sich grundsätzlich mit einem Texteditor wie z.B. Notepad++ oder auch Wordpad eingeben und über die Kommandozeile durch den Aufruf der Programme `javac`, `jar` und `java` übersetzen, paketieren und starten. Die vollständigen Programmaufrufe sind bei jeder Aufgabe angegeben.

Sollte Ihnen beim Erlernen der Programmiersprache selbst eine Entwicklungsumgebung nicht zu aufwendig oder unübersichtlich erscheinen, steht Ihnen nichts im Wege, die zu den Aufgaben zugehörigen Programm- und Klassendateien zum Testen oder auch Ergänzen in eine solche einzubetten.

Downloads

Die Website zum Buch mit der Adresse `http://www.mitp.de/0449` beinhaltet den plattformunabhängigen Quellcode der Lösungsvorschläge und die für Windows kompilierte ausführbare Version als Download-Archiv. Diese Archivdatei enthält alle Java-Quellcodes, übersetzten Klassen und Bilddateien in einer Verzeichnisstruktur, die mit der im Buch beschriebenen übereinstimmt. Zusätzlich finden Sie auf dieser Webseite die Datei `Java9Migration.pdf`, in der die Migration von Anwendungen nach Java 9 beschrieben wird, die durch verschiedene Kategorien von Modulen unterstützt werden kann.

Ich wünsche Ihnen viel Erfolg beim Programmieren mit Java.

Elisabeth Jung

Autorin

Elisabeth Jung ist freie Autorin und wohnhaft in Frankfurt am Main.

Nach dem Studium der Mathematik an der Universität Temeschburg in Rumänien hat Elisabeth Jung parallel zur klassischen Mathematik Grundlagen der Informatik und Fortran unterrichtet. Im Jahr 1982 hat sie bereits eine Aufgabensammlung für Fortran an der gleichen Universität veröffentlicht.

In den Jahren 1984 und 2001 hat Elisabeth Jung bei der Firma Siemens in Frankfurt in einer Vielzahl von Projektarbeiten umfangreiche Erfahrungen gesammelt in den Bereichen Programmiersprachen (Assembler, Fortran, Pascal, C, C++, Java), Datenbanken (Text-Retrieval, relationale Systeme, Client-Server-Architekturen) und der Entwicklung und dem Test von Hardware-naher Systemsoftware.

Seit 2001 beschäftigt sie sich mit ihren bevorzugten Themen, der Mathematik und den objektorientierten Programmiersprachen, insbesondere Java. Ihre Bücher sind aus der Erfahrung im Unterricht und dem Erlernen von Programmiersprachen entstanden, bei dem insbesondere das eigene Programmieren und die praktische Anwendung des Gelernten eine große Rolle spielt.

Kapitel 1

Klassendefinition und Objektinstanziierung

1.1 Klassen und Objekte

Klassen und Objekte bilden die Basis in der objektorientierten Programmierung. Eine Klasse ist eine Ansammlung von Attributen, die Eigenschaften definieren und Felder genannt werden, und von Funktionen, die deren Zustände und Verhaltensweisen festlegen und als Methoden bezeichnet werden. Felder und Methoden werden auch Member der Klasse genannt.

Klassen werden mit dem Schlüsselwort `class` eingeleitet und definieren eine logische Abstraktion, die eine Basisstruktur für Objekte vorgibt. Sie sind als eine Erweiterung der primitiven Datentypen zu sehen. Während Klassen Vorlagen (Modelle) definieren, sind Objekte konkrete Exemplare, auch Instanzen der Klasse genannt.

Eine Schnittstelle (Interface) ist eine reine Spezifikation, die definiert, wie eine Klasse sich zu verhalten hat. Sie wird mit dem Schlüsselwort `interface` eingeleitet und konnte zu den Anfangszeiten von Java keine Implementationen von Feldern und Methoden enthalten, mit Ausnahme von Konstantendefinitionen (die als `static` und `final` deklariert werden). Mit der Weiterentwicklung von Java wurden sowohl `public static`-Methoden als auch `public default`-Methoden in Interfaces zugelassen (Java 8). Um redundanten Code zu vermeiden und die Benutzung von Interfaces zu verbessern, wurden mit Java 9 zusätzlich `private`-Methoden zugelassen. Einige dieser Methoden können Implementierungen liefern. Wir werden im weiteren Verlauf darauf näher eingehen.

