Michael Kofler

Java

Der Grundkurs

- Die kompakte Einführung in Java
- Vom ersten Schritt bis zum objektorientierten Programm
- Mit kommentierten Codebeispielen, Übungen und Lösungen zum Selbstlernen

Alle Codebeispiele zum Download

Rheinwerk Computing

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 Java-Entwicklungswerkzeugen wie `java`, `javac` und der IntelliJ IDEA vertraut.

1.1 Einführung

Programmieren heißt, einem Computer in einer für ihn verständlichen Sprache Anweisungen zu geben. Als Nichtprogrammierer können Sie Computerprogramme selbstverständlich schon *anwenden*, im Web surfen, eine Mail verfassen, eine Android-App bedienen oder auf dem iPad Schach spielen. Aber wenn Sie ein Programm benötigen, das es in dieser Form noch nicht gibt, dann müssen Sie dieses *selbst* entwickeln. Dieses Buch vermittelt Ihnen die dazu erforderlichen Grundkenntnisse.

Natürlich gibt es auch ganz pragmatische Gründe: Sie wollen (oder müssen?) programmieren lernen, weil dies Teil Ihrer Schul- oder Universitätsausbildung ist. Oder Sie möchten programmieren können, um damit Geld zu verdienen. Oder es reizt Sie einfach, die IT-Welt besser verstehen zu

können. Aus meiner persönlichen Sicht hat Programmieren auch etwas Spielerisches an sich. Programmieren kann Spaß machen – so wie das Lösen eines Denksporträtsels.

Warum Java?

Programmiersprachen gibt es sprichwörtlich wie Sand am Meer. Warum gerade Java? Java hat sich in den vergangenen Jahrzehnten als *die* erste Programmiersprache etabliert, mit der unzählige Studenten ihre ersten Programme verfasst haben. Dafür gibt es gute Gründe:

- Java ist kostenlos.
- Java zeichnet sich durch eine klare, relativ leicht zu erlernende Syntax aus, was gerade im schulischen Bereich ein großer Vorteil ist.
- Java ist plattformunabhängig: Ein einmal entwickeltes Java-Programm kann daher gleichermaßen unter Windows, Linux oder macOS ausgeführt werden.
- Java steht als Open-Source-Code zur Verfügung. Damit kann Java unkompliziert auf neue Plattformen portiert werden. Genau das war vor einigen Jahren der Fall, als Google Android entwickelte und auf Java als Programmiersprache für die App-Entwicklung setzte.
- Java ist im Unternehmensbereich sehr beliebt. Es gibt unzählige Entwicklungswerkzeuge, die dabei helfen, exakten, *sauberen* und wartbaren Code zu entwickeln – auch bei sehr großen Projekten.
- Java ist sehr universell anwendbar. Sie können damit einfache Konsolenkommandos ebenso entwickeln wie Windows-Programme mit grafischer Benutzeroberfläche, Webanwendungen für den Server-Einsatz oder Android-Apps.

»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, neues Android-Spiel oder eine interaktive Webseite programmieren.

Zwar enthält Kapitel 17, »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 Java-Fundament, mit dem Sie sich dann weiter in ein Java-Teilgebiet einarbeiten können, z. B. in die Entwicklung von Webanwendungen.

Das große Versionsnummernchaos

Zwischen Java 8 (vorgestellt im März 2014) und Java 9 (September 2017) vergingen mehr als drei Jahre. Oracle erschien das zu langsam und hat deswegen 2017 den Java-Entwicklungszyklus auf den Kopf gestellt – und das Kind sprichwörtlich mit dem Bade ausgeschüttet. Seit Herbst 2017 erscheint alle sechs Monate eine neue Java-Version.

Für Java-Freaks, die ständig die neuesten Features ausprobieren möchten, ist das fantastisch. Für alle anderen Entwickler ist das weniger toll. Warum?

