Java – Der Grundkurs

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

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`

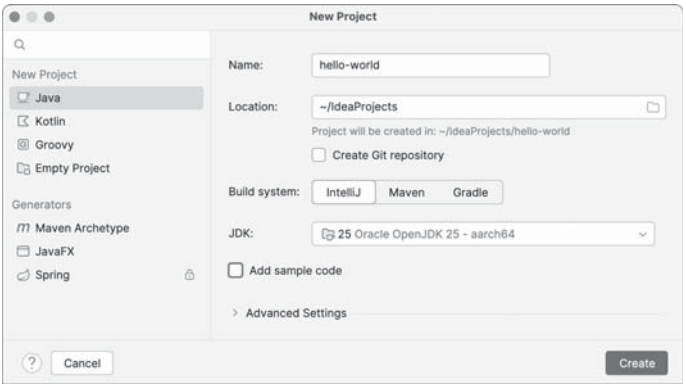

Abbildung 1.1 Ein neues Java-Projekt einrichten

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.

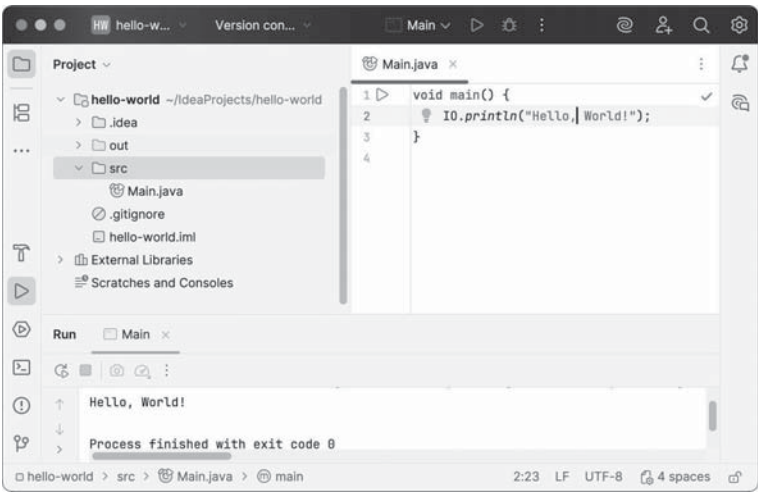

Abbildung 1.2 Ein typisches Setup in IntelliJ: links die Projektübersicht, rechts eine Codedatei, unten die Ausgaben eines Programms

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.

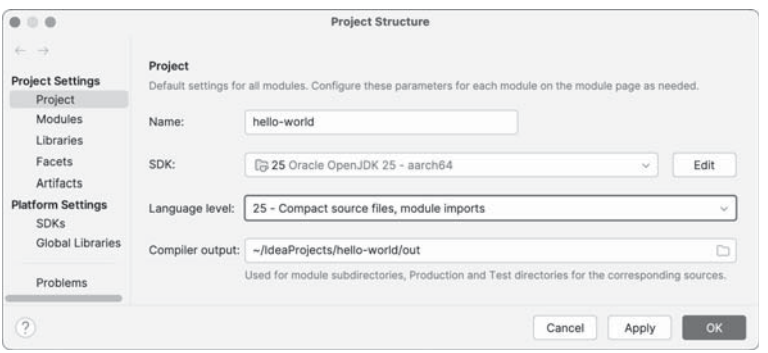

Abbildung 1.3 In den Projekteinstellungen stellen Sie die Java-Version ein.

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`.

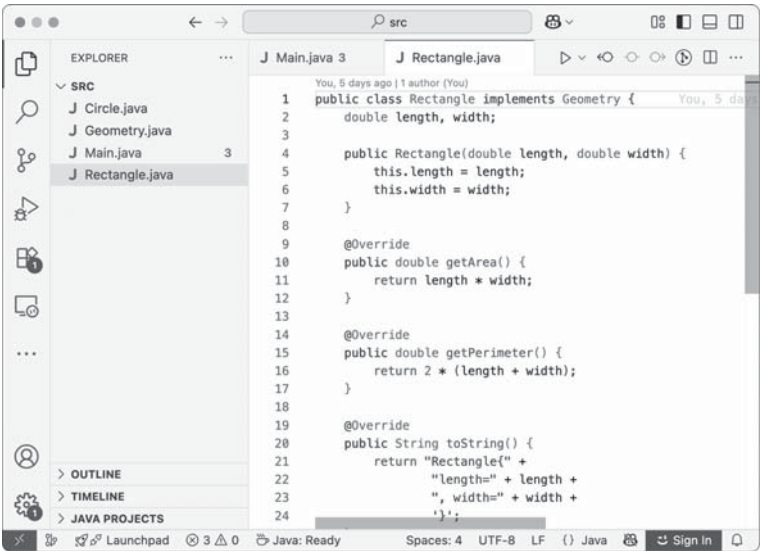

Abbildung 1.4 Visual Studio Code

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 `->` 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 →	`()`	Methodenaufruf
	`[]`	Zugriff auf Felder (Arrays)
	`.`	Zugriff auf Objekte, Methoden etc.
	`::`	Referenz auf Methoden
2 →	`++ --`	Inkrement/Dekrement (Postfix, z. B. `a++`)
3 ←	`++ --`	Inkrement/Dekrement (Präfix, z. B. `--b`)
	`+ -`	Vorzeichen
	`! ~`	logisches Nicht, binäres Nicht (*NOT*)
	`(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 →	`> >=`	Vergleich größer bzw. größer-gleich
	`< <=`	Vergleich kleiner bzw. kleiner-gleich
	`instanceof`	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 (*Short-circuit Evaluation*)
13 →	`||`	logisches Oder (*Short-circuit Evaluation*)
14 ←	`a ? b : c`	Auswahloperator (*Ternary Operator*)
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 Operatorentabelle 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 euklidschen Modulo-Operation! Weitere Details zu diesem Thema können Sie in der Wikipedia nachlesen:

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 (*Short-Circuit Evaluation*)
`||`	logisches Oder (*Short-Circuit Evaluation*)

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 Lösungs*vorschläge* 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 dazugehörende 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++) {
    ...
}
```