Ein Objekt einer Klasse wird in Java mit dem `new`-Operator und einem Konstruktor erzeugt. Damit werden auch seine Felder initialisiert und der erforderliche Speicherplatz für das Objekt reserviert. Ein Objekt wird über eine Referenz angesprochen. Eine Referenz entspricht in Java in etwa einem Zeiger in anderen Programmiersprachen und ist einem Verweis auf das Objekt gleichzustellen, womit dieses identifiziert werden kann.

Mit Referenztypen werden Datentypen bezeichnet, die im Gegensatz zu den primitiven Datentypen vom Entwickler selbst definiert werden. Diese können vom Typ einer Klasse, einer Schnittstelle oder eines Arrays sein. Ein Arraytyp identifiziert ein Objekt, das mehrere Werte von ein und demselben Typ speichern kann.

Die mit dem Modifikator `static` deklarierten Felder und Methoden in einer Klasse werden als Klassenfelder bzw. Klassenmethoden bezeichnet. Alle anderen Felder

und Methoden einer Klasse werden auch Instanzfelder bzw. Instanzmethoden genannt.

Die Klassen bilden in Java eine Klassenhierarchie. Jede Klasse hat eine Oberklasse, deren Felder und Methoden sie erbt. Die Oberklasse aller Klassen in Java ist die Klasse `java.lang.Object` (siehe dazu Kapitel 2, *Abgeleitete Klassen und Vererbung*).

Beim Definieren von Klassen ist zu beachten, dass eine Klasse eine in sich funktionierende Einheit darstellt, die alle benötigten Felder und Methoden definiert.

Konstruktoren

Für die Initialisierungen eines Objekts der Klasse werden Konstruktoren genutzt. Diese sind eine spezielle Art von Methoden. Sie haben den gleichen Namen wie die Klasse, zu der sie gehören, und verfügen über keinen Rückgabewert. Weil der `new`-Operator eine Referenz auf das erzeugte Objekt zurückgibt, ist eine zusätzliche Rückgabe von Werten in Konstruktoren nicht mehr erforderlich. Jede Klasse besitzt einen impliziten Konstruktor, der auch als Standard-Konstruktor bezeichnet wird. Dieser hat eine leere Parameterliste und übernimmt das Initialisieren der Instanzfelder mit den Defaultwerten der jeweiligen Datentypen. Eine Klasse kann mehrere explizite Konstruktoren definieren, die sich durch ihre Parameterlisten unterscheiden. Der parameterlose Konstruktor wird nur dann vom Compiler generiert, wenn die Klasse keinen expliziten Konstruktor definiert. Ist dies jedoch der Fall und die Klasse möchte auch den parameterlosen Konstruktor benutzen, so muss dieser ebenfalls explizit definiert werden.

Öffentliche (`public`) Konstruktoren werden von Klassen angeboten, damit auch ihre Benutzer, in der Literatur häufig als Clients bezeichnet, Instanzen davon erzeugen können. In diesem Zusammenhang wird immer öfter darauf hingewiesen (insbesondere wenn es um den produktiven Einsatz von Klassen geht), dass noch eine weitere Möglichkeit besteht, Instanzen von Klassen zu erzeugen, indem die Klasse eine oder mehrere statische Factory-Methoden zur Verfügung stellt. Laut Definition sind dies Klassenmethoden, die eine Instanz der Klasse zurückgeben. Ein Beispiel dafür sind die `valueOf()`-Methoden der Wrapper-Klassen für primitive Datentypen, wie `Integer`, `Double` etc. Diese Methoden müssen nicht unbedingt bei jedem Aufruf ein Objekt zurückliefern. Damit wird das unnötige Erzeugen von identischen Objekten vermieden und Klassen die Möglichkeit gegeben, mit bereits konstruierten Objekten zu arbeiten, indem diese abgespeichert und von den Methoden wiederholt zurückgegeben werden.

Der große Nachteil von derartigen Methoden, der auch beim Erstellen der Beispielklassen aus diesem Buch berücksichtigt wurde, ist jedoch, dass Klassen, die über keine `public` oder `protected` Konstruktoren verfügen, nicht erweitert werden können. Auch wenn damit Programmierer aufgefordert werden, des Öfteren Komposition anstelle von Vererbung (siehe dazu Kapitel 2) zu benutzen, ist und bleibt das Vererben von Klassen ein wesentlicher Bestandteil der Programmiersprache Java. Um »nahe an der Programmiersprache« alle Einzelheiten von Java mit möglichst einfachen Beispielen zu erlernen, brauchen wir Klassen, die das Ableiten

zulassen und somit ihren Unterklassen den Aufruf der Konstruktoren von Oberklassen ermöglichen.

Klassenfelder und Klassenmethoden

Klassenfelder gehören nicht zu einzelnen Objekten, sondern zu der Klasse, in der sie definiert wurden. Alle durchgeführten Änderungen ihrer Werte werden von der Klasse und allen ihren Objekten gesehen. Jedes Klassenfeld ist nur einmal vorhanden. Darum sollten Klassenfelder benutzt werden, um Informationen, die von allen Objekten der Klasse benötigt werden, zu speichern. Diese Felder können direkt über den Klassennamen angesprochen werden und stehen zur Verfügung, bevor irgendein Objekt der Klasse erzeugt wurde.

Klassenmethoden können ebenfalls über den Klassennamen angesprochen werden.

Innerhalb der eigenen Klasse können alle Klassenfelder und Klassenmethoden auch ohne Klassennamen angesprochen werden, sollten aber, um den Richtlinien der objektorientierten Programmierung zu genügen, möglichst mit diesem verwendet werden.

Instanzfelder und Instanzmethoden

Instanzfelder sind mehrfach vorhanden, da für jedes Objekt eine Kopie von allen Instanzfeldern einer Klasse erstellt wird. Die Instanzen einer Klasse unterscheiden sich voneinander durch die Werte ihrer Instanzfelder. Innerhalb einer Klasse kann der Zugriff darauf direkt über ihren Namen erfolgen oder in der Form `this.name` bzw. `obj.name` (wobei `obj` eine Referenz auf ein Objekt der Klasse ist).

Das Schlüsselwort `this` bezeichnet die Referenz auf das »aktuelle Objekt« der Klasse, auch das »aufrufende Objekt« genannt. Damit ist ein Zugriff auf Objekteigenschaften (in Instanzfeldern gespeichert) jederzeit möglich. Konstruktoren werden mit dem Schlüsselwort `new` aufgerufen und innerhalb eines anderen Konstruktors über das Schlüsselwort `this`, gefolgt von der Parameterliste.

Java-Programme

Die Java-Programmtechnologie basiert auf die Zusammenarbeit von einem Compiler und einem Interpreter. Die Programme werden zuerst kompiliert, was einer syntaktischen Prüfung und der Erstellung von Bytecode entspricht. Es entsteht dadurch noch kein ausführbares Programm, sondern ein plattformunabhängiger Code, der an einen Interpreter, die virtuelle Java-Maschine (JVM), übergeben wird. Die JVM ist ein plattformspezifisches Programm, das Bytecode lesen, interpretieren und ausführen kann.

Ein Java-Programm manipuliert Werte, die durch Typ und Name gekennzeichnet werden. Über die Festlegung von Name und Typ wird eine Variable definiert. Man spricht von Variablen in Zusammenhang mit einem Programm und von Feldern in Zusammenhang mit einer Klassendefinition.

Eine Variable ist im Grunde genommen ein symbolischer Name für eine Speicheradresse. Während für primitive Variablen der Typ des Werts, der an dieser Adresse gespeichert wird, gleich dem Typ des Namens der Variablen ist, wird im Falle einer Referenzvariablen nicht der Wert von einem bestimmten Objekt an dieser Adresse gespeichert, sondern die Angabe, wo das Programm den Wert (das Objekt) von diesem Typ finden kann.

Im Gegensatz zu lokalen Variablen, die keinen Standardwert haben und deswegen nicht verwendet werden können, bevor sie nicht explizit initialisiert wurden, werden alle Felder in einer Klassendefinition automatisch mit Defaultwerten initialisiert (mit `0`, `0.0`, `false` primitive Typen und mit `null` Referenztypen).

Die Definition einer Referenzvariablen besteht aus dem Namen der Klasse bzw. eines Interface gefolgt vom Namen der Variablen. Eine so definierte Referenzvariable kann eine Referenz auf ein beliebiges Objekt der Klasse oder einer Unterklasse oder den Defaultwert `null` aufnehmen. Weil Arrays als Objekte implementiert werden, müssen Arrays mit einem Array-Initialisierer oder mit dem `new`-Operator erzeugt werden.

Bevor ein Programm Objekte von einer Klasse bilden kann, wird diese mit dem Java-Klassenlader (Klasse `java.lang.ClassLoader`) geladen und mit dem Java-Bytecode-Verifier geprüft.

Nach der Art der Ausführung existieren mehrere Arten von Java-Programmen:

- Ein Java-Applet ist ein Java-Programm, das im Kontext eines Webbrowsers mit bestimmten Sicherheitseinschränkungen abläuft. Es wird mittels einer HTML-Seite gestartet und kann im Browser oder mithilfe des Appletviewers ausgeführt werden. Applets wurden mit der Version 9 von Java als deprecated gekennzeichnet und werden nicht mehr weiterentwickelt.
- Ein Servlet ist ein Java-Programm, das im Kontext eines Webservers abläuft.
- Eine Java-Applikation ist ein eigenständiges Programm, das direkt von der JVM gestartet wird. Alle nachfolgenden Beispiele werden als Java-Applikationen präsentiert.

Jede Java-Applikation benötigt eine `main()`-Methode, die einen Eingangspunkt für die Ausführung des Programms durch die JVM definiert. Diese Methode muss für alle Klassen der JVM zugänglich sein und deshalb mit dem Modifikator `public` definiert werden. Sie muss auch mit dem Modifikator `static` als Klassenmethode deklariert werden, da ein Aufruf dieser Methode möglich sein muss, ohne dass eine Instanz der Klasse erzeugt wurde. Von hier aus werden alle anderen Programmabläufe gesteuert.

Auf die Definition der Parameterliste der `main()`-Methode wird in nachfolgenden Programmbeispielen eingegangen.

Gleich in den ersten Beispielen werden für Bildschirmausgaben die Methoden `System.out.print(...)` und `System.out.println(...)` der Klasse `java.io.PrintStream` verwendet. Mit der Methode `System.out.print(...)` wird in der

gleichen Bildschirmzeile weitergeschrieben, in der eine vorangehende Ausgabe erfolgte. Die Methode `System.out.println()` ohne Parameter schließt eine vorher ausgegebene Bildschirmzeile ab und bewirkt, dass danach in eine neue Zeile geschrieben wird. Ein Aufruf der Methode `System.out.println(...)` mit Parameter ist äquivalent mit dem Aufruf von `System.out.print(...)` gefolgt von einem Aufruf von `System.out.println()` ohne Parameter, das heißt, dieser Aufruf führt immer zu einem Zeilenende. Auf die Definition von diesen Methoden kommen wir, in der Beschreibung der Java-Standard-Klasse `java.lang.System` noch einmal zurück.

Ein Programm wird als Quelltext in einer oder mehreren `.java`-Dateien und als übersetztes Programm in einer oder mehreren `.class`-Dateien abgelegt.

Aufgabe 1.1 ☆

Definition einer Klasse

Definieren Sie eine Klasse `KlassenDefinition`, die die `main()`-Methode als einzige Klassenmethode implementiert. Aus dieser soll die Bildschirmanzeige »Dies ist eine einfache Klassendefinition« erfolgen.

Hinweise für die Programmierung

Ein Erzeugen von Instanzen der Klasse ist nicht erforderlich.

Achten Sie auf den richtigen Abschluss der Ausgabezeile.

Java-Dateien: `KlassenDefinition.java`
Programmaufruf: `java KlassenDefinition`

Aufgabe 1.2 ☆

Objekt (Instanz) einer Klasse erzeugen

Definieren Sie eine Klasse `ObjektInstanziierung`, die in einem parameterlosen Konstruktor die Bildschirmanzeige »Instanz einer Java-Klasse erzeugen« vornimmt und in ihrer `main()`-Methode eine Instanz der Klasse erzeugt.

Java-Dateien: `ObjektInstanziierung.java`
Programmaufruf: `java ObjektInstanziierung`

1.2 Das Überladen von Methoden

Eine Klasse kann mehrere Methoden mit gleichem Namen besitzen, wenn diese eine verschiedene Anzahl von Parametern bzw. Parameter von unterschiedlichen Typen im Methodenkopf definieren. Dabei ist ohne Bedeutung, ob es sich um Klassen- oder Instanzmethoden handelt.

Parallel zur Parameterliste unterscheidet sich auch die Aufrufsyntax der Methode. Dieses Konzept ist unter dem Namen »Überladen von Methoden« bekannt.

Aufgabe 1.3 ☆

Eine Methode überladen

Definieren Sie eine Klasse `QuadratDefinition`, die ein Instanzfeld `a` vom Typ `int` besitzt, das die Seitenlänge eines Quadrats angibt. Im Konstruktor der Klasse wird ein `int`-Wert zum Initialisieren des Instanzfelds übergeben.

Implementieren Sie zwei Methoden für die Berechnung des Flächeninhalts eines Quadrats mit der Formel `f = a*a`. Definieren Sie eine parameterlose Instanzmethode `flaeche()` und eine Klassenmethode, die die Instanzmethode überlädt und eine Referenz vom Typ der eigenen Klasse übergeben bekommt.

Die Klasse `QuadratDefinitionTest` erzeugt eine Instanz der Klasse `QuadratDefinition`, berechnet auf zwei Arten deren Flächeninhalt über den Aufruf der Methoden der Klasse und zeigt die errechneten Ergebnisse am Bildschirm an.

Java-Dateien: `QuadratDefinition.java`, `QuadratDefinitionTest.java`
Programmaufruf: `java QuadratDefinitionTest`

1.3 Die Datenkapselung, ein Prinzip der objektorientierten Programmierung

Den Feldern und Methoden einer Klasse können über Modifikatoren verschiedene Sichtbarkeitsebenen zugeordnet werden.

Der bereits erwähnte Modifikator `public` sagt aus, dass der Zugriff auf Member einer Klasse von überall aus erfolgen kann, von wo aus auch die Klasse erreichbar ist.

Sind die Felder oder Methoden mit `private` definiert, können sie nur innerhalb der eigenen Klasse direkt angesprochen werden. Felder sollten immer als `private` definiert werden, wenn die Zuweisung von unzulässigen Werten verhindert werden soll. Dies ist der Fall, wenn sie von einer eigenen Methode der Klasse, die diesen Wert auch ändern kann, verwendet oder weitergegeben werden.

Definiert die Klasse keine Einschränkungen diesbezüglich oder einen zugelassenen Wertebereich für Felder innerhalb, von dem auch andere Klassen Werte setzen können, sollte sie über Zugriffsmethoden (»accessor-methods«) verfügen, die die Werte dieser Felder zurückgeben und ggf. setzen können. Dies entspricht dem sogenannten Prinzip der Datenkapselung: Auf die Felder einer Klasse soll nur mithilfe von Methoden der Klasse zugegriffen werden können.