- Professionelle Entwickler brauchen für die Entwicklung neuer Programme Zeit. Danach soll das Programm in der Regel beim Kunden über Jahre laufen. Diese Prozesse erfordern ein Maß an Stabilität, das mit halbjährlichen Updates nicht mehr gegeben ist.
- Im Unterricht ist es eine Katastrophe, wenn jeder Student bzw. jede Studentin eine andere Java-Version verwendet. Administratoren von Schulen und Universitäten sind damit überfordert, Laborrechner halbjährlich zu aktualisieren. Die eingesetzten Entwicklungsumgebungen (z. B. Eclipse oder IntelliJ IDEA) müssen ebenfalls zur jeweiligen Java-Version passen.
- Auch für Linux-Distributoren sind die ständigen Versionswechsel eine Herausforderung. Auf vielen Linux-Systemen laufen deswegen stan-

dardmäßig relativ alte Java-Versionen. Aktuelle Versionen können zwar manuell installiert werden, das verursacht aber Update-Probleme.

- Bücher und Schulungsvideos zu Java können unmöglich alle sechs Monate neu gedruckt bzw. aktualisiert werden.

Diese Probleme hat natürlich auch Oracle erkannt und mit sogenannten LTS-Versionen teilweise gelöst. Alle paar Jahre gibt es besondere Java-Versionen mit *Long Time Support* (LTS). Zuletzt war das bei Java 11 der Fall, das nächste Mal wird dies voraussichtlich für Java 17 gelten. Die Besonderheit derartiger LTS-Versionen besteht darin, dass sie über einen langen Zeitraum mit Sicherheits-Updates und Fehlerbehebungen (Bugfixes) versorgt werden und somit jahrelang produktiv genutzt werden können.

Das LTS-Modell hat aber einen Haken: Die Bugfixes gibt es nur für Kunden mit einer kostenpflichtigen Lizenz. Als Privatanwender können Sie natürlich kostenfrei bei der jeweils letzten LTS-Version bleiben, müssen dann aber auf Bugfixes verzichten. Die Alternative besteht darin, dass Sie eben alle sechs Monate die jeweils neueste Java-Version samt einer dazu passenden Entwicklungsumgebung installieren. Eine dritte Variante ist es, die originalen Java-Pakete von Oracle durch dazu kompatible Pakete mit einer liberaleren Update-Politik zu ersetzen. Entsprechende Downloads finden Sie z. B. unter *https://adoptopenjdk.net*.

Empfehlung

Wenn es Ihnen nur darum geht, Java zu lernen, empfehle ich Ihnen, entweder die letzte LTS-Version oder eben die gerade aktuelle Java-Version zu installieren und es dann dabei zu belassen. Fertig!

Dieses Buch bezieht sich zwar auf Java 11, aber es ist durchaus kein Problem, wenn Sie auf Ihrem Rechner z. B. Java 13 verwenden. Niemand zwingt Sie, die seit Java 11 durchgeführten Neuerungen zu nutzen. Der rasche Versionszyklus bringt es mit sich, dass die Neuerungen pro Version in der Regel bescheiden sind und häufig Features betreffen, die für Einsteiger gar nicht relevant sind. Ein Großteil der in diesem Buch präsentierten Programme funktioniert sogar noch mit Java 9!

Falls Sie IntelliJ als Entwicklungsumgebung verwenden, erkennt IntelliJ beim Laden der Beispielprogramme zu diesem Buch möglicherweise die auf Ihrem Rechner installierte Java-Version nicht. Dieses Problem lässt sich zum Glück ganz leicht lösen: Sie führen File • Project Structure aus und stellen die beiden Listenfelder für Project SDK und Project language level neu ein – fertig! (Details folgen in Abschnitt 1.8, »Hello IntelliJ IDEA« sowie im Anhang.)

Kleines Java-Glossar

Spätestens beim Versuch, Java zu installieren, werden Sie mit einer Menge Abkürzungen konfrontiert. Deren Bedeutung ist rasch erklärt:

- **JRE versus JDK:** Je nachdem, ob Sie Java-Programme nur ausführen oder aber selbst entwickeln möchten, müssen Sie auf Ihrem Rechner das *Java Runtime Environment* (JRE) oder aber das *Java Development Kit* (JDK) installieren. Die Trennung zwischen diesen Varianten hat den Vorteil, dass für reine Java-Anwender das viel kleinere JRE ausreicht. (Das JDK enthält übrigens das JRE, Sie müssen also nicht beides installieren.)

- **Java SE versus Java EE:** Ich habe einleitend erwähnt, dass Java zur Entwicklung von ganz unterschiedlichen Anwendungstypen verwendet wird. Gewissermaßen als *Normalfall* gelten einfache Programme mit oder ohne grafische Benutzeroberfläche. Dafür ist die *Java Standard Edition* (Java SE) gedacht. Speziell für die Entwicklung von Web- und Server-Anwendungen gibt es die *Java Enterprise Edition* (Java EE). Java EE ist eine Erweiterung von Java SE um diverse Zusatzkomponenten bzw. Bibliotheken.

- **Oracle Java versus OpenJDK:** Oracle kaufte 2010 die Firma Sun auf und übernahm damit auch die Kontrolle über das von Sun entwickelte Java. Die *offiziellen* Java-Versionen stammen somit von Oracle.

  Da Sun aber bereits 2006 den Quellcode von Java als Open-Source-Code freigab, gibt es weitere Java-Implementierungen. Am populärsten ist das *OpenJDK* des Iced-Tea-Projekts. Es kommt standardmäßig auf allen

Linux-Distributionen zum Einsatz, primär deswegen, weil Oracle eine unkomplizierte Integration des offiziellen Java in Linux-Distributionen nicht erlaubt.

Kurz zusammengefasst: Um die Beispiele aus diesem Buch nachvollziehen zu können, benötigen Sie das **JDK** von **Java SE**. Dabei ist es egal, ob Sie die offiziellen Pakete von Oracle verwenden oder alternative Pakete auf der Basis von OpenJDK.

JavaScript hat (fast) nichts mit Java zu tun!

JavaScript erlaubt es, Code direkt in Webseiten einzubetten und auf dem Browser auszuführen. Für moderne Webanwendungen ist das sehr wichtig. JavaScript ist aber grundverschieden von Java: Die Syntax ist simpler, es gibt keinen Byte-Code, keine *Java Virtual Machine* etc. In diesem Buch geht es um Java, nicht um JavaScript!

Windows, Linux oder macOS?

Ganz egal, ob Sie am liebsten unter Windows, Linux oder macOS arbeiten: Jedes dieser Betriebssysteme ist wunderbar geeignet, um Java kennenzulernen! Dass die meisten Screenshots dieses Buchs unter Windows erstellt wurden, hat nichts mit meinen persönlichen Vorlieben zu tun. Vielmehr ist Windows gerade im Schulbereich noch immer das dominierende Betriebssystem.

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/4877

1.2 Java und die IntelliJ IDEA installieren

Bevor Sie Ihr erstes Java-Programm verfassen und ausführen können, benötigen Sie zwei Dinge: das sogenannte *Java Development Kit* (JDK) und einen Code-Editor. Vor allem für die Entwicklung größerer Programme ist außerdem eine grafische Entwicklungsumgebung zweckmäßig, also ein *Integrated Development Environment*, kurz IDE. Aber der Reihe nach:

- **JDK:** Um selbst Java-Programme zu entwickeln, müssen Sie das *Java Development Kit* (JDK) installieren. Es enthält unzählige Entwicklungswerkzeuge und Bibliotheken. Für die Beispiele in diesem Buch sollten Sie das JDK in der Version 11 oder neuer verwenden. Für viele Beispiele reicht sogar Version 9 aus.
- **Editor:** Windows-Anwender können ihre ersten Java-Programme sogar mit dem Editor *Notepad* verfassen, der jeder Windows-Version beiliegt. Besser ist es aber, einen Editor zu verwenden, der Java-Code *versteht*, verschiedene Elemente des Codes in unterschiedlichen Farben darstellt und Sie bei der Eingabe unterstützt.

  Für Windows-Anwender ist das kostenlose Programm *Notepad++* der perfekte Startpunkt. Linux-Anwendern lege ich für die ersten Versuche *Gedit*, *Kate* oder *KWrite* ans Herz. Unter macOS bietet sich *TextMate* an. Für Fortgeschrittene sind plattformübergreifend die Editoren *Atom*, *Brackets* oder *Visual Code* empfehlenswert.
- **Entwicklungsumgebung (IDE):** Mit einem Editor können Sie Code nur eingeben. *Integrated Development Environments* (IDEs) unterstützen Sie außerdem bei der Strukturierung Ihres Codes, bei der Fehlersuche, beim Kompilieren und beim Beheben von Fehlern, bei der Dokumentation, bei der Versionsverwaltung etc. Aus diesen Gründen ist eine IDE für größere Projekte nahezu zwingend erforderlich. Für Java-Einsteiger gilt dies aber nicht! Alle Java-IDEs bringen mit ihren unzähligen Funktionen eine zusätzliche Komplexität in den Lernprozess.

  Die drei populärsten Java-IDEs sind *Eclipse*, *IntelliJ IDEA* und *NetBeans*. Ich konzentriere mich in diesem Buch auf die IntelliJ IDEA, die aus meiner Sicht die beste Funktionalität bietet und gleichzeitig einsteigertauglich ist.

1.3 Installation unter Windows

Die aktuelle JDK-Version der Java Standard Edition finden Sie auf der folgenden Seite zum kostenlosen Download:

https://www.oracle.com/technetwork/java/javase/downloads/index.html

Laden Sie die rund 180 MByte große EXE-Datei herunter, und führen Sie sie aus. Damit installieren Sie alle erforderlichen Programme und Bibliotheken (siehe Abbildung 1.1). Sie können dabei alle Voreinstellungen unverändert übernehmen.

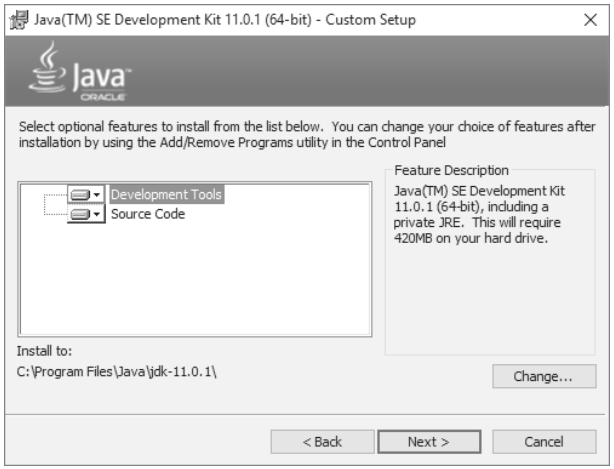

Abbildung 1.1 JDK-Installation unter Windows

Path-Variable einstellen

Als Teil des JDK werden die Programme `java.exe` und `javac.exe` installiert, üblicherweise in das Verzeichnis `C:\Program Files\Java\jdk-<n>\bin`. Wenn Sie manuell Java-Programme kompilieren möchten, *ohne* dabei eine Entwicklungsumgebung zu verwenden, müssen Sie dieses Verzeichnis in die Windows-Systemvariable `Path` einfügen.

Die erforderliche Konfiguration ist leider ein wenig umständlich: Im Startmenü oder in der Systemsteuerung suchen Sie nach dem Punkt Umgebungsvariablen für dieses Konto bearbeiten. Im Dialog Umgebungsvariablen können Sie nun die Variable `Path` zur Bearbeitung öffnen.

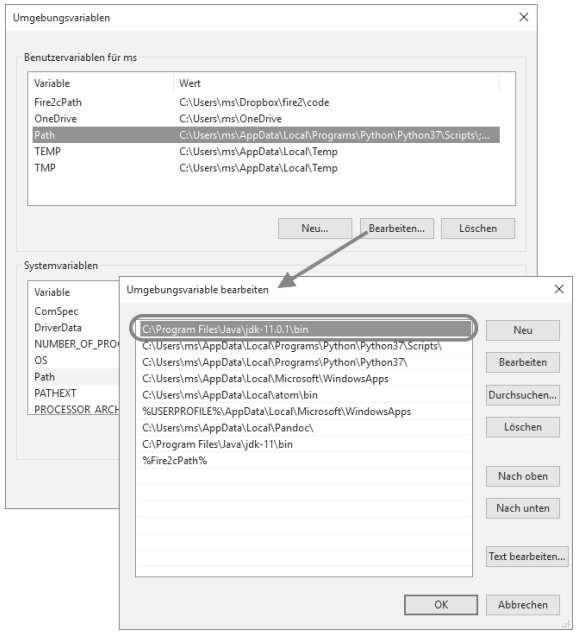

Abbildung 1.2 Die Path-Variable unter Windows verändern

Nun klicken Sie zuerst auf Neu und dann auf Durchsuchen und fügen der Path-Variablen das Verzeichnis `C:\Program Files\Java\jdk-<n>\bin` hinzu (siehe Abbildung 1.2). Beachten Sie, dass der Dateiauswahldialog anstelle von `Program Files` die deutsche Übersetzung `Programme` anzeigt.

Es ist zulässig, dass mehrere Java-Versionen parallel installiert sind. Klicken Sie daher so lange auf den Button Nach oben, bis der neue Eintrag am Beginn der Liste steht und so Vorrang vor eventuell älteren Java-Installationen hat.

Wenn Sie sich vergewissern möchten, dass alles funktioniert hat, starten Sie das Programm *Eingabeaufforderung* (Programm `cmd.exe`) und führen dort das Kommando `javac -version` aus. Das Ergebnis muss wie folgt aussehen:

```
> javac -version
  javac 11.0.1
```

Notepad++ und IntelliJ IDEA installieren

Den ausgezeichneten Editor Notepad++ können Sie von der folgenden Webseite kostenlos herunterladen und unkompliziert installieren:

https://notepad-plus-plus.org

Die kostenlose Community-Variante der IntelliJ IDEA (*Integrated Development Environment Application*) laden Sie von der folgenden Seite herunter:

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

Achten Sie darauf, dass Sie nicht irrtümlich die Ultimate-Edition herunterladen, die nach dem Auslaufen der kostenlosen Testperiode eine Lizenz erfordert!

Zur Installation führen Sie die resultierende `*.exe`-Datei aus. Im Installationsprogramm belassen Sie alle Voreinstellungen. Wenn Sie möchten, können Sie ein Desktop-Icon einrichten, damit Sie das Programm nicht im Startmenü suchen müssen (Option CREATE DESKTOP SHORTCUT • 64-BIT LAUNCHER).

Beim ersten Start müssen Sie sich zwischen einem hellen und einem dunklen Farbschema entscheiden. Das Farbschema können Sie bei Bedarf später in den Einstellungen ändern. IntelliJ IDEA bietet Ihnen dann noch die Möglichkeit an, zusätzliche Plug-ins zu installieren. Darauf sollten Sie verzichten. Die IntelliJ IDEA enthält bereits in der Grundausstattung mehr Funktionen, als Sie momentan benötigen.

Sobald Sie das erste Java-Projekt starten (CREATE NEW PROJECT), müssen Sie der IntelliJ IDEA einmalig mitteilen, wo sich das JDK befindet. Auf diesen Schritt gehe ich in Abschnitt 1.8, »Hello IntelliJ IDEA«, näher ein.

1.4 Installation unter Ubuntu Linux

Jede Linux-Distribution stellt Java-Pakete zur Verfügung, die unkompliziert und schnell installiert werden können. Unter Ubuntu Linux öffnen Sie dazu ein Terminalfenster und führen dieses Kommando aus:

```
sudo apt install openjdk-11-jdk
```

Damit wird eine Open-Source-Variante von Java 11 installiert. Vergewissern Sie sich in einem Terminalfenster, dass alles funktioniert hat! `java -version` und `javac -version` zeigen an, welche Java-Version installiert wurde (siehe Abbildung 1.3).

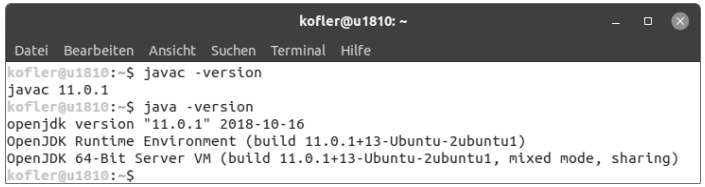

Abbildung 1.3 Java wurde erfolgreich installiert.

IntelliJ IDEA installieren

Unter Ubuntu steht IntelliJ als sogenanntes Snap-Paket zur Verfügung. Zur Installation verwenden Sie einfach das Paketverwaltungsprogramm *Ubuntu Software*. Bei anderen Linux-Distributionen laden Sie von der folgenden Webseite das komprimierte TAR-Archiv der Community-Version herunter und speichern es:

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

Anschließend führen Sie die folgenden Kommandos aus, um die Entwicklungsumgebung im Heimatverzeichnis auszupacken und zu starten:

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

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

1.5 Installation unter macOS

Das aktuelle JDK für macOS steht hier zum Download bereit:

http://www.oracle.com/technetwork/java/javase/downloads

Zur Installation öffnen Sie das heruntergeladene DMG-Image per Doppelklick und führen dann den darin enthaltenen PKG-Installer aus. Auch unter macOS können Sie sich in einem neu geöffneten Terminal-Fenster mit dem Kommando `javac -version` davon überzeugen, dass der Java-Compiler nun in der richtigen Version zur Verfügung steht.

IntelliJ IDEA installieren

Die kostenlose *Community Edition* der IntelliJ IDEA finden Sie auf der folgenden Seite zum Download:

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

Die heruntergeladene DMG-Datei öffnen Sie nun per Doppelklick. Danach verschieben Sie das IntelliJ-Icon in das Verzeichnis `Anwendungen`. Fertig!

1.6 »Hello World« mit javac und java manuell übersetzen

Ich empfehle Ihnen, die erste Version des Hello-World-Programms *nicht* mit der IntelliJ IDEA oder einer anderen Entwicklungsumgebung zu entwickeln. Es trägt sehr zum Verständnis für Java bzw. für den Prozess des Programmierens bei, wenn Sie einmal die manuellen Abläufe kennenlernen und sehen, mit wie wenig Overhead ein Java-Programm entwickelt werden kann.

Code verfassen und speichern

Der Code für das Hello-World-Programm umfasst die folgenden Zeilen, deren Bedeutung ich Ihnen etwas weiter unten genauer erläutern werde:


```
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

Beispielprogramme

Alle Beispielprogramme des Buchs finden Sie hier zum Download:

www.rheinwerk-verlag.de/4877

Achten Sie darauf, dass Sie den Text *exakt* abschreiben, inklusive aller Klammern und Strichpunkte. Anschließend speichern Sie die Datei unter dem Namen `HelloWorld.java` in einem beliebigen Verzeichnis. Der Dateiname *muss* mit `.java` enden, und er muss exakt mit der Bezeichnung nach dem Schlüsselwort `class` übereinstimmen. Das gilt auch für die Groß- und Kleinschreibung! Übrigens ist es üblich, dass der Klassenname mit einem großen Anfangsbuchstaben beginnt – so wie hier `HelloWorld`.

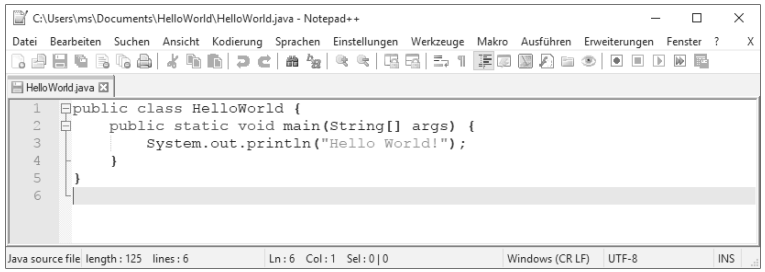

Abbildung 1.4 Der Hello-World-Code in Notepad++

Zur Code-Eingabe starten Sie einen beliebigen Editor, der Ihren Text ohne irgendwelche Formatierung speichern kann. Microsoft Word ist *nicht* geeignet! Unter Windows können Sie zur Not das Programm Notepad verwenden. Viel besser geeignet ist Notepad++ (siehe Abbildung 1.4). Beachten Sie, dass die farbige Hervorhebung des Codes erst funktioniert, nachdem

Sie den Code in einer Datei mit der Kennung `.java` gespeichert haben – denn erst damit weiß Notepad++, dass es sich um Java-Code handelt.

Das Programm kompilieren und ausführen

Als Nächstes geht es darum, aus dieser Textdatei ein durch den Java-Interpreter ausführbares Programm zu machen. Der von Ihnen eingegebene Code muss also in ein Format umgewandelt werden, das der Computer lesen kann. Diesen Umwandlungsprozess nennt man Kompilieren. Verantwortlich dafür ist der Java-Compiler, also das Kommando `javac`.

Um `javac` auszuführen, öffnen Sie unter Windows mit ⊞ `cmd` ↵ eine Eingabeaufforderung. Alternativ können Sie auch die PowerShell verwenden. Unter Linux oder macOS öffnen Sie ein Terminalfenster.

Mit `cd` wechseln Sie nun in das Verzeichnis, in dem Sie die Java-Datei gespeichert haben. `javac Name.java` kompiliert den Java-Code zu einer sogenannten Klassendatei `Name.class`:

```
cd code-verzeichnis
javac HelloWorld.java
```

Die Klassendatei enthält Byte-Code. Das ist eine binäre Darstellung Ihres Codes, die von jedem Java-Interpreter ausgeführt werden kann – egal, unter welchem Betriebssystem. Zum Ausführen übergeben Sie den Namen der Klassendatei *ohne* die Endung `.class` an den Java-Interpreter `java` (siehe Abbildung 1.5):

```
java HelloWorld
  Hello World!
```

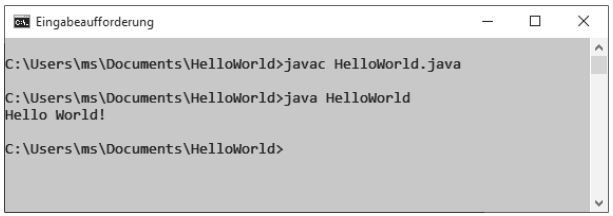

Abbildung 1.5 »Hello World« unter Windows ausführen und kompilieren

Unser Beispielprogramm gibt lediglich eine Zeile mit dem Text *Hello World!* aus, aber im weiteren Verlauf dieses Buchs präsentiere ich Ihnen natürlich eine Menge interessantere Programme.

Unter Linux und macOS gehen Sie im Prinzip wie unter Windows vor: Zum Verfassen des Codes verwenden Sie Gedit, Kate, Vi, Emacs, TextEdit oder einen beliebigen anderen Texteditor. In einem Terminalfenster kompilieren Sie dann den Code und führen ihn aus.

Nichts als Fehlermeldungen!

Wenn Ihnen auch nur ein winziger Fehler bei der Code-Eingabe unterläuft, liefert `javac` eine oder mehrere Fehlermeldungen. Oft ist nur ein fehlender Strichpunkt oder eine vergessene Klammer schuld. Die Fehlersuche gestaltet sich oft schwierig: Die englischsprachigen Fehlermeldungen sind schwer zu interpretieren und weisen nicht immer auf die tatsächliche Ursache hin.

Grundsätzlich ist es zweckmäßig, sich bei der Fehlersuche auf die *erste* Fehlermeldung zu konzentrieren. Ist der erste Fehler behoben, verschwinden oft auch alle daraus resultierenden Folgefehler. Suchen Sie also den Fehler, beheben Sie ihn, speichern Sie die Code-Datei, und führen Sie dann `javac Name.java` neuerlich aus.

Kompilieren und Ausführen in einem Kommando

Seit Version 11 reicht in einfachen Fällen *ein* Kommando aus, um eine Java-Datei zu kompilieren und sofort auszuführen:

```
java HelloWorld.java
```

Dieses Kommando funktioniert allerdings nur, wenn im aktuellen Verzeichnis nicht schon eine entsprechende `*.class`-Datei existiert. (Diese müssen Sie gegebenenfalls zuerst löschen.)

`java HelloWorld.java` speichert das Kompilat nur im Arbeitsspeicher, nicht in einer lokalen Datei. Deswegen finden Sie nach dem Ausführen des Programms keine neue `*.class`-Datei im aktuellen Verzeichnis.

Der Hello-World-Code

Ich bin Ihnen noch die Erklärung schuldig, was die fünf Zeilen des Hello-World-Codes eigentlich bedeuten. Schließlich geht es nicht an, dass bereits die ersten Code-Zeilen unbegreiflich sind, oder? Leider ist eine 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. Eine Klasse ist ein abgeschlossener Code-Bereich. Die Strukturierung von Code in Klassen bietet eine Menge Vorteile, sowohl bei der Nutzung dieser Klassen in anderen Code-Teilen als auch bei der Aufteilung des Codes in überschaubare Code-Portionen.

Auf `class` folgt der Name der Klasse. Klassennamen beginnen immer mit einem Großbuchstaben. Dem Klassennamen folgt schließlich 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 sogenannten *Methoden*. Methoden bündeln mehrere Zeilen Code zu einer Einheit, die eine bestimmte Aufgabe erledigt. Im Hello-World-Programm gibt es nur *eine* Methode mit dem Namen `main`. Diese Methode hat dafür eine herausragende Bedeutung: `main` gilt als Startpunkt jedes Java-Programms.

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 der Code der Methode ausgeführt werden kann, ohne dass vorher ein Objekt der Klasse (eine Instanz der Klasse) erzeugt werden muss.
- 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. `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.

Der Code der Methode beginnt mit der Klammer { und endet mit der dazugehörenden Klammer }.

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

`System.out.println` gibt den nachfolgenden, in runden Klammen angegebenen Parameter auf dem Bildschirm aus. Bei unserem Beispielprogramm handelt es sich um eine durch Anführungszeichen gekennzeichnete Zeichenkette. `println` kann aber auch Zahlen und andere Daten ausgeben.

`println` ist eine in der Java-Bibliothek vordefinierte Methode. Sie können diese Methode also nutzen, ohne sie vorher selbst definiert zu haben.

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 wiederum eine statische Variable (ein Attribut oder auf Englisch

Field) dieser Klasse, das auf das Standardausgabeobjekt verweist. Dieses Objekt wird von Java automatisch beim Start des Programms erzeugt.

Der Vollständigkeit halber sei noch erwähnt, dass sich die `System`-Klasse im Paket `java.lang` befindet. Weil dieses Paket besonders wichtige Klassen bzw. Typen enthält, dürfen diese verwendet werden, ohne `java.lang` voranzustellen. (Die Java-Bibliothek enthält noch viele weitere Pakete. Deren Verwendung erfordert aber eine `import`-Anweisung oder die explizite Nennung des Paketnamens.)

Die gesamte Anweisung endet wie alle Java-Anweisungen mit einem Strichpunkt. Diesen zu vergessen zählt zu den häufigsten Fehlern, die Java-Einsteiger 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 Code-Beispiele.

Ich bin ein Anhänger der zweiten Variante: Ich setze die fünf Zeilen des Hello-World-Programms im weiteren Verlauf des Buchs einfach als gottgegeben voraus und führe Sie zuerst in die (einfacheren) Grundlagen der Programmierung ein. 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 man eine Menge interessante Beispiele entwickeln und noch mehr lernen kann, auch 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 11, »Klassen«, in dem ich Ihnen die Syntax der objektorientierten Programmierung im Detail vorstelle und das Schlüsselwort `class` *richtig* erkläre. Erst dann werden Sie das scheinbar so simple Hello-World-Programm restlos verstehen.

Zulässige Codeänderungen

Ich habe Sie gebeten, das Programm exakt abzuschreiben. Sie müssen die Syntax von Java einhalten, sonst funktioniert das Programm nicht. Gewisse Freiheiten bleiben Ihnen aber:

- **Klassenname:** Sie können den Klassennamen frei wählen. Anstelle von `HelloWorld` funktioniert auch `MeinErstesProgramm` oder `Xyz` oder `Main`. Der Klassenname darf keine Leer- oder Sonderzeichen enthalten (mit der Ausnahme von _), und er sollte mit einem Großbuchstaben beginnen.

  Wichtig: Der Dateiname *muss* mit dem Klassennamