

Michael Kofler

Aktuell  
zu  
Java 25 LTS

Java

Java

Java

Java

JAVA

Java

Java

Java

Der Grundkurs

Java

Java

Java

Java

JAVA

- ▶ Eine kompakte Einführung in Java
- ▶ Auf einen Blick: objektorientierte Programmierung, Generics, Vererbung, Bibliotheken und mehr
- ▶ Mit Codebeispielen, Übungen und Lösungen



Alle Codebeispiele zum Download



Rheinwerk  
Computing

# Vorwort

Gleichzeitig mit dem Erscheinen dieses Buchs wird Java 30 Jahre alt – ein beeindruckendes Alter für eine Programmiersprache, die bis heute relevant ist! Im Tiobe-Index, der die Popularität von Programmiersprachen misst, pendelt Java seit Jahren zwischen Platz zwei und vier. (Platz eins gehört – vorerst wohl uneinholbar – Python.)

Vor drei Jahrzehnten löste Java eine Revolution aus. Es war damit möglich, auf einem Betriebssystem (z. B. Windows) kompilierte Programme auch auf anderen Betriebssystemen auszuführen (Unix, Linux). Dieses Konzept war damals neu und passte genau zum Internet-Hype, der gerade begann.

Heute ist Java vor allem aus zwei Gründen nicht aus der IT-Welt wegzudenken:

- ▶ Einerseits ist Java eine ideale Sprache, um objektorientierte Programmierung kennenzulernen: Java ist syntaktisch sehr exakt. Das hilft, sich mit Konzepten wie Methoden, Klassen und Schnittstellen anzufreunden. Und im Unterschied zu moderneren Sprachen wie Kotlin oder Swift genießt Java bis heute eine riesige Verbreitung abseits von Nischen.
- ▶ Andererseits hat sich Java als Sprache für Server- und Web-Backends etabliert. In Banken, Versicherungen und anderen großen Firmen laufen auf vielen Servern Java-Programme und wickeln Geschäftsprozesse ab. Weil dieser Code weiterentwickelt und gewartet werden muss, sind Java-Kenntnisse auf dem Job-Markt so wertvoll.

## Modernes Java

Java ist als Open-Source-Software auf jedem Rechner kostenlos installierbar. Die Rechte an der Sprache gehört allerdings Oracle. Die Firma hat Java zusammen mit diversen anderen Software-Produkten bei der Über-

nahme von Sun erworben. Oracle ist es nach anfänglichen Schwierigkeiten überraschend gut gelungen, Java in kleinen Schritten zu modernisieren. In halbjährlich neuen Versionen werden inkrementell neue Features implementiert.

Das heutige Java ist noch immer als die Programmiersprache erkennbar, die IT-Veteranen vor 20 oder 25 Jahren kennenlernten. Aber im Unterschied zu alten Java-Versionen ist Java heute besser denn je für Java-Einsteiger geeignet. Sie können Ihre ersten Java-Programme mit minimalem Overhead entwickeln, was gerade im Unterricht ein riesiger Vorteil ist. Gleichzeitig bieten aktuelle Java-Versionen Features, mit denen erfahrene Entwicklerinnen modernen, gut les- und wartbaren Code formulieren können: Records, Sealed Classes, Pattern Matching, Flexible Constructors etc.

## Dieses Buch

In diesem Buch versuche ich den Spagat, Ihnen moderne Java-Techniken zu vermitteln und gleichzeitig Rücksicht darauf zu nehmen, dass viele Firmen bei der Auswahl der Java-Version konservativ agieren.

Auf den folgenden Seiten präsentiere ich Ihnen Java in Form eines Grundkurses. Sie lernen die wichtigsten Sprachelemente sowie ausgewählte Klassen aus der riesigen Java-Standardbibliothek kennen:

- ▶ Installation und erste Schritte (Hello, World!)
- ▶ Variablenverwaltung, Datentypen, Zahlen, Zeichenketten und Arrays
- ▶ Operatoren
- ▶ Verzweigungen, Schleifen
- ▶ Methoden, Exceptions, Klassen samt Vererbung und Schnittstellen
- ▶ generische Klassen, Lambda-Ausdrücke und Collections
- ▶ Umgang mit Dateien
- ▶ JavaFX
- ▶ Pakete, Module und Librarys

Viele konkrete Beispiele zeigen Ihnen dabei, wie Sie die Sprachelemente von Java und elementare Klassen richtig anwenden. Wiederholungsfragen und Übungen helfen Ihnen dabei, Ihr frisch erlerntes Wissen zu festigen.

Dieser Grundkurs richtet sich gleichermaßen an Leserinnen und Leser, die gerade im Unterricht oder mit KI-Hilfe Java als erste (oder zweite ...) Programmiersprache erlernen, als auch an Personen mit IT-Vorwissen, die rasch und ohne viele Umwege die Syntax von Java verstehen wollen.

## Programmieren lernen im KI-Zeitalter

Wie lernt man programmieren? Nicht durch das Lesen eines Buchs, nicht durch das Kopieren von Code aus dem Internet, nicht durch noch so clevere KI-Prompts. Sie müssen selbst intensiv programmieren und üben! Mit anderen Worten: Auch wenn ich Ihnen die Java-Grundlagen in verdaulichen Häppchen präsentiere, kann und will ich Ihnen nicht die Arbeit abnehmen, selbst Code zu entwickeln.

Nach einem Jahrzehnt Java-Unterricht auf der Fachhochschule in Kapfenberg bin ich überzeugt davon, dass Sie so viel und so früh wie möglich selbstständig programmieren sollen. Daraus resultiert der Aufbau dieses Buchs: Es beginnt nicht mit der Theorie der Objektorientierung, sondern mit Grundlagen, Beispielen und Übungen, die sich auch ohne die Strukturierung des eigenen Codes in Klassen lösen lassen. Am Beginn stehen Variablen, Schleifen und Methoden. (Nur keine Angst: Die Objektorientierung kommt nicht zu kurz!)

KI-Tools können Sie im Lernprozess entscheidend unterstützen. Allerdings liegt in diesen Hilfsmitteln auch eine Gefahr: Wenn Sie die ganze Arbeit an die KI delegieren und jede Übung damit lösen, wird der Lernerfolg ausbleiben. Gerade in den ersten Wochen sollten Sie unbedingt auf die von vielen Editoren angebotene automatische Code-Ergänzung verzichten und KI-Tools stattdessen gezielt zur Recherche und Fehlersuche einsetzen.

Ich will hier keine falschen Erwartungen wecken: Auch mit KI-Tools bleibt das Erlernen neuer Konzepte ein zeitaufwendiger und mühseliger Prozess. Umgekehrt gilt aber: Sobald Sie einmal ein solides Fundament aufgebaut haben und Java verstehen, in Java-Konzepten denken, Java gleichsam verinnerlichen, beschleunigen KI-Tools die Entwicklung praxistauglichen Codes enorm. Mein Ziel beim Verfassen dieses Buchs war es, Ihnen dieses Fundament mit vielen Beispielen und Übungen zu vermitteln.

## Neu in dieser Auflage

In der hier vorliegenden fünften Auflage dieses Buchs hat sich wieder eine Menge geändert:

- ▶ Ich habe das Buch mit seinen hundert (!) Beispielprojekten umfassend im Hinblick auf die zahlreichen Neuerungen in Java 25 aktualisiert.
- ▶ Ein eigenes Kapitel gibt Tipps, wie Sie Ihren Lernprozess mit KI-Hilfe beschleunigen können.
- ▶ Ebenfalls neu ist ein Kapitel, das Ihnen zeigt, wie Sie mit Java einen (beinahe) praxistauglichen REST-Server entwickeln und in einem zweiten Programm mit HTTP-Requests steuern können.

## Viel Erfolg!

Aus meiner Sicht ist Programmieren eine faszinierende Beschäftigung. Die erfolgreiche Lösung einer kniffligen Programmierübung kann Ihnen Erfolgserlebnisse wie im Sport bescheren. Lassen Sie sich von meiner Begeisterung für das Programmieren anstecken, und fangen Sie an – jetzt!

Michael Kofler (<https://kofler.info>)

# Kapitel 1

## Hello, World!

Traditionell ist *Hello, World* das erste Programm in jeder Programmieranleitung bzw. in jedem Programmierbuch. Die Aufgabe dieses Programms besteht darin, die Zeichenkette 'Hello, World!' auf dem Bildschirm bzw. in einem Terminalfenster auszugeben.

»Eine ziemlich triviale Aufgabe«, werden Sie einwenden. »Dazu muss ich nicht das Programmieren lernen!« Damit haben Sie natürlich recht. Tatsächlich besteht der Sinn des Hello-World-Programms nicht darin, eine Zeichenkette auszugeben, sondern vielmehr darin, die Syntax und Werkzeuge einer neuen Programmiersprache erstmals auszuprobieren.

Genau darum geht es in diesem Kapitel: Sie lernen die wichtigsten Eigenschaften von Java kennen, erfahren, was Sie installieren müssen, bevor Sie Ihr erstes Programm verfassen können, und werden mit dem Editor Visual Studio Code oder der Entwicklungsoberfläche IntelliJ IDEA vertraut.

### 1.1 Einführung

*Programmieren* heißt, einem Computer in einer für ihn verständlichen Sprache Anweisungen zu geben. Als Nichtprogrammierer und oder Nichtprogrammiererin können Sie Computerprogramme selbstverständlich schon *anwenden*, im Web surfen, eine Mail verfassen oder eine Smartphone-App bedienen. Aber wenn Sie ein Programm benötigen, das es in dieser Form noch nicht gibt, dann müssen Sie es selbst entwickeln.

Natürlich gibt es auch ganz pragmatische Gründe: Sie wollen (oder müssen?) programmieren lernen, weil dies Teil Ihrer Schul- oder Universitätsausbildung ist. Oder es reizt Sie einfach, die IT-Welt besser verstehen zu können. Aus meiner persönlichen Sicht hat Programmieren auch etwas Spielerisches an sich, wie das Lösen von Denksporträtseln.

### »Die Beispiele in diesem Buch sind so langweilig!«

Wenn Sie dieses Buch durchblättern, werden Sie schnell feststellen, dass ich Ihnen hier leider nicht zeigen kann, wie Sie ein tolles Spiel oder ein komplexes Web-Service programmieren.

Zwar enthält Kapitel 20, »JavaFX«, einen knappen Einstieg in die Programmierung grafischer Benutzeroberflächen, aber ansonsten dominieren Textmodusprogramme. Warum?

Dieses Buch konzentriert sich auf die Java-Grundlagen. Diese Grundkenntnisse können am besten in kompakten, kleinen Programmen vermittelt werden, also *ohne* grafische Oberfläche und *ohne* Einbindung in eine Webseite. Sorry! Sobald Sie dieses Buch durchgearbeitet haben, verfügen Sie über ein solides Fundament, mit dem Sie sich dann weiter in ein Java-Teilgebiet einarbeiten können, z. B. in die Entwicklung von Webanwendungen.

## 1.2 Java installieren

Bevor Sie Ihr erstes Java-Programm verfassen und ausführen können, müssen Sie das sogenannte *Java Development Kit* (JDK) installieren. Dieses Software-Paket enthält diverse Kommandos und Bibliotheken zum Kompilieren (also das Umwandeln Ihres Quellcodes in eine ausführbare Form) und Ausführen von Java-Code.

Unter macOS oder Windows laden Sie die aktuelle JDK-Version am einfachsten von der folgenden Website herunter:

<https://www.oracle.com/java/technologies/downloads>

### JDK versus JRE

JRE steht für *Java Runtime Environment*. Damit können Sie Java-Programme nur ausführen, nicht aber selbst entwickeln. Für dieses Buch reicht das nicht aus.

Sollten Sie unter Linux arbeiten, öffnen Sie ein Terminalfenster und installieren dort das in Ihrer Distribution enthaltene openJDK-Paket. Dabei müssen Sie statt 25 gegebenenfalls eine andere Versionsnummer angeben. Unter Ubuntu können Sie mit `apt search openjdk` feststellen, welche Versionen zur Auswahl stehen.

```
sudo apt install openjdk-25-jdk          (Ubuntu)
sudo dnf install java-latest-openjdk    (Fedora)
```

### Zu viele Köche verderben den Brei

Die Installation von Java ist ein Kinderspiel. Deswegen kommt es in der Praxis recht oft vor, dass auf einem Rechner *mehrere* Java-Versionen installiert sind. Auch das ist erlaubt, führt aber mitunter zu Konfusion im Java-Editor. Neue Features funktionieren nicht, weil Ihre Entwicklungsumgebung die falsche Version verwendet. Die Lösung besteht darin, die Entwicklungsumgebung oder das aktive Projekt richtig zu konfigurieren. Tipps dazu gebe ich in den folgenden Abschnitten.

### Welche Versionsnummer?

Das von Oracle geleitete OpenJDK-Projekt veröffentlicht alle sechs Monate eine neue Java-Version. Zumeist werden dabei lediglich kleinere Neuerungen in der Programmiersprache realisiert, die für den Java-Einstieg selten relevant sind. Die Grundformel für die private Nutzung lautet: Installieren Sie einfach die neueste Version!

Für die Arbeit mit diesem Buch wäre es gut, wenn Sie Java 25 oder eine neuere Version verwenden. Java 25 markiert einen Meilenstein, weil diese Version den Java-Einstieg deutlich erleichtert! Es gibt eine Reihe neuer Features, die den Code-Aufbau winziger Programme vereinfachen.

Das ist auch aus didaktischer Sicht eine riesige Erleichterung: Als Vortragende müssen Sie für *Hello, World!* nicht diverse Details erklären, die zu Beginn weder relevant noch verständlich sind. Weil ich weiß, dass gerade in Bildungseinrichtungen die aktuellste Java-Version oft *nicht* installiert ist, nehme ich in diesem Buch auch auf ältere Versionen Rücksicht.

Java 25 ist eine Version mit *Long Time Support* (LTS). Das ist vor allem für Unternehmen relevant, die Java bei einer Firma beziehen, die entsprechenden Support anbietet (Oracle, Red Hat etc.). Davon losgelöst finden LTS-Versionen aber eine größere Verbreitung und sind insofern empfehlenswert. Die nächste LTS-Version wird voraussichtlich Java 29 sein und im Herbst 2027 erscheinen.

### Beispieldateien

Auf der folgenden Webseite finden Sie die Beispiele aus den folgenden Kapiteln in Form einer ZIP-Datei. Sie können die Beispiele unter Windows, Linux oder macOS nutzen. Lesen Sie dazu die Readme-Datei zu den Beispieldateien!

[https://www.rheinwerk-verlag.de/java\\_6127](https://www.rheinwerk-verlag.de/java_6127)

### Alternative JDKs

Oracle kaufte 2010 die Firma Sun auf und übernahm damit auch die Kontrolle über das von Sun entwickelte Java. Java war zu diesem Zeitpunkt bereits Open-Source-Code, und daran hat sich nichts geändert. Deswegen gibt es neben Oracle mehrere Firmen und Organisationen, die Java kostenlos oder gegen Bezahlung und mit Support anbieten. 2021 haben sich mehrere namhafte Firmen (darunter Google, IBM und Microsoft) im Projekt *Adoptium* zusammengetan, um gemeinsam eine OpenJDK-Implementierung zu warten. Dieses JDK können Sie hier herunterladen:

<https://adoptium.net>

Sämtliche JDK-Versionen basieren letztlich auf dem gleichen Code und sind kompatibel zueinander. Fortgeschrittenen Entwicklern, die mehrere JDK-Versionen parallel auf ihrem Rechner verwalten möchten, empfehle ich dringend den *Software Development Kit Manager* (SDKMAN). Dieses kostenlose Script hilft unter macOS und Linux beim Download und Wechsel zwischen verschiedenen JDK-Versionen:

<https://sdkman.io>

## 1.3 IntelliJ IDEA

Bevor Sie losprogrammieren können, benötigen Sie einen Editor, der Ihren Java-Code *versteht*. Die Minimalanforderungen bestehen darin, dass das Programm verschiedene Elemente Ihres Codes in unterschiedlichen Farben darstellt, Ihre Eingaben vervollständigt und einen Button zum Starten Ihres Codes zur Verfügung stellt.

Die Auswahl geeigneter Kandidaten ist grenzenlos. In diesem Buch beziehe ich mich auf *IntelliJ IDEA*. Dabei handelt es sich um eine vollständige Java-Entwicklungsumgebung, also ein Programm, das Sie bei allen Arbeitsschritten rund um die Programmentwicklung unterstützt. (IDEA steht für *Integrated Development Environment Application*. Im Weiteren lasse ich dieses Kürzel einfach weg.)

Warum gerade IntelliJ? In den vergangenen Jahren hat sich das Programm als die populärste Entwicklungsumgebung für Java etabliert. IntelliJ glänzt nicht nur durch Funktionen für Profis, sondern unterstützt Sie auch beim Java-Einstieg – z. B., indem das Programm Korrekturen oder Verbesserungen in Ihrem Code vorschlägt.

### IntelliJ installieren

Auf der folgenden Website stehen eine kostenlose Community-Variante von IntelliJ sowie eine kommerzielle Ultimate-Version mit noch mehr Funktionen zur Wahl. Passen Sie auf, dass Sie nicht versehentlich die Profiversion herunterladen. Die Community-Variante ist für dieses Buch mehr als ausreichend!

<https://www.jetbrains.com/idea/download>

Zur Installation führen Sie unter Windows einfach die \*.exe-Datei aus. Unter macOS öffnen Sie zur Installation die DMG-Datei und führen dann den darin enthaltenen PKG-Installer aus.

Einzig unter Linux ist die Sache etwas komplizierter: In einem Terminal führen Sie die folgenden Kommandos aus, um das komprimierte TAR-Archiv in Ihrem Heimatverzeichnis auszupacken und IntelliJ zu starten:

```
cd
tar xzf Downloads/ideaIC-<nnn>.tar.gz
./idea-IC-<nnn>/bin/idea.sh
```

Beim ersten Start verankert der Setup-Assistent IntelliJ im Menü, sodass Sie die Entwicklungsumgebung in Zukunft im Startmenü finden.

In den Einstellungen von IntelliJ können Sie sich zwischen einem hellen und einem dunklen Farbschema entscheiden. IntelliJ IDEA bietet Ihnen außerdem die Möglichkeit an, zusätzliche Plug-ins zu installieren. Darauf sollten Sie vorerst verzichten. Die IntelliJ IDEA enthält bereits in der Grundausstattung mehr Funktionen, als Sie momentan benötigen.

### Tipp

Sie können IntelliJ auch über die JetBrains Toolbox installieren. Das erleichtert spätere Updates und ist empfehlenswert, wenn Sie auf Ihrem Rechner auch andere Programme von JetBrains verwenden:

<https://www.jetbrains.com/toolbox-app>

### »Hello, World!« in IntelliJ

Um ein erstes Java-Projekt zu starten, führen Sie **FILE • NEW • PROJECT** aus. Im folgenden Dialog (siehe Abbildung 1.1). geben Sie dem Projekt einen Namen und deaktivieren die Option **ADD SAMPLE CODE**.

In seltenen Fällen passiert es, dass im SDK-Auswahlmenü das am Rechner installierten JDK fehlt. In diesem Fall wählen Sie den Eintrag **ADD JDK** aus. Im nächsten Schritt können Sie das JDK-Verzeichnis angeben. Die üblichen Orte sind:

- ▶ Windows: `C:\Program Files\Java\jdk-25`
- ▶ macOS: `/Library/Java/JavaVirtualMachines/jdk-25.jdk/Contents/Home`
- ▶ Linux: `/usr/lib/jvm/java-25-openjdk-amd64`

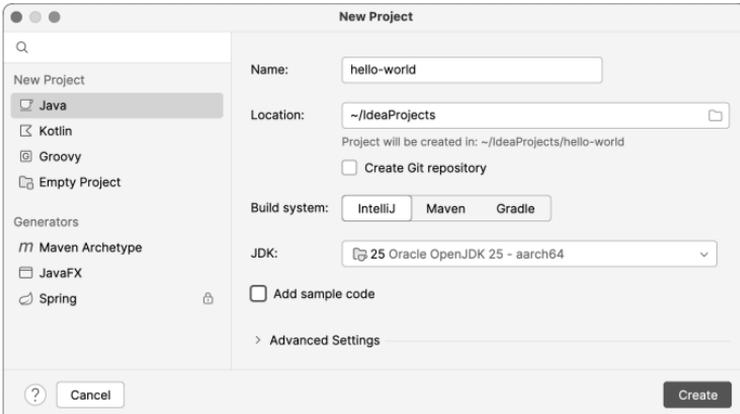


Abbildung 1.1 Ein neues Java-Projekt einrichten

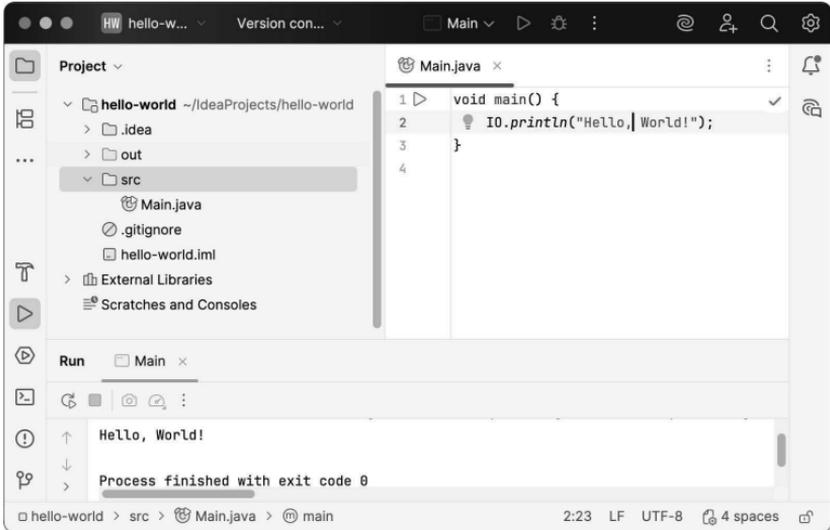
Unter macOS können Sie das erforderliche Verzeichnis im Terminal ermitteln, indem Sie `/usr/libexec/java_home` ausführen. Vergleichbare Hilfsmittel für Windows und Linux gibt es leider nicht.

Das Ziel dieses Kapitels ist ja ein winziges Programm, das den Text »Hello, World!« ausgibt. Wenn Sie beim Einrichten des neuen Projekts `ADD SAMPLE CODE` aktiviert haben (was normalerweise empfehlenswert ist), enthält das neue Projekt bereits die Klassendatei `Main.java` mit allen erforderlichen Codezeilen. Ich haben Ihnen aber dazu geraten, die Option nicht zu setzen. Das gibt Ihnen die Möglichkeit, bei diesem Einführungsbeispiel wenigstens ein bisschen selbst beizutragen.

Dazu klicken Sie im Dialogblatt `PROJECT` den Eintrag `SRC` mit der rechten Maustaste an, führen das Kontextmenükommando `NEW FILE` aus und benennen die Datei `Main.java`. Anschließend geben Sie die folgenden drei Zeilen ein:

```
// Java-25-Variante
void main() {
    IO.println("Hello, World!");
}
```

Starten Sie das Miniprogramm nun durch einen Klick auf den grünen Pfeil-Button bzw. durch das Menükommando RUN • RUN MAIN.



**Abbildung 1.2** Ein typisches Setup in IntelliJ: links die Projektübersicht, rechts eine Codedatei, unten die Ausgaben eines Programms

Es kann sein, dass IntelliJ den Code als fehlerhaft betrachtet und die Ausführung verweigern. Die drei wahrscheinlichsten Fehlerursachen sind:

- ▶ Auf Ihrem Rechner ist das Java-JDK 25 oder neuer nicht verfügbar.
- ▶ Java 25 ist installiert, aber IntelliJ verwendet eine ältere JDK-Version.
- ▶ Java 25 ist verfügbar, aber der LANGUAGE LEVEL von IntelliJ ist auf eine ältere Version gestellt.

Vorausgesetzt, es sind aktuelle Versionen sowohl von Java als auch von IntelliJ installiert, löst die korrekte Konfiguration von FILE • PROJECT STRUCTURE alle Probleme (siehe Abbildung 1.3). Alternativ können Sie den Hello-World-Code auch so formulieren, dass er mit *allen* Java-Versionen der letzten 30 Jahre kompatibel ist. Erklärungen zum Code folgen in Abschnitt 1.4, »Der Hello-World-Code«.

```
// für ältere Java-Versionen
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

## Beispieldateien öffnen

Alle Beispieldateien zu diesem Buch gibt es in zweifacher Ausfertigung: einmal als IntelliJ-Projekte für die Java-Version 25 und einmal als pure \*.java-Dateien, falls Sie statt mit IntelliJ mit einem anderen Editor oder einer anderen IDE arbeiten möchten.

Wenn auf Ihrem Rechner anstelle von Java 25 eine andere Version installiert ist, dann zeigt IntelliJ nach dem Laden eines Beispielprojekts die Fehlermeldung *Project SDK not defined* an. Der Code ist mit roten Wellenlinien unterlegt und kann nicht ausgeführt werden.

Dieses Problem lässt sich leicht lösen: Mit FILE • PROJECT STRUCTURE gelangen Sie in einen Dialog, in dem Sie diverse Projekteinstellungen verändern können (siehe Abbildung 1.3). Dort wählen Sie als PROJECT SDK die auf Ihrem Rechner installierte Java-Version aus. Als PROJECT LANGUAGE LEVEL verwenden Sie dieselbe Version wie beim SDK.

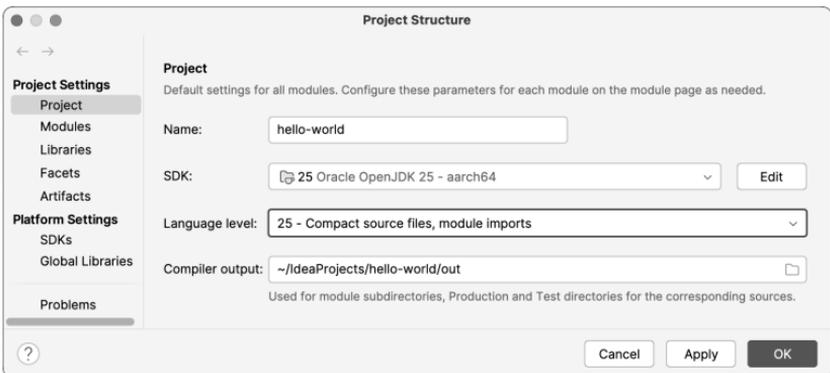


Abbildung 1.3 In den Projekteinstellungen stellen Sie die Java-Version ein.

## 1.4 Der Hello-World-Code

Ich bin Ihnen noch die Erklärung schuldig, was die drei bzw. fünf Zeilen des Hello-World-Codes eigentlich bedeuten. Schließlich geht es nicht an, dass bereits die ersten Codezeilen unbegreiflich sind, oder?

### Vereinfachte Syntax ab Java 25

Beginnen wir mit der vereinfachten Version, die ab Java 25 erlaubt ist:

```
// Hello-World-Code in Java >= 25
void main() {
    IO.println("Hello, World!");
}
```

In Java wird Code in »Methoden« formuliert. Methoden, in anderen Sprachen auch »Funktionen«, enthalten mehrere Anweisungen, die dann ausgeführt werden.

Der Startpunkt für jedes Java-Programm ist die `main`-Methode, die mit `main()` eingeleitet wird. Die geschwungenen Klammern machen klar, welcher Code zur Methode gehört. Das vorangestellte Schlüsselwort `void` bedeutet, dass die Methode kein Ergebnis (keine Zahl, keine Zeichenkette, kein Objekt) zurückgibt.

`println` ist ebenfalls eine Methode. Sie gibt die als Parameter übergebene Zeichenkette, hier "Hello, World!" am Bildschirm aus. Im Gegensatz zu `main` ist `println` eine vordefinierte Methode, steht also immer zur Verfügung. Allerdings gibt es es in Java Tausende von Methoden. Um die Methoden in logische Gruppen zusammenzufassen, sind diese über diverse Pakete und Klassen verteilt.

In Java-Programmen steht das Paket `java.lang` standardmäßig zur Verfügung. Das Paket gilt automatisch als »importiert« und stellt oft benötigte Klassen zur Verfügung. Eine davon hat den Namen `IO` (für *Input/Output*) und stellt die `println`-Methode zur Verfügung. Beim Aufruf von `println` muss der Klassenname `IO` vorangestellt werden. (`IO` ersetzt `System.out` für `print`, `println` und `readln`, aber leider nicht für `format` oder `printf`.)

Noch drei abschließende Bemerkungen zur Java-Syntax:

- ▶ Anweisungen wie `IO.println(...)` enden mit einem Strichpunkt.
- ▶ Einzeilige Kommentare werden mit `\` eingeleitet.
- ▶ Mehrzeilige Kommentare beginnen mit `\*` und enden mit `*/`.

## Die klassische Syntax

Die exakte Beschreibung von »Hello, World!« komplizierter, als Sie vielleicht denken. Erwarten Sie nicht, dass Sie die folgenden Absätze wirklich auf Anhieb verstehen. In ihnen kommen viele Begriffe vor, die ich Ihnen erst im weiteren Verlauf des Buchs in aller Ruhe erkläre.

```
class HelloWorld {
```

Das Schlüsselwort `class` leitet den Code für eine *Klasse* ein. Wenn man es sehr stark vereinfacht, dann bieten Klassen die Möglichkeit, große Projekte in mehrere Komponenten bzw. Codedateien zu zerlegen. Im Gegensatz zu anderen Programmiersprachen ist die Verwendung von Klassen in Java auch bei winzigen Projekten notwendig. (Das gilt auch in Java 25. Aber dort erzeugt der Compiler bei einfachen Programmen selbst eine Klasse, wenn deren Name nicht aus Ihrem Code hervorgeht.)

Hinter `class` wird der Name der Klasse angegeben. Danach folgt der Code, der die Merkmale der Klasse definiert. Damit der Java-Compiler weiß, wo dieser Code beginnt und wo er endet, steht am Anfang die Klammer `{` und am Ende der Klasse die Klammer `}`.

```
public static void main(String[] args) {
```

Der eigentliche Code einer Klasse besteht aus *Methoden*. Diesen Begriff haben wir schon kennengelernt. Die Methode `main` hat drei besondere Eigenschaften. Sie werden durch Schlüsselwörter ausgedrückt, die vor dem Methodennamen stehen:

- ▶ Die Methode ist `public`, also öffentlich bzw. von außen zugänglich und nicht innerhalb der Klasse versteckt.
- ▶ Sie ist `static` (statisch). Das bedeutet, dass die Methode ausgeführt werden kann, ohne dass vorher ein Objekt der Klasse (eine Instanz der

Klasse) erzeugt werden muss. Was das genau bedeutet, lernen Sie in Kapitel 12, »Klassen und Records«.

- Die Methode liefert kein Ergebnis. Darauf deutet das Schlüsselwort `void` hin (wörtlich übersetzt: »nichtig, leer«).

Dem Methodennamen `main` folgt schließlich in runden Klammern eine Liste von Parametern. Parameter bieten einen Mechanismus zum Übertragen von Daten in Methoden. `main` hat genau einen Parameter, den wir `args` genannt haben. `String[]` bedeutet, dass an diesen Parameter mehrere Zeichenketten übergeben werden können (genau genommen ein Array von Zeichenketten). Die Zeichenketten enthalten die Parameter, die beim Start eines Java-Programms angegeben werden können.

Unser Hello-World-Programm wertet den Parameter `args` gar nicht aus. Dennoch muss der Parameter samt dem Datentyp `String[]` angegeben werden! Vergessen Sie das, erkennt der Java-Compiler `main` nicht als Startmethode. Beim Versuch, das Java-Programm auszuführen, tritt dann die Fehlermeldung auf, dass eine korrekt definierte `main`-Methode fehlt.

Wieder ein kurzer Rückblick zur einfachen Variante: Java 25 akzeptiert, anders als die Vorgängerversionen, eine einfachere `main`-Deklaration, bei der Sie sowohl auf `public static` als auch auf den Parameter `args` verzichten. Sie können nun zwar keine Daten an das Programm übergeben, aber beim Hello-World-Programm hatten Sie das ohnedies nie vor.

```
System.out.println("Hello, World!");
```

Die `println`-Methode kennen Sie schon. Methoden werden üblicherweise auf Objekte angewendet. Als *Objekt* gilt in diesem Fall die Standardausgabe, mit der Ausgaben auf den Bildschirm bzw. in das gerade aktive Terminal geleitet werden. Der Zugriff auf das Objekt erfolgt hier durch `System.out`. Dabei bezeichnet `System` den Namen der `System`-Klasse, die ebenfalls durch die Java-Bibliothek vorgegeben ist. `out` ist eine statische Variable dieser Klasse und verweist auf das Standardausgabeobjekt. Dieses Objekt wird von Java automatisch beim Start des Programms erzeugt.

In Java 25 haben sich die Entwicklerinnen und Entwickler dazu entschieden, zusätzlich zu `System.out.println` das vereinfachte `IO.println` zu erlau-

ben. Beide Formen sind gleichwertig, aber die IO-Variante ist leichter zu vermitteln.

Die gesamte Anweisung endet wie alle Java-Anweisungen mit einem Strichpunkt. Diesen zu vergessen zählt zu den häufigsten Fehlern, die Java-Einsteiger und -Einsteigerinnen unweigerlich machen.

```
    }
}
```

Der Programmcode endet schließlich mit zwei geschwungenen Klammern. Die erste gibt an, dass an dieser Stelle die Definition der Methode `main` endet. Die zweite Klammer macht dem Compiler klar, dass nun auch der Code der Klasse zu Ende geht.

### Die reine OO-Lehre

Es gibt zwei Ansätze, den Umgang mit Java zu vermitteln bzw. zu unterrichten. Die erste Variante ist das, was ich salopp als die *reine Lehre der Objektorientierung* (OO) bezeichne. Bücher, die nach diesem Schema aufgebaut sind, beginnen mit einem langen Theorieteil, der die Konzepte der objektorientierten Programmierung erläutert. Erst wenn das erledigt ist, folgen die ersten richtigen Codebeispiele.

Ich bin ein Anhänger der zweiten Variante: Ich setze die drei oder fünf Zeilen des Hello-World-Programms im weiteren Verlauf des Buchs einfach als gottgegeben voraus und führe Sie zuerst in die Grundlagen der Programmierung ein, also z. B. in den Umgang mit Variablen oder die Formulierung von Schleifen. Um die Beispiele der folgenden Kapitel auszuprobieren, verändern Sie nur den Inhalt der `main`-Methode. Wie Klassen und Methoden exakt funktionieren, ist vorerst unwichtig.

Sie werden sehen, dass Sie eine Menge interessante Beispiele entwickeln und noch mehr lernen können, ohne zu wissen, wodurch sich Klassen von Objekten unterscheiden und wann statischen bzw. nichtstatischen Methoden der Vorzug zu geben ist!

Da selbst einfache Java-Programme nicht ohne Objektorientierung auskommen, finden Sie im folgenden Kapitel eine erste, eher allgemein

gehaltene Einführung in die objektorientierte Programmierung (siehe Abschnitt 2.1, »Die Idee des objektorientierten Programmierens«). Alle weiteren Details folgen dann ab Kapitel 12, »Klassen und Records«, in dem ich Ihnen die Syntax der objektorientierten Programmierung im Detail vorstelle und das Schlüsselwort `class` *richtig* erkläre.

## 1.5 Visual Studio Code

Die Entwicklungsumgebung IntelliJ hat einen großen Nachteil: Das Programm bietet unzählige Profifunktionen, die Sie aktuell nicht brauchen.

Die Funktionsfülle macht die Bedienung unübersichtlich und das Programm schwerfällig. Im Unterricht merke ich immer wieder, dass der Umgang mit IntelliJ anfangs mehr Probleme verursacht als Java an sich. Dennoch setze ich IntelliJ beginnend mit der ersten Unterrichtsstunde ein – nicht, weil ich damit restlos glücklich bin, aber weil aus meiner Sicht die Vorteile überwiegen. Oder, anders formuliert: Jede Entwicklungsumgebung, jeder Editor ist ein Kompromiss. IntelliJ ist aus meiner Sicht der beste. (Werfen Sie auch einen Blick in den Anhang, wo ich einige Tipps zur effizienten Bedienung von IntelliJ gesammelt habe!)

Wenn Sie sich partout nicht mit IntelliJ anfreunden möchten, gibt es natürlich Alternativen. Einen ähnlichen Funktionsumfang wie IntelliJ bieten die Entwicklungsumgebungen Eclipse und NetBeans. Damit richten sich auch dieses Programme an das Profisegment und leiden unter den gleichen Problemen wie IntelliJ.

Weniger überladen sind reine Editoren. Insbesondere das von Microsoft entwickelte Programm *Visual Studio Code* (siehe Abbildung 1.4) hat in den letzten Jahren eine riesige Anwendergemeinde gefunden. Grundsätzlich kommt VSCode in Kombination mit dem `EXTENSION PACK FOR JAVA` ausgezeichnet mit Java zurecht. Dabei ist aber entscheidend, dass Sie jedes Java-Programm als eigenständiges Projekt betrachten. Wenn Sie die Beispielprogramme zu diesem Buch mit VSCode ausprobieren möchten, müssen Sie für jedes Beispiel mit `FILE • OPEN FOLDER` das unmittelbare

Verzeichnis des betreffenden Beispiels öffnen, für dieses Kapitel also das Verzeichnis kap01-helloworld.

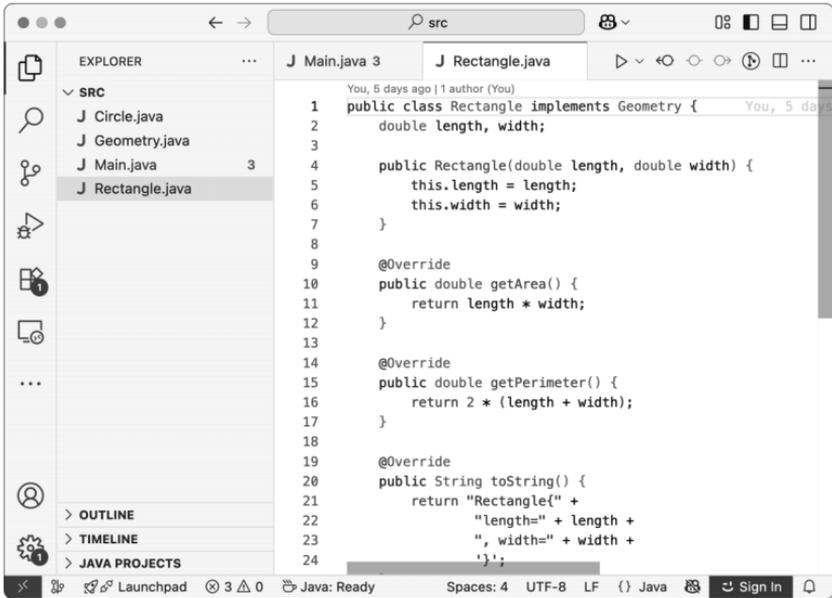


Abbildung 1.4 Visual Studio Code

Viel naheliegender wäre es, das übergeordnete Verzeichnis mit allen Beispielen als VSCode-Folder zu öffnen. Damit könnten Sie rasch von einem Beispiel zum nächsten wechseln und mehrere Dateien parallel öffnen. Leider funktioniert dann die Programmausführung nicht mehr: VSCode glaubt, alle Java-Dateien seien Teil eines riesigen Projekts. Dementsprechend führt VSCode häufig die falsche Datei aus oder zeigt obskure Fehlermeldungen an, dass sich die Datei nicht im *Classpath* befindet und daher nicht gestartet werden kann. Abhilfe: Öffnen Sie mit FILE • OPEN FOLDER direkt das Projektverzeichnis!

# Kapitel 5

## Operatoren

Im Ausdruck  $a = b + c$  gelten die Zeichen  $=$  und  $+$  als Operatoren. Dieses Kapitel stellt Ihnen alle Java-Operatoren vor.

### 5.1 Überblick

Java kennt Operatoren zur Zuweisung von Variablen, zum Vergleich von Werten, zur Berechnung mathematischer Ausdrücke etc. (siehe Tabelle 5.1). Sonderfälle sind  $\rightarrow$  zur Formulierung von Lambda-Ausdrücken sowie  $::$  zur Angabe von Referenzen auf Methoden. Diese Operatoren stelle ich Ihnen in Kapitel 18, »Lambda-Ausdrücke«, näher vor.

Priorität	Operator	Bedeutung
1 $\rightarrow$	()	Methodenaufruf
	[]	Zugriff auf Felder (Arrays)
	.	Zugriff auf Objekte, Methoden etc.
	::	Referenz auf Methoden
2 $\rightarrow$	++ --	Inkrement/Dekrement (Postfix, z. B. $a++$ )
3 $\leftarrow$	++ --	Inkrement/Dekrement (Präfix, z. B. $--b$ )
	+ -	Vorzeichen
	! ~	logisches Nicht, binäres Nicht ( <i>NOT</i> )
	(typ)	explizite Typumwandlung (Casting)
	new	Objekte erzeugen

Tabelle 5.1 Java-Operatoren

Priorität	Operator	Bedeutung
4 →	* / %	Multiplikation, Division, Restwert
5 →	+ - +	Addition, Subtraktion Verbindung von Zeichenketten
6 →	<< >> >>>	Bits nach links/rechts schieben Bits ohne Vorzeichen nach rechts schieben
7 →	> >= < <= instanceof	Vergleich größer bzw. größer-gleich Vergleich kleiner bzw. kleiner-gleich Typenvergleich
8 →	== !=	Vergleich gleich bzw. ungleich
9 →	&	bitweises/logisches Und (AND)
10 →	^	bitweises/logisches Exklusiv-Oder (XOR)
11 →		bitweises/logisches Oder (OR)
12 →	&&	logisches Und ( <i>Short-circuit Evaluation</i> )
13 →		logisches Oder ( <i>Short-circuit Evaluation</i> )
14 ←	a ? b : c	Auswahloperator ( <i>Ternary Operator</i> )
15 ←	= += -= *= /= %= <<= >>= >>>= &=  = ^=	Zuweisung Grundrechenarten und Zuweisung Grundrechenarten und Zuweisung Bit-Shift und Zuweisung Bit-Operationen und Zuweisung

Tabelle 5.1 Java-Operatoren (Forts.)

Die erste Spalte der Operatortabelle gibt die Priorität und Assoziativität an:

- ▶ Die **Priorität** bestimmt, in welcher Reihenfolge ein Ausdruck verarbeitet wird. Beispielsweise wird beim Ausdruck  $a * b + c$  zuerst das Produkt aus  $a$  mal  $b$  berechnet und erst dann  $c$  addiert. Die Operatoren  $*$  und  $/$  haben also eine höhere Priorität als  $+$  und  $-$ .
- ▶ Die **Assoziativität** gibt an, ob gleichwertige Operatoren von links nach rechts oder von rechts nach links verarbeitet werden sollen. Beispielsweise ist  $-$  (Minus) ein linksassoziativer Operator. Die Auswertung erfolgt von links nach rechts.  $17 - 5 - 3$  wird also in der Form  $(17 - 5) - 3$  verarbeitet und ergibt 9. Falsch wäre  $17 - (5 - 3) = 15!$

Der Zuweisungsoperator  $=$  ist dagegen rechtsassoziativ. Beim Ausdruck  $a = b = 3$  wird zuerst  $b=3$  ausgeführt. Das Ergebnis dieser Zuweisung (also 3) wird dann auch  $a$  zugewiesen, sodass schließlich die Variablen  $a$  und  $b$  beide den Wert 3 enthalten.

In den weiteren Abschnitten dieses Kapitels folgen Detailinformationen zu einzelnen Operatoren. Beachten Sie, dass Java keine Möglichkeit bietet, Operatoren neu zu definieren. Einige andere Programmiersprachen kennen dieses Konzept unter dem Namen *Operator Overloading*.

### Tipp

Verwenden Sie im Zweifelsfall Klammern, um die Reihenfolge festzulegen. Das macht Ihren Code klarer und für andere Programmierer und Programmiererinnen lesbarer!

## 5.2 Details und Sonderfälle

Natürlich müssen Sie die Operatortabelle aus dem vorigen Abschnitt nicht auswendig können – zumal sie Operatoren enthält, deren richtige Anwendung Sie erst nach dem Studium der weiteren Kapitel so richtig verstehen werden. Vielmehr soll diese Tabelle auch für die Zukunft als zentrale

Referenz dienen. Die folgenden Abschnitte erläutern den Einsatz einiger Operatoren und weisen auf Sonderfälle hin.

### Zuweisungen

Mit dem Zuweisungsoperator = speichern Sie elementare Daten bzw. Referenzen auf Objekte in Variablen. Die Zuweisung kann wahlweise direkt bei der Deklaration der Variablen oder später erfolgen.

```
int a = 3;
var p = new java.awt.Point(2, 5);
```

Java unterstützt Mehrfachzuweisungen:

```
a = b = 3;      // weist a und b den Wert 3 zu
a = b = c = d; // weist a, b und c den Inhalt von d zu
```

Es ist auch möglich, eine Zuweisung mit einem Vergleich zu kombinieren. Die Zuweisung muss dabei in Klammern gestellt werden! Im folgenden Code wird eine Datei zeilenweise mit der Methode `ReadLine` ausgelesen. Die gerade aktuelle Zeile wird in der Variablen `line` gespeichert. Wenn das Ende der Datei erreicht ist, liefert `ReadLine` den Zustand `null`, und die Schleife wird abgebrochen.

```
String line;
while((line = breader.ReadLine()) != null) {
    ... // Textzeile in line verarbeiten
}
```

### Mathematische Operatoren

Die Anwendung der Operatoren +, -, \* und / für die Grundrechenarten bedarf keiner weiteren Erklärung. Beachten Sie, dass Java automatisch Integerberechnungen durchführt, wenn alle Operanden ganze Zahlen sind, sodass z. B. der Term  $5/3$  den Wert 1 ergibt. Wenn die Berechnung mit Fließkommaarithmetik durchgeführt werden soll, müssen Sie entweder eine Typumwandlung durchführen (z. B. `(double)i`) oder Zahlen explizit als Fließkommazahlen angeben (z. B. `2.0` statt `2`).

Der Restwertoperator % berechnet den Rest einer Division mit einem ganzzahligen Ergebnis. Der Operator kann auch für Fließkommaberechnungen verwendet werden. Im zweiten Beispiel ergibt die Division von  $22.3 / 3.5$  das Ergebnis 6. Der Rest wird mit  $22.3 - 6 * 3.5$  errechnet.

```
int i = 22;
IO.println(i % 5);    // liefert 2
double d = 22.3;
IO.println(d % 3.5); // liefert 1.30000
```

### Hinweis

Bei negativen Argumenten gibt es verschiedene Möglichkeiten, den Rest einer Division auszurechnen. Bei Java, C, C# und vielen anderen Programmiersprachen stimmt das Vorzeichen des Ergebnisses immer mit dem des Dividenden überein.  $-7 \% 3$  liefert somit  $-1$ . Das entspricht aber *nicht* der euklidischen Modulo-Operation! Weitere Details zu diesem Thema können Sie in der Wikipedia nachlesen:

[https://en.wikipedia.org/wiki/Modulo\\_operation](https://en.wikipedia.org/wiki/Modulo_operation)

Die Grundrechenarten können mit einer Zuweisung kombiniert werden, was bei langen Variablennamen den Tipp- und Platzaufwand minimiert:

```
a += 1; // entspricht a = a + 1;
a -= 2; // entspricht a = a - 2;
a *= 3; // entspricht a = a * 3;
a /= 2; // entspricht a = a / 2;
a %= 2; // entspricht a = a % 2;
```

Java kennt keinen Operator zum Potenzieren.  $a^b$  müssen Sie unter Zuhilfenahme der Methode `Math.pow` berechnen:

```
double d = Math.pow(2, 3); // ergibt 8.0
```

In der `Math`-Klasse sind unzählige weitere mathematische Funktionen wie `sqrt`, `sin`, `cos` sowie die Konstante `PI` enthalten.

## Inkrement und Dekrement

Wie C kennt Java die Inkrement- und Dekrement-Operatoren `++` und `--`:

```
a++; // entspricht a = a + 1;
a--; // entspricht a = a - 1;
```

Diese Operatoren können wahlweise nach oder vor dem Variablennamen angegeben werden (Postfix- bzw. Präfix-Notation). Im ersten Fall liefert der Ausdruck den Wert vor der Veränderung, im zweiten Fall den Wert nach der Veränderung. Das gilt gleichermaßen für Zuweisungen wie für Berechnungen.

```
int n = 7;
int a = n++; // a = 7, n = 8
int b = ++n; // b = 9, n = 9
```

Das folgende Beispiel beweist, wie wichtig die richtige Anwendung der Post- und Präfix-Notation ist. Die beiden `while`-Schleifen versuchen, die drei im Array gespeicherten Zeichen auf dem Bildschirm ausgeben. Die erste Schleife ist korrekt, die zweite hingegen nicht! Wegen der Postfix-Notation wird `i` zuerst um 1 erhöht und dann ausgewertet. Deswegen versucht die Schleife, auf ein Array-Element zuzugreifen, das es gar nicht gibt. Die Folge ist ein Fehler. (Details zu Arrays folgen in Kapitel 7, »Arrays und Listen«.)

```
char[] ca = {'a', 'b', 'c'};
int i = 0;
while (i < ca.length)
    // OK, Ausgabe 'a', 'b', 'c'
    IO.println(ca[i++]);

i = 0;
while (i < ca.length)
    // Vorsicht: Ausgabe 'b', 'c',
    // dann ArrayIndexOutOfBoundsException
    IO.println(ca[++i]);
```

## Vergleiche

Zwei Zahlen können Sie mit `==`, `!=`, `<`, `<=`, `>` und `>=` vergleichen (gleich, ungleich, kleiner, kleiner/gleich, größer, größer/gleich). Derartige Vergleiche benötigen wir in Kapitel 6, »Verzweigungen und Schleifen«. Vorweg ein Beispiel:

```
int a = 3, b = 5;
if (a == b) {
    IO.println("a und b sind gleich groß.");
} else {
    IO.println("a und b sind unterschiedlich.");
}
```

Bei Objektvariablen testet `==`, ob beide Variablen auf das gleiche Objekt verweisen. `!=` liefert `true`, wenn die Variablen auf unterschiedliche Objekte zeigen. Der Inhalt der Objekte wird in beiden Fällen nicht berücksichtigt. Details zum Umgang mit Klassen und Objekten folgen in Kapitel 12.

### Vorsicht

Zeichenketten werden wie Objekte behandelt. Um zu testen, ob zwei Zeichenketten übereinstimmen, müssen Sie `s1.equals(s2)` ausführen. Vermeiden Sie unbedingt `s1 == s2`! Dieser Vergleich ist syntaktisch erlaubt, berücksichtigt aber nicht den Inhalt der Zeichenketten. Es ist durchaus möglich, dass zwei Zeichenketten (`String`-Objekte) dieselben Zeichen enthalten, sich aber trotzdem an unterschiedlichen Orten im Speicher befinden.

```
String s1 = "abc";
String s2 = "abc".toUpperCase();
IO.println(s2 == "ABC");           // liefert false
IO.println(s2.equals("ABC"));     // liefert true
```

## Boolesche Ausdrücke (verknüpfte Bedingungen)

Wenn Sie Bedingungen für Schleifen oder Abfragen formulieren, wollen Sie oft mehrere Ausdrücke miteinander verknüpfen. Beispielsweise soll eine bestimmte Reaktion Ihres Programms nur erfolgen, wenn  $a$  größer als drei *und*  $b$  größer als fünf ist:

```
if (a > 3 && b > 5) { ... }
```

Operator	Bedeutung
!	logisches Nicht, binäres Nicht (NOT)
&	logisches Und (AND)
	logisches Oder (OR)
^	logisches Exklusiv-Oder (XOR)
&&	logisches Und ( <i>Short-Circuit Evaluation</i> )
	logisches Oder ( <i>Short-Circuit Evaluation</i> )

**Tabelle 5.2** Verknüpfung von Bedingungen

Java sieht für die Verknüpfung mehrerer Bedingungen eine ganze Palette boolescher Operatoren vor (siehe Tabelle 5.2). Sollten Sie noch nie mit logischen Operatoren zu tun gehabt haben, folgt hier eine kurze Erklärung:

- ▶ **Logisches Nicht:** Entspricht einer Inversion. Aus `true` wird `false` und umgekehrt.
- ▶ **Logisches Und:** Bei `a & b` müssen sowohl `a` als auch `b` den Wert `true` haben, damit das Ergebnis `true` lautet.
- ▶ **Logisches Oder:** Bei `a | b` reicht es aus, wenn `a` oder `b`, also zumindest ein Ausdruck, den Wert `true` hat, damit das Ergebnis `true` lautet.
- ▶ **Logisches Exklusiv-Oder:** Für `a ^ b` gilt ein strenges Entweder-oder: Damit das Ergebnis `true` lautet, muss *genau ein* Ausdruck `true` und der andere `false` sein.

Besonders interessant sind die Und- bzw. Oder-Varianten `&&` und `||`, die eine sogenannte *Short-Circuit Evaluation* unterstützen; dabei wird die Auswertung abgebrochen, sobald das Ergebnis feststeht:

- ▶ Bei `&&` endet die Auswertung beim ersten `false`-Teilergebnis. Das Gesamtergebnis ist dann zwingend `false`.
- ▶ Bei `||` endet die Auswertung beim ersten `true`-Teilergebnis. In diesem Fall ist das Gesamtergebnis zwingend `true`.

Beispielsweise spielt der Inhalt von `b` im folgenden Beispiel überhaupt keine Rolle. Da `a` nicht zutrifft, wird `b` gar nicht mehr ausgewertet.

```
boolean a = false;
boolean b = true;
if (a && b) { ... }
```

Wenn `a` und `b` nicht einfach boolesche Variablen sind, sondern wenn an dieser Stelle komplexe Ausdrücke oder zeitaufwendige Methodenaufrufe stehen, dann spart die *Short-Circuit Evaluation* Rechenzeit. Zugleich vermeidet sie die Auswertung von Teilausdrücken, die Fehler verursachen können. Im folgenden Beispiel wird die Division `a/b` nur durchgeführt, wenn `b` ungleich null ist:

```
int a = 3, b = 5;
if (b != 0 && a / b > 3) { ... }
```

## Rechnen mit Bits

Die logischen Operatoren `&`, `|`, `^` und `~` (AND, OR, XOR und NOT) können nicht nur für boolesche Ausdrücke verwendet werden, sondern auch für ganze Zahlen. In diesem Fall gilt ihre Wirkung jeweils bitweise. Wenn `a` und `b` zwei ganze Zahlen sind, dann wendet Java in `a & b` die Und-Logik für alle Bits von `a` und `b` an:

```
int a = 0b11100;           // 28 = 0b11100
int b = 0b01111;           // 15 = 0b01111
System.out.println(a & b); // 12 = 0b01100
```

>> verschiebt die Bits einer Zahl um  $n$  Bits nach rechts (entspricht einer Division durch  $2^n$ ), << verschiebt entsprechend nach links (entspricht einer Multiplikation mit  $2^n$ ). >>> funktioniert wie >>, betrachtet die Zahl aber, als wäre sie vorzeichenlos.

```
int a = 16;
int b = a << 2; // entspricht b = a * 4, Ergebnis b = 64
int c = a >> 1; // entspricht c = a / 2, Ergebnis 8
```

### Sonstige Operatoren

Beim Ausdruck  $a ? b : c$  testet Java, ob  $a$  zutrifft (true ergibt). Ist das der Fall, lautet das Ergebnis  $b$ , sonst  $c$ . Im Detail stelle ich Ihnen diesen Operator in Kapitel 6, »Verzweigungen und Schleifen«, vor.

```
int x = 1, y = 2, result;
result = (x < y) ? x : y; // result enthält jetzt die
                        // kleinere der beiden Zahlen
```

Mit `new` erzeugen Sie neue Instanzen (Objekte) einer Klasse. `instanceof` ermöglicht einen Test, ob das Objekt eine Instanz der angegebenen Klasse ist. Wenn `javac` feststellt, dass die Objektvariable sicher kein Objekt der angegebenen Klasse enthält, kann der Code nicht kompiliert werden. Mehr Details zu `new` und `instanceof` folgen in Kapitel 12, »Klassen und Records«.

## 5.3 Wiederholungsfragen

- ▶ **W1:** Sie wollen den Rest der Division  $225 / 17$  ermitteln. Wie gehen Sie vor?
- ▶ **W2:** Welchen Wert haben  $a$ ,  $b$ ,  $c$  und  $d$ ?

```
int a = 7, b = 12, c = 20;
int d = a---b---c;
```
- ▶ **W3:** Was ist die *Short-Circuit Evaluation*? Nennen Sie ein Beispiel!

# Anhang B

## Lösungen

Dieses Kapitel fasst die Lösungen zu den Wiederholungsfragen und Übungen zusammen. Beachten Sie, dass es sich bei Codelösungen immer um Lösungsvorschläge handelt. Zu fast allen Aufgabenstellungen gibt es viele Lösungswege.

### B.1 Kapitel 2, »Java-Crashkurs«

#### W1: Methoden

Methoden sind in sich abgeschlossene Codeeinheiten. Methoden erfüllen Aufgaben, führen z. B. eine Ausgabe oder eine Berechnung durch. Methoden werden meist auf dazugehörige Daten (»Objekte«) angewendet.

#### W2: Klassen versus Objekte

Klassen bilden die Infrastruktur für die Arbeit mit einer bestimmten Art von Daten. Objekte sind konkrete Daten.

Wenn Sie möchten, können Sie eine Klasse als Bauplan eines Hauses betrachten. Mit diesem Bauplan können Sie dann mehrere Häuser bauen. Das sind die Objekte.

#### W3: Kommentare

// leitet einen Kommentar ein, der bis zum Zeilenende reicht.

/\* leitet einen mehrzeiligen Kommentar ein, der mit \*/ endet.

\*\* leitet einen Javadoc-Kommentar ein. Auch dieser Kommentar endet mit \*/.

#### W4: Aufruf von Methoden

Methoden werden in der Form `objektvariable.methode()` aufgerufen. Bei vielen Methoden können bzw. müssen zwischen den runden Klammern Parameter angegeben werden.

Es gibt auch statische Methoden, deren Aufruf kein Objekt erfordert. In diesem Fall erfolgt der Aufruf in der Form `Klassenname.methode()`.

#### W5: Strichpunkte

Strichpunkte trennen Java-Anweisungen voneinander. Jede Anweisung muss mit einem Strichpunkt enden:

```
objekt1.methodeA();  
objekt2.methodeB();
```

*Keine* Strichpunkte folgen hingegen nach Java-Konstrukten wie Klassen, Schleifen, Verzweigungen etc. Hierbei werden Bedingungen in runde Klammern gestellt. Der Anfang und das Ende des nachfolgenden Codeblocks werden durch geschwungene Klammern markiert.

```
if (i > 3) { // kein Strichpunkt nach if ()  
    anweisung1;  
    anweisung2;  
} // auch hier kein Strichpunkt!
```

#### W6: Groß- und Kleinschreibung

In Java ist es üblich, dass Namen von Klassen bzw. Typen mit einem Großbuchstaben beginnen, Namen von Methoden, Feldern und Variablen hingegen mit einem Kleinbuchstaben.

#### W7: import

`import java.util.Arrays` bedeutet, dass in der Codedatei die `Array`-Klasse verwendet werden kann, ohne jedes Mal den Paketnamen `java.util` voranzustellen.

## W8: import-Vereinfachungen ab Java 25

Für Java-Dateien ohne implizite Klassendeklaration gelten alle Klassen aus `java.lang`, `java.util`, `java.io`, `java.nio`, `java.net` und `java.time` automatisch als importiert.

Bei »gewöhnlichen« Java-Dateien können mit `import module name` alle im Modul deklarierten Pakete auf einmal importiert werden. Besonders hilfreich ist `import module java.base` für die gerade aufgezählten Pakete.

## B.2 Kapitel 4, »Variablenverwaltung«

### W1: Lebensdauer von Variablen

Der Inhalt von Variablen bleibt maximal so lange erhalten, wie ein Java-Programm läuft. Viele Variablen haben sogar eine viel kürzere Lebensdauer. Sie können nur genutzt werden, solange ein Objekt existiert, d. h., solange Ihr Programm durch eine Variable auf das Objekt verweist.

Wenn Sie Daten dauerhaft speichern möchten, müssen Sie sie in Dateien speichern oder in einer Datenbank ablegen und später von dort wieder lesen.

### W2: Datentypen für ganze Zahlen

Wenn Sie in einer Variablen ganze Zahlen zwischen 0 und 1.000 speichern möchten, würde ich als Datentyp `int` empfehlen. Prinzipiell käme auch `short` in Frage (zulässiger Wertebereich von  $-32.768$  bis  $+32.767$ ). `short` bietet aber nur dann Vorteile im Vergleich zu `int`, wenn sehr viele gleichartige Daten gespeichert werden sollen – z. B. in einem Array mit 100.000 Elementen. In diesem Fall würden Sie mit `short` 200.000 Byte Arbeitsspeicher sparen.

### W3: Fließkommadivision durch 0

Java führt eine Fließkommadivision durch 0 ohne Fehler aus und liefert im folgenden Beispiel den Wert Infinity:

```
double x = 2, y = 0;  
IO.println(x / y);
```

#### W4: Variablen müssen initialisiert werden

Das folgende Programm kann nicht kompiliert werden, weil versucht wird, die Variable `z` zu verwenden, bevor sie initialisiert wird. Die javac-Fehlermeldung lautet *variable z might not have been initialized*.

```
int x, y, z;  
x = 3;  
y = x + z;  
z = 5;  
IO.println("x=" + x + " y=" + y + " z=" + z);
```

#### W5: Explizites Casting

Das folgende Programm kann nicht kompiliert werden, weil versucht wird, das Ergebnis einer `int`-Addition in einer `short`-Variablen zu speichern. Dabei könnte es zu einem Datenverlust kommen.

```
short s = 3;  
int i = 4;  
s = s + i;  
IO.println(s);
```

Um das Problem zu beheben, muss der Code so umformuliert werden:

```
s = (short)(s + i);
```

#### W6: Literale

Hexadezimalen Zahlen stellen Sie `0x` voran, binären `0b`:

```
int i1 = 0xAA00;  
int i2 = 0b10101111;
```

### W7: Konstanten

Java kennt zwar nicht direkt Konstanten, kann aber durch `final` verhindern, dass der Inhalt einer Variablen für einen elementaren Datentyp später nochmals verändert wird. Es ist üblich, solche Variablen mit Großbuchstaben zu definieren:

```
final double E = 2.71828182845904;
```

### W8: Quadrat ausrechnen

Eine mögliche Lösung finden Sie in den Beispieldateien zu diesem Buch im Projekt `loesungen-kap04-quadrat`.

## B.3 Kapitel 5, »Operatoren«

### W1: Restwertoperator

Den Rest der Division  $225 / 17$  ermitteln Sie mit dem `%`-Operator:

```
IO.println(225 % 17); // Ergebnis 4
```

### W2: Postfix versus Präfix

```
int a = 7, b = 12, c = 20;  
int d = a---b---c;
```

Java verarbeitet die Zuweisung an `d` so:

```
d = (a--) - (b--) - c;
```

Da `--` hier in der Postfix-Notation angewendet wird, berücksichtigt Java die ursprünglichen Inhalte von `a` und `b`, also:

```
d = 7 - 12 - 20;
```

Nach der Berechnung des Werts für `d` werden auch die Variablen `a` und `b` geändert. Somit ergibt sich: `a=6, b=11, c=20` und `d=-25`.

**W3: Short-circuit Evaluation**

Die logischen Operatoren `&&` und `||` verzichten auf die Auswertung des zweiten Operanden, wenn der erste Operand bereits zum Ergebnis führt. Wenn im folgenden Beispiel `myfunc(x)` den Wert 0 oder eine negative Zahl liefert, dann wird `myfunc(y)` nicht aufgerufen. Das ist nicht notwendig, weil `&&` nur dann `true` liefern kann, wenn beide Teilergebnisse `true` sind.

```
double x = 2, y = 3;
if (myfunc(x) > 0 && myfunc(y) > 0) {
    // Code
}
```

**B.4 Kapitel 6, »Verzweigungen und Schleifen«****W1: Schaltjahrtest**

Eine der vielen möglichen Lösungen sieht so aus:

```
// Projekt loesungen-kap06-schaltjahr-schleife
int year = 2023;
boolean leapyear;
if (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0))
    leapyear = true;
else
    leapyear = false;
```

Dabei wird zuerst getestet, ob die Jahreszahl durch vier ohne Rest teilbar ist. Ist das nicht der Fall, kann es kein Schaltjahr sein; das Ergebnis ist sofort `false`. Andernfalls muss eine von zwei Zusatzbedingungen erfüllt sein: Das Jahr ist nicht durch 100 teilbar, oder es ist durch 400 teilbar.

**W2: Schleife für Schaltjahrtest**

```
// Projekt loesungen-kap06-schaltjahr-schleife
for (year = 1999; year <= 2030; year++) {
    ...
}
```

# Auf einen Blick

1	Hello, World! .....	17
2	Java-Crashkurs .....	32
3	Java lernen mit KI-Unterstützung .....	44
4	Variablenverwaltung .....	52
5	Operatoren .....	77
6	Verzweigungen und Schleifen .....	87
7	Arrays und Listen .....	111
8	Zeichenketten .....	124
9	Methoden .....	145
10	Datum und Uhrzeit .....	172
11	Exceptions .....	191
12	Klassen und Records .....	204
13	Listen, Sets und Maps .....	234
14	Vererbung .....	260
15	Schnittstellen .....	288
16	Geschachtelte und versiegelte Klassen .....	298
17	Generische Klassen und Methoden .....	310
18	Lambda-Ausdrücke .....	319
19	Dateien und Verzeichnisse .....	339
20	JavaFX .....	360
21	Java-Interna und -Bibliotheken .....	382
22	Eine REST-API anbieten und nutzen .....	406

---

# Inhalt

Vorwort .....	13
<b>1 Hello, World!</b>	<b>17</b>
<hr/>	
1.1 Einführung .....	17
1.2 Java installieren .....	18
1.3 IntelliJ IDEA .....	21
1.4 Der Hello-World-Code .....	26
1.5 Visual Studio Code .....	30
<b>2 Java-Crashkurs</b>	<b>32</b>
<hr/>	
2.1 Die Idee des objektorientierten Programmierens .....	32
2.2 Java-Syntax .....	37
2.3 Wiederholungsfragen .....	42
<b>3 Java lernen mit KI-Unterstützung</b>	<b>44</b>
<hr/>	
3.1 Der richtige Einsatz von KI-Tools .....	45
3.2 Mein Programm funktioniert nicht! Warum? .....	46
3.3 Ich möchte diese Aufgabe lösen. Hilf mir! .....	49
3.4 Sprachkonzepte erklären .....	51

<b>4</b>	<b>Variablenverwaltung</b>	<b>52</b>
4.1	Variablen .....	52
4.2	Elementare Datentypen .....	56
4.3	Literale .....	63
4.4	Variablen einlesen und ausgeben .....	65
4.5	Variablen im größeren Java-Kontext .....	68
4.6	Konstanten und Enums .....	74
4.7	Wiederholungsfragen und Übungen .....	76
<b>5</b>	<b>Operatoren</b>	<b>77</b>
5.1	Überblick .....	77
5.2	Details und Sonderfälle .....	79
5.3	Wiederholungsfragen .....	86
<b>6</b>	<b>Verzweigungen und Schleifen</b>	<b>87</b>
6.1	if-Verzweigungen .....	88
6.2	if-Kurzschreibweise (ternärer Operator) .....	93
6.3	switch-Verzweigungen .....	94
6.4	for-Schleifen .....	97
6.5	for-each-Schleifen .....	104
6.6	while- und do-while-Schleifen .....	105
6.7	break und continue .....	107
6.8	Wiederholungsfragen und Übungen .....	109
<b>7</b>	<b>Arrays und Listen</b>	<b>111</b>
7.1	Syntax .....	111
7.2	Mit Arrays arbeiten .....	117

---

7.3	Listen .....	121
7.4	Wiederholungsfragen und Übungen .....	123
<b>8</b>	<b>Zeichenketten</b> .....	<b>124</b>

---

8.1	Der Datentyp char .....	124
8.2	Die String-Klasse .....	126
8.3	Formatierung und Konvertierung .....	134
8.4	Die StringBuilder-Klasse .....	140
8.5	Beispiele .....	141
8.6	Wiederholungsfragen und Übungen .....	144
<b>9</b>	<b>Methoden</b> .....	<b>145</b>

---

9.1	Einführung .....	146
9.2	Parameterliste .....	149
9.3	Rückgabewert und return .....	154
9.4	Rekursion .....	157
9.5	Tipps zum richtigen Einsatz von Methoden .....	158
9.6	Beispiele .....	159
9.7	Wiederholungsfragen und Übungen .....	169
<b>10</b>	<b>Datum und Uhrzeit</b> .....	<b>172</b>

---

10.1	Datum und Zeit seit Java 8 .....	173
10.2	Veraltete Datums- und Zeitklassen (Date, Calendar) .....	182
10.3	Wiederholungsfragen und Übungen .....	190

<b>11</b>	<b>Exceptions</b>	<b>191</b>
<hr/>		
11.1	Exception-Klassen .....	192
11.2	try-catch .....	194
11.3	Fehleranfällige Methoden deklarieren (throws) .....	198
11.4	Selbst Exceptions auslösen (throw) .....	199
11.5	Beispiel .....	200
11.6	Wiederholungsfragen und Übungen .....	202
<b>12</b>	<b>Klassen und Records</b>	<b>204</b>
<hr/>		
12.1	Eigene Klassen entwickeln .....	205
12.2	Der Konstruktor .....	215
12.3	get- und set-Methoden (Getter/Setter) .....	219
12.4	Records .....	222
12.5	UML-Diagramme .....	226
12.6	Beispiel: Bücher und Kapitel .....	227
12.7	Wiederholungsfragen und Übungen .....	231
<b>13</b>	<b>Listen, Sets und Maps</b>	<b>234</b>
<hr/>		
13.1	Interna: Schnittstellen und generische Klassen .....	235
13.2	Listen .....	238
13.3	Sets .....	240
13.4	Maps .....	243
13.5	Beispiel: Polygone .....	245
13.6	Beispiel: Schachfigur Springer .....	249
13.7	Beispiel: Mitarbeiter verwalten .....	255
13.8	Wiederholungsfragen und Übungen .....	259

---

<b>14</b>	<b>Vererbung</b>	260
<hr/>		
14.1	Grundlagen .....	260
14.2	Beispiel: Klassen für Fahrräder .....	265
14.3	Finale und abstrakte Klassen .....	269
14.4	Die Object-Klasse .....	276
14.5	Beispiel: Schachfiguren .....	281
14.6	Wiederholungsfragen und Übungen .....	285
<b>15</b>	<b>Schnittstellen</b>	288
<hr/>		
15.1	interface und implements .....	288
15.2	Beispiel: Geometrische Figuren .....	293
15.3	Wiederholungsfragen und Übungen .....	297
<b>16</b>	<b>Geschachtelte und versiegelte Klassen</b>	298
<hr/>		
16.1	Geschachtelte Klassen .....	299
16.2	Anonyme Klassen .....	301
16.3	Versiegelte Klassen und Schnittstellen .....	307
<b>17</b>	<b>Generische Klassen und Methoden</b>	310
<hr/>		
17.1	Einführung .....	310
17.2	Deklaration generischer Klassen und Schnittstellen .....	312
17.3	Deklaration generischer Methoden .....	314
17.4	Wildcards .....	315
17.5	Wiederholungsfragen .....	318

<b>18</b>	<b>Lambda-Ausdrücke</b>	<b>319</b>
<hr/>		
18.1	Hello, Lambda World! .....	319
18.2	Lambda-Syntax .....	321
18.3	Generische Lambda-Schnittstellen .....	326
18.4	Funktionale Programmierung mit Streams .....	329
18.5	Beispiel: Mitarbeiter sortieren und auswählen .....	335
18.6	Wiederholungsaufgaben .....	337
<b>19</b>	<b>Dateien und Verzeichnisse</b>	<b>339</b>
<hr/>		
19.1	Klassen- und Schnittstellenüberblick .....	339
19.2	Dateien und Verzeichnisse ergründen .....	341
19.3	Dateien und Verzeichnisse bearbeiten .....	348
19.4	Textdateien lesen und schreiben .....	351
19.5	Beispiel: Evaluierungsdaten auswerten .....	354
19.6	Wiederholungsaufgaben und Übungen .....	358
<b>20</b>	<b>JavaFX</b>	<b>360</b>
<hr/>		
20.1	Hello, JavaFX! .....	360
20.2	Arbeiten mit Steuerelementen .....	366
20.3	Grafikprogrammierung .....	374
20.4	Wiederholungsaufgaben und Übungen .....	380
<b>21</b>	<b>Java-Interna und -Bibliotheken</b>	<b>382</b>
<hr/>		
21.1	Java-Compiler und -Interpreter .....	383
21.2	Der args-Parameter .....	389
21.3	Die Java-Shell .....	391

---

21.4	import .....	394
21.5	Pakete .....	396
21.6	Bibliotheken .....	398
21.7	Module (Jigsaw) .....	403
21.8	Wiederholungsfragen .....	405
<b>22</b>	<b>Eine REST-API anbieten und nutzen</b> .....	<b>406</b>

---

22.1	Hello, Javalin! .....	407
22.2	REST-Server zur Aufgabenverwaltung .....	411
22.3	REST-Client zur Aufgabenverwaltung .....	418

<b>A</b>	<b>Crashkurs IntelliJ IDEA</b> .....	<b>425</b>
----------	--------------------------------------	------------

---

A.1	Benutzeroberfläche .....	425
A.2	Projekte .....	432
A.3	Einstellungen .....	433

<b>B</b>	<b>Lösungen</b> .....	<b>436</b>
----------	-----------------------	------------

---

B.1	Kapitel 2, »Java-Crashkurs« .....	436
B.2	Kapitel 4, »Variablenverwaltung« .....	438
B.3	Kapitel 5, »Operatoren« .....	440
B.4	Kapitel 6, »Verzweigungen und Schleifen« .....	441
B.5	Kapitel 7, »Arrays und Listen« .....	443
B.6	Kapitel 8, »Zeichenketten« .....	445
B.7	Kapitel 9, »Methoden« .....	446
B.8	Kapitel 10, »Datum und Uhrzeit« .....	449
B.9	Kapitel 11, »Exceptions« .....	450
B.10	Kapitel 12, »Klassen und Records« .....	452

B.11	Kapitel 13, »Listen, Sets und Maps« .....	456
B.12	Kapitel 14, »Vererbung« .....	459
B.13	Kapitel 15, »Schnittstellen« .....	461
B.14	Kapitel 17, »Generische Klassen und Methoden« .....	462
B.15	Kapitel 18, »Lambda-Ausdrücke« .....	462
B.16	Kapitel 19, »Dateien und Verzeichnisse« .....	465
B.17	Kapitel 20, »JavaFX« .....	467
B.18	Kapitel 21, »Java-Interna und -Bibliotheken« .....	469
	Index .....	471

# Index

## A

---

Abgeleitete Klasse ..... 261  
abstract (Schlüsselwort) ..... 271  
Abstrakte Klasse ..... 271  
    *versus Schnittstelle* ..... 292  
add (Methode) ..... 121, 238  
addAll (Methode) ..... 238  
Adoptium ..... 20  
Aktuelles Datum ..... 176  
Aktuelles Verzeichnis ermitteln 342  
Annotation ..... 154, 261, 262  
Anonyme Klasse ..... 301  
    *Beispiel* ..... 301, 320  
    *Syntax* ..... 305  
APPEND (Konstante) ..... 351  
append (Methode) ..... 140  
Application (Klasse) ..... 364  
args (Parameter) ..... 389  
Array ..... 111  
    *assoziatives* ..... 243  
    *Duplikatstest* ..... 155, 444  
    *duplizieren* ..... 119  
    *Elementzugriff* ..... 113  
    *initialisieren* ..... 111, 119  
    *Interna* ..... 116  
    *Klasse* ..... 117  
    *mehrdimensionales* ..... 113  
    *Methoden* ..... 117  
    *Mustervergleich* ..... 164  
ArrayList (Klasse) ..... 121, 238  
Assoziatives Array ..... 243  
Assoziativität ..... 79  
Aufgabenverwaltung (Beispiel) 411

Aufzählung ..... 234  
Ausnahme (Exception) ..... 191  
AutoCloseable (Schnittstelle) .... 196  
average (Methode) ..... 330  
AWT ..... 360

## B

---

Basisklasse ..... 261  
Bedingungen ..... 84  
Beispieldateien ..... 20  
between (Methode) ..... 179, 180  
Bibliothek ..... 398  
binarySearch (Methode) .... 117, 239  
Binäre Zahl ..... 64  
Bitweises Rechnen ..... 85  
boolean (Datentyp) ..... 59  
Boolean (Klasse) ..... 72  
Boolescher Ausdruck ..... 84  
Boolescher Wert ..... 59  
BorderPane (Klasse) ..... 368  
Bounding Box ..... 245  
Boxing/Unboxing ..... 72  
break (Schlüsselwort) ..... 94, 107  
Breakpoint ..... 428  
BufferedReader (Klasse) ..... 353  
BufferedWriter (Klasse) ..... 351  
Build System ..... 360  
Button (Klasse) ..... 364  
byte (Datentyp) ..... 57  
Byte (Klasse) ..... 72

## C

Calendar (Klasse) .....	187
Canvas (Klasse) .....	374
case (switch) .....	94
Casting .....	60, 274
<i>bei Klassen</i> .....	273
<i>explizit/implizit</i> .....	60
catch (Schlüsselwort) .....	194
char (Datentyp) .....	124
Character (Klasse) .....	72, 125
charAt (Methode) .....	131
ChatGPT .....	44
CheckBox (Klasse) .....	368
ChronoUnit (Klasse) .....	180
Class (Klasse) .....	277
class (Schlüsselwort) .....	205
clear (Methode) .....	238
clone (Methode) .....	119, 276
close (Methode) .....	67
<i>IO-Klassen</i> .....	341
Closure .....	319
Collator (Klasse) .....	130
collect (Methode) .....	330
Collections .....	234, 239
ComboBox (Klasse) .....	372
Comparable (Schnittstelle) .....	293
<i>Beispiel</i> .....	293, 335
Comparator (Schnittstelle) .....	336
compare (Methode) .....	294
compareTo (Methode) .....	293
comparingInt (Methode) .....	336
Compiler .....	387
Consumer (Schnittstelle) .....	326
containsKey (Methode) .....	243
containsValue (Methode) .....	243
copy (Methode) .....	348
copyOfRange (Methode) .....	117
cos (Methode) .....	58

CREATE (Konstante) .....	351
createDirectories (Methode) .....	348
createDirectory (Methode) .....	348
createFile (Methode) .....	348
createTempDirectory .....	348
CSV-Datei verarbeiten .....	354
curl (Kommando) .....	410

## D

Dangling else .....	89
Data Transfer Object .....	226
Date (Klasse) .....	182
Datei .....	339
<i>Eigenschaften ermitteln</i> .....	345
<i>Existenz testen</i> .....	344
<i>kopieren</i> .....	348
<i>löschen</i> .....	348
<i>suchen</i> .....	346
<i>temporäre Dateien</i> .....	348
<i>Textdateien lesen</i> .....	353
<i>Textdateien schreiben</i> .....	351
<i>verschieben</i> .....	348
Datentypen .....	56
<i>elementare</i> .....	56
DateTimeFormatter .....	176, 178
Datum .....	172, 176
<i>aktuelles</i> .....	176
<i>in Zeichenkette umwandeln</i> ...	178
Debugging (IntelliJ) .....	428
default (Schlüsselwort) .....	289
default (switch) .....	94
Defaultpaket .....	398
Dekrement (Operator) .....	82
delete (Methode) .....	348
deleteIfExists (Methode) .....	348
Deprecated (Annotation) .....	262
Destructuring .....	308
Destruktor .....	218

Diamond-Operator .....	312
Dictionary .....	243
DirectoryStream (Schnittstelle) .....	346
distinct (Methode) .....	330
Division durch Null .....	57, 58
Doppelgängertest .....	155, 444
double (Datentyp) .....	57
<i>Schleifen</i> .....	101, 443
Double (Klasse) .....	72
Downcast .....	274
Duplikat erkennen .....	155, 444
Duration (Klasse) .....	179, 181
Durchschnitt (Arrays) .....	119

## E

---

Effectively-final-Fehlermeldung .....	334
Einrückung von Code .....	91
Elementare Datentypen .....	56
else (Schlüsselwort) .....	88
Elternklasse .....	261
Endlosschleife .....	109
<i>abbrechen</i> .....	100
endsWith (Methode) .....	131
entrySet (Methode) .....	244
enum (Schlüsselwort) .....	75
Enumeration .....	75
equals (Methode) .....	117, 128, 277
Error (Klasse) .....	193
Event (Klasse) .....	372
EventHandler (Schnittstelle) .....	364
Exception .....	191
Exemplar .....	34
exists (Methode) .....	344
exit (Methode) .....	364
Explizites Casting .....	60
exports (Module) .....	403
extends (Schlüsselwort)	
<i>generische Klassen</i> .....	313

## F

---

Fakultät	
<i>Beispiel</i> .....	442
<i>mit Rekursion</i> .....	157
<i>ohne Rekursion</i> .....	109
Fallunterscheidung .....	94
Fehler	
<i>absichern</i> .....	194
<i>Exception</i> .....	191
<i>IOException</i> .....	340
Fibonacci-Zahlen .....	171
Field .....	36
<i>Initialisierung</i> .....	210
<i>statisches</i> .....	213
FIFO (First In – First Out) .....	240
FilenameFilter .....	301, 322
Files (Klasse) .....	344
FileTime (Klasse) .....	345
fill (Methode) .....	117
fillXxx (JavaFX-Methoden) .....	374
filter (Methode) .....	330
<i>Beispiel</i> .....	336
final (Schlüsselwort) .....	270
<i>effectively final</i> .....	305
<i>Lambda-Ausdrücke</i> .....	334
<i>Parameter</i> .....	151
<i>Variablen</i> .....	74
finally (Schlüsselwort) .....	194
Flexible Constructor Bodies .....	264
Fließkommazahl .....	57
<i>lokalisieren</i> .....	138
<i>parse (Methode)</i> .....	138
float (Datentyp) .....	57
Float (Klasse) .....	72
Font (Klasse) .....	372
for-Schleife .....	97
<i>Arrays</i> .....	113
<i>Beispiele</i> .....	101

for-Schleife (Forts.)	
<i>break und continue</i> .....	107
<i>Endlosschleifen</i> .....	109
<i>Fließkommazahlen</i> .....	101
<i>for-each-Variante</i> .....	104
forEach (Methode) .....	319
<i>Beispiel</i> .....	324
format (Methode) .....	135, 176, 178
<i>Datum und Uhrzeit</i> .....	184
formatted (Methode) .....	135
FunctionalInterface .....	289
Funktionale Programmierung	329
Funktionale Schnittstelle .....	289
<i>Lambda-Ausdrücke</i> .....	319

## G

---

Ganze Zahl .....	57
Garbage Collector .....	388
Generalisierung .....	272
<i>Schnittstellen</i> .....	292
Generische Klasse .....	310
<i>anwenden</i> .....	235, 236
Generische Methode .....	314
Generische Schnittstelle ....	310, 326
Geometry (Schnittstelle) .....	293
Geschachtelte Klasse .....	298, 299
Geschwungene Klammern .....	89
get (Methode) .....	219, 238, 243, 342
<i>Javalin</i> .....	409
getAsDouble (Methode) .....	331
getClass (Methode) .....	277
getDayOfMonth (Methode) .....	176
getFileName (Methode) .....	343
getGraphicsContext2D .....	374
getInstance (Methode) .....	184
getLastModifiedTime .....	345
getMonthValue (Methode) .....	176
getOwner (Methode) .....	345

getParent (Methode) .....	343
getProperty (Methode) .....	342
getSource (Methode) .....	372
getTime (Methode) .....	183
getValue (Methode) .....	244
getYear (Methode) .....	176
Gradle .....	360, 400
Grafikprogrammierung .....	374
GraphicContext (Klasse) .....	374
GridPane (Klasse) .....	468
Grundrechenarten .....	80
Gültigkeitsebene .....	69
Gutter Icons (IntelliJ) .....	434, 435

## H

---

handle (Methode) .....	364
hashCode (Methode) .....	277
HashMap (Klasse) .....	243
HashSet (Klasse) .....	240
HBox (Klasse) .....	368
Heimatverzeichnis ermitteln ...	342
Hello, World! .....	26
<i>Generics</i> .....	310
<i>JavaFX</i> .....	360
<i>Javalin</i> .....	407
<i>Lambda</i> .....	319
Hexadezimale Zahl	
<i>Ausgabe</i> .....	64
<i>parse (Methode)</i> .....	137
Hierarchische Datenstrukturen	255
HTTP-Requests .....	406
HttpClient (Klasse) .....	420
Humane Time Line .....	173

## I

---

if-Verzweigung .....	88
<i>Kurzschreibweise</i> .....	93

- 
- Immutable
    - Parameter in Methoden* ..... 151
    - String (Klasse)* ..... 128
  - implements (Schlüsselwort) ..... 291
  - Implizites Casting ..... 60
  - import (Schlüsselwort) ..... 40, 394
    - Java 25* ..... 42
    - statischer Import* ..... 396
  - in (Field) ..... 66
  - indexOf (Methode) ..... 131
  - Inkrement (Operator) ..... 82
  - Inlay Hints (IntelliJ) ..... 434
  - Inner Class ..... 298
  - Input/Output (IO) ..... 339
  - instanceof (Schlüsselwort) ..... 273
    - Pattern Matching* ..... 273
  - Instant (Klasse) ..... 181
  - Instanz ..... 34
  - Instanzmethode ..... 148
  - Instanzvariable ..... 207
    - Initialisierung* ..... 210
  - int (Datentyp) ..... 57
  - Integer (Klasse) ..... 72
  - IntelliJ IDEA ..... 21, 425
    - Debugging* ..... 428
    - Einstellungen* ..... 433
    - externe Bibliotheken* ..... 400
    - Projekte* ..... 432
    - Tastenkürzel* ..... 431
  - interface (Schlüsselwort) ..... 288
  - Interpreter ..... 387
  - Invoke-RequestMethod ..... 410
  - IO (Input/Output) ..... 339
  - IOException (Klasse) ..... 340
  - isAbsolute (Methode) ..... 343
  - isBlank (Methode) ..... 133
  - isDigit (Methode) ..... 125
  - isDirectory (Methode) ..... 344
  - isEmpty (Methode) ..... 131
  - isExecutable (Methode) ..... 344
  - isHidden (Methode) ..... 344
  - isLeapYear (Methode) ..... 177
  - isLowerCase (Methode) ..... 125
  - isPresent (Methode) ..... 331
  - isReadable (Methode) ..... 344
  - isUpperCase (Methode) ..... 125
  - isWhitespace (Methode) ..... 125
  - isWritable (Methode) ..... 344
- J**
- 
- Jackson (JSON-Bibliothek) ..... 408
  - JAR-Format ..... 398
  - Java installieren ..... 18
  - java (Kommando) ..... 384
    - cp-Option* ..... 398
  - Java Virtual Machine ..... 388
  - Java-Shell ..... 391
  - Java-Standardbibliothek ..... 40
  - java.base (Modul) ..... 42
  - java.io (Paket) ..... 339
  - java.lang (Paket) ..... 40, 187, 395
  - java.nio (Paket) ..... 339, 342
  - java.text (Paket) ..... 138, 184
  - java.time (Paket) ..... 173
  - java.util (Paket) .... 117, 121, 138, 182
  - java.util.function (Paket) ..... 326
  - javac (Kommando) ..... 384
    - cp-Option* ..... 398
  - JavaFX ..... 360
    - Ereignisse* ..... 372
    - Grafik* ..... 374
    - Pakete* ..... 364
  - Javalin ..... 407
  - Jigsaw-Projekt ..... 403
  - JIT-Compiler ..... 388
  - jlink (Kommando) ..... 403
  - join (Methode) ..... 133

- 
- jshell-Kommando ..... 391  
json (Javalin-Methode) ..... 409  
JSON-Format ..... 407  
JVM ..... 388
- ## K
- 
- Key-Value Pair ..... 243  
keySet (Methode) ..... 244  
KI-Tools ..... 44  
    *Intellij* ..... 430  
Kindklasse ..... 261  
Klasse ..... 33, 204  
    *abgeleitete* ..... 261  
    *abstrakte* ..... 271, 292  
    *anonyme* ..... 301, 305, 320  
    *Aufbau* ..... 207  
    *finale Klassen* ..... 270  
    *generische* ..... 310  
    *geschachtelte* ..... 298, 299  
    *konkrete* ..... 272  
    *Konstruktor* ..... 215  
    *lokale* ..... 299  
    *Top-Level* ..... 205  
    *Vererbung* ..... 261  
    *versiegeln* ..... 307  
Klassenmethode ..... 148  
Klassenvariable ..... 213  
Kommentar ..... 40  
Konkrete Klasse ..... 272  
Konstante ..... 74  
Konstruktor ..... 215  
    *Referenz auf* ..... 324  
    *Vererbung* ..... 264
- ## L
- 
- Lambda-Ausdruck ..... 319  
    *für JavaFX-Ereignisse* ..... 372  
    *in DirectoryStreams* ..... 347  
    *in Streams* ..... 331  
    *Syntax* ..... 321  
lastIndexOf (Methode) ..... 131  
launch (Methode) ..... 364  
Left-Shift (bitweises Rechnen) .... 85  
length  
    *Array* ..... 116  
    *String* ..... 131  
lengthOfMonth (Methode) ..... 177  
LIFO (Last In – First Out) ..... 240  
limit (Methode) ..... 330  
lines (Methode) ..... 353  
LinkedList (Klasse) ..... 240  
Lissajous-Figur ..... 378  
list (Methode) ..... 347  
Listen ..... 121, 234, 238  
Literale  
    *für Fließkommazahlen* ..... 65  
    *für Zahlen* ..... 63  
    *für Zeichen (char)* ..... 124  
Local-Variable Type Inference ..... 54  
LocalDate (Klasse) ..... 176  
LocalDateTime (Klasse) ..... 181  
Locale (Klasse)  
    *Datum und Uhrzeit* ..... 176, 184  
    *Fließkommazahlen* ..... 138  
    *Zeichenketten sortieren* ..... 130  
LocalTime (Klasse) ..... 178  
log4j-Sicherheitsproblem ..... 399  
Lokale Klasse ..... 299  
Lokalisierung  
    *Fließkommazahlen* ..... 138  
    *Zeichenketten sortieren* ..... 130  
long (Datentyp) ..... 57, 63  
Long (Klasse) ..... 72  
Long-Term Support (LTS) ..... 19  
Lower-Bounded Wildcards ..... 318

**M**

Machine Time Line .....	173
main (Methode) .....	27, 211
Map .....	234, 243
<i>als Stream verarbeiten</i> .....	463
map (Methode) .....	330
Map.Entry (Schnittstelle) .....	244
mapToDouble (Methode) .....	330
<i>Beispiel</i> .....	463
mapToInt (Methode) .....	330, 332
mapToLong (Methode) .....	330
Math (Klasse) .....	58
Mathematische Operatoren .....	80
Maven .....	360, 400
max (Methode) .....	330, 331
Maximum (Arrays) .....	119
Mehrdimensionales Array .....	113
Mehrzeilige Zeichenkette .....	127
Member Class .....	298
Methode .....	33, 145
<i>anonyme</i> .....	319
<i>Exception-Weitergabe</i> .....	196
<i>finale Methoden</i> .....	270
<i>generische</i> .....	314
<i>Getter/Setter</i> .....	219
<i>main</i> .....	27
<i>Parameter</i> .....	149
<i>Referenz auf Methoden</i> .....	323
<i>Rekursion</i> .....	157
<i>Rückgabewert</i> .....	154
<i>statische</i> .....	213
<i>statische/nichtstatische</i> .....	148
<i>überschreiben (Vererbung)</i> ....	261
min (Methode) .....	330, 331
Minimum (Arrays) .....	119
minusXxx (Methoden) .....	180
Modifizierer .....	62

**Module**

<i>Jigsaw</i> .....	403
<i>Import in Java 25</i> .....	42
module-info.java (Datei) .....	403
move (Methode) .....	348
Mustervergleich in Array .....	164

**N**

Nested Class .....	298
newBufferedReader (Methode) .....	353
newBufferedWriter (Methode) .....	351
newBuilder (Methode) .....	420
newDirectoryStream .....	346
newline (Methode) .....	351
nextDouble (Methode) .....	59, 201
nextGaussian (Methode) .....	59
nextInt (Methode) .....	59
nextLine (Methode) .....	66
nextLong (Methode) .....	59
No operation (NOP) .....	38
Not-effectively-final .....	334
now (Methode) .....	176, 178
NumberFormat (Klasse) .....	138

**O**

Object (Klasse) .....	276
ObjectMapper (Jackson-Klasse) .....	420
Objekt .....	34
<i>Objektvariablen</i> .....	70
Objektorientierte Program-	
<i>mierung</i> .....	204
ObservableList (Schnittstelle) ...	372
of (Methode) .....	179, 239
ofInstant (Methode) .....	181
ofPattern (Methode) .....	176
Oktale Zahl .....	64
OpenJDK .....	20

- 
- Operatoren ..... 77  
*Assoziativität* ..... 79  
*Grundrechenarten* ..... 80  
*mathematische* ..... 80  
*Overloading* ..... 79  
*Priorität* ..... 79  
*ternäre* ..... 93
- OptionalDouble (Klasse) ... 331, 332
- out (Field) ..... 65
- Overloading ..... 152
- Override (Annotation) ..... 261
- P**
- 
- package (Schlüsselwort) ..... 396
- Paket ..... 396  
*Defaultpaket* ..... 398
- Palindrom ..... 143
- Pane (Klasse) ..... 365
- parallelStream (Methode) ..... 330
- Parameter ..... 149  
*Defaultwerte* ..... 152  
*finale Parameter* ..... 151  
*Overloading* ..... 152  
*variable Anzahl* ..... 153  
*Wildcards* ..... 315
- parse (Methode)  
*Datum und Uhrzeit* ..... 178, 186  
*Fließkommazahlen* ..... 138  
*hexadezimale Zahlen* ..... 137
- parseByte (Methode) ..... 137
- parseDouble (Methode) ..... 137
- ParseException (Klasse) ..... 138
- parseFloat (Methode) ..... 137
- parseInt (Methode) ..... 137
- parseLong (Methode) ..... 137
- parseShort (Methode) ..... 137
- Path (Schnittstelle) ..... 342
- Path-Variable ..... 384
- Paths (Klasse) ..... 342
- Pattern Matching  
*instanceof* ..... 273  
*switch* ..... 308
- Period (Klasse) ..... 179
- Pferd (Schachbeispiel) ..... 249
- PI (Konstante) ..... 58
- plusXxx (Methoden) ..... 180
- Polygon (Beispielklasse) ..... 245
- Polymorphie ..... 275  
*Beispiel* ..... 282
- pom.xml (Maven-Datei) .... 362, 401
- Postfix-Notation  
 (Inkrement/Dekrement) ..... 82
- Potenzieren ..... 59
- pow (Methode) ..... 59
- print (Methode) ..... 66
- printf (Methode) ..... 135  
*Datum und Uhrzeit* ..... 184
- println (Methode) ..... 65, 66
- Priorität von Operatoren ..... 79
- private (Schlüsselwort) ..... 211
- protected (Schlüsselwort) ..... 211
- Präfix-Notation ..... 82
- public (Schlüsselwort) ..... 211
- pull (Methode) ..... 240
- push (Methode) ..... 240
- put (Methode) ..... 243
- Q**
- 
- Quadratwurzel ..... 58
- R**
- 
- RadioButton (Klasse) ..... 368
- Random (Klasse) ..... 59
- raw-Zeichenkette ..... 126
- readAllLines ..... 353

- 
- readLine (Methode) ..... 353  
 readln (Methode) ..... 134  
 Record (Schlüsselwort) ..... 222  
     *geschachtelt* ..... 298  
 reduce (Methode) ..... 323, 330  
 Refactoring ..... 427  
 Referenzen ..... 323, 324  
 Reguläre Ausdrücke ..... 132  
 Rekursion ..... 157  
     *Beispiele* ..... 464, 458  
     *Fibonacci-Zahlen* ..... 449  
     *hierarchische Strukturen* ..... 256  
     *Verzeichnisse durchsuchen* .... 466  
 relativize (Methode) ..... 343  
 remove (Methode) ..... 121, 238  
 removeRange (Methode) ..... 238  
 repeat (Methode) ..... 132  
 REPL ..... 391  
 replace (Methode) ..... 132  
 requires (Module) ..... 403  
 resolve (Methode) ..... 342  
 REST-API ..... 406  
 return (Schlüsselwort) ..... 154  
 RIA ..... 360  
 Rich Internet Applications ..... 360  
 Right-Shift (bitweises Rechnen) 85  
 Rundungsfehler ..... 58  
 RuntimeException (Klasse) ..... 193
- S**
- 
- SafeVarargs-Annotation ..... 154  
 scanDouble (Methode) ..... 67  
 Scanner (Klasse) ..... 66  
     *Fehlerabsicherung* ..... 201  
 Scene (Klasse) ..... 365  
 Scene Graph (JavaFX) ..... 365  
 Schach  
     *Beispiel Springer (Klasse)* ..... 249  
     *Beispiel Vererbung* ..... 281  
 Schaltjahr ..... 92, 177  
 Schleife ..... 87  
     *forEach (Methode)* ..... 319  
     *verschachteln* ..... 103  
 Schnittstelle ..... 288  
     *funktionale* ..... 289  
     *generische* ..... 310  
     *Lambda-Ausdrücke* ..... 319  
     *versus abstrakte Klassen* ..... 292  
 SDKMAN ..... 20  
 sealed (Schlüsselwort) ..... 307  
 Semikolon ..... 38  
 Set ..... 234, 240  
 set (Methode) ..... 219, 238  
 setBottom (Methode) ..... 368  
 setCenter (Methode) ..... 368  
 setOnAction (Methode) ..... 364, 372  
 setPadding (Methode) ..... 371  
 setScene (Methode) ..... 365  
 setSpacing (Methode) ..... 371  
 setStyle (Methode) ..... 372  
 setText (Methode) ..... 364  
 setTitle (Methode) ..... 364  
 setValue (Methode) ..... 244  
 setXxx (JavaFX-Methoden) ..... 374  
 short (Datentyp) ..... 57  
 Short (Klasse) ..... 72  
 Short-Circuit-Evaluation ..... 84  
 show (Methode) ..... 365  
 shuffle (Methode) ..... 239  
 SimpleFormat (Klasse) ..... 184  
 sin (Methode) ..... 58  
 size (Methode) ..... 238, 345  
 skip (Methode) ..... 330  
 sort (Methode) ..... 117, 239, 336  
 split (Methode) ..... 132, 331  
 Springer (Schachbeispiel) ..... 249  
 sqrt (Methode) ..... 58

Stack ..... 158  
StackPane (Klasse) ..... 365  
Stage (Klasse) ..... 364  
Standardausgabe ..... 65  
Standardbibliothek ..... 40  
StandardCopyOption ..... 349  
Standardeingabe ..... 66  
StandardOpenOption ..... 351  
start (Methode) ..... 364  
startWith (Methode) ..... 131  
static (Schlüsselwort) ..... 209, 213  
Statische Methode ..... 148, 213  
Statische Variable ..... 213  
Statischer Import ..... 396  
Stream ..... 329  
    *von Maps* ..... 463  
    *Dateien durchlaufen* ..... 347  
    *Minimum ermitteln* ..... 323  
    *Textdateien lesen* ..... 353  
stream (Methode) ..... 330  
Strichpunkt ..... 38  
String (Klasse) ..... 126  
    *Eigenheiten* ..... 128  
    *sortieren* ..... 130  
    *vergleichen* ..... 128  
StringBuilder (Klasse) ..... 140  
strip (Methode) ..... 133  
strokeXxx (JavaFX-Methoden) ... 374  
Subklasse ..... 261  
substring (Methode) ..... 132  
sum (Methode) ..... 330  
super (Schlüsselwort) ..... 263  
    *Lambda-Ausdrücke* ..... 322  
Superklasse ..... 261  
Swing ..... 360  
switch (Schlüsselwort) ..... 94  
    *für versiegelte Klassen* ..... 308  
    *moderne Syntax* ..... 96  
System (Klasse) ..... 342

## T

---

Tag pro Monat (Beispiel) ..... 95  
Task-Manager (Beispiel) ..... 411  
Template (IntelliJ) ..... 431  
Temporäre Dateien ..... 348  
Temporäre Verzeichnisse ... 342, 348  
Ternärer Operator ..... 93  
TextArea (Klasse) ..... 368  
Textdatei  
    *lesen* ..... 353  
    *schreiben* ..... 351  
this (Schlüsselwort) ..... 216  
    *Lambda-Ausdrücke* ..... 322  
ThreeTen-Projekt ..... 172  
Throwable (Klasse) ..... 193  
toAbsolute (Methode) ..... 343  
toCharArray (Methode) ..... 139  
toHours (Methode) ..... 179  
toList (Methode) ..... 330  
toLowerCase (Methode) .... 125, 132  
toMinutes (Methode) ..... 179  
Top-Level-Klasse ..... 205, 207  
toRealPath (Methode) ..... 343  
toString (Methode) ..... 278  
toUpperCase (Methode) .... 125, 132  
transient (Modifier) ..... 62  
trim (Methode) ..... 132, 133  
try (Schlüsselwort) ..... 194  
Type Inference ..... 54  
Typumwandlung ..... 60

## U

---

Überlaufkontrolle ..... 57  
Uhrzeit ..... 172, 178  
    *konvertieren in Zeichenkette* 178  
UML-Diagramm ..... 226  
Unterklasse ..... 261

Upcast ..... 274  
 Upper-Bounded Wildcards ..... 317  
 uri (Methode) ..... 420  
 UserPrincipal (Schnittstelle) ..... 345  
 UTF-8-Codierung ..... 351

## V

---

valueOf (Methode) ..... 135  
 values (Methode) ..... 243  
 var (Schlüsselwort) ..... 54  
   *Arrays* ..... 112  
   *for-each-Schleifen* ..... 105  
   *Instanzvariablen* ..... 207  
 varargs (Parameter) ..... 153  
 Variable ..... 52  
   *statische* ..... 213  
 Variable Capture ..... 305  
   *Lambda-Ausdrücke* ..... 322  
 VBox (Klasse) ..... 368  
 Vererbung ..... 260  
 Vergleich von Objekten ..... 83  
 Verknüpfte Bedingungen ..... 84  
 Verschachtelte Klassen ..... 298  
 Versiegelte Klassen ..... 307  
 Verzeichnis ..... 339  
   *aktuelles* ..... 342  
   *alle Dateien ermitteln* ..... 346  
   *erzeugen* ..... 348  
   *Existenz testen* ..... 344  
   *Heimatverzeichnis* ..... 342  
   *rekursiv durchlaufen* ..... 347  
   *temporäres* ..... 342, 348  
   *umbenennen* ..... 348  
   *zusammensetzen* ..... 342  
 Verzweigung ..... 87  
 Visual Studio Code ..... 30  
 volatile (Modifier) ..... 62

Vorzeichen ..... 57  
 VSCode ..... 30

## W

---

walk (Methode) ..... 347  
 Warnhinweise (IntelliJ) ..... 434  
 while-Schleife ..... 105  
   *break und continue* ..... 107  
   *Endlosschleifen* ..... 109  
 Whitespace in Zeichenketten ..... 133  
 Wildcard-Import ..... 394  
 Wildcards (generische Klassen) ..... 315  
   *Lower-Bounded Wildcards* ..... 318  
   *Upper-Bounded Wildcards* ..... 317  
 Windows-Path-Konfiguration ..... 384  
 withLocale (Methode) ..... 176  
 withMonth (Methode) ..... 177  
 Wrapper-Klasse ..... 72  
   *generische Klassen* ..... 237, 312  
 write (Methode) ..... 351, 352

## Y

---

yield (switch-Rückgabewert) ..... 97

## Z

---

Zahl  
   *binäre* ..... 64  
   *hexadezimale* ..... 64  
   *oktale* ..... 64  
 Zeichenketten ..... 124, 126  
   *konvertieren in Zahl* ..... 137  
   *mehrzeilige* ..... 127  
   *sortieren* ..... 130  
   *vergleichen* ..... 83, 128  
   *Whitespace* ..... 133



Dieser Grundkurs gibt Ihnen einen **schnellen und gründlichen Java-Einstieg**, ganz ohne Vorkenntnisse. Sie lernen die **kompletten Grundlagen** der Sprache sowie alle wichtigen Programmierkonzepte. Immer mit Rücksicht auf Ihre Lernkurve führt Michael Kofler Sie von einfachen Codebeispielen hin zu objektorientierten Programmen. Dabei zeigt er auch, wie **KI-Tools** den Lernprozess unterstützen können. Zu jedem Kapitel finden Sie **erprobte Übungen und Beispiele**, die Ihnen beim Verstehen helfen. So werden Sie fit in Java!

**Dieser Grundkurs eignet sich hervorragend für Studierende der Informatik und anderer technischer Fächer.**

## import java.grundkurs.rheinwerk

Installation von Java und IntelliJ IDEA • Schleifen, Verzweigungen und Methoden • Vererbung und Interfaces • Objektorientierung und Schnittstellen • Exceptions (Fehlerbehandlung) • Umgang mit Datum und Uhrzeit • Zugriff auf Dateien • Listen, Sets und Maps • Lambda-Ausdrücke • Generische Klassen und Methoden • GUIs mit JavaFX • REST-API erstellen



**Michael Kofler** ist der erfolgreichste IT-Fachbuchautor im deutschsprachigen Bereich. Er lehrt Software Development an der FH JOANNEUM in Kapfenberg und setzt Java bei seinen freiberuflichen Entwicklungsprojekten ein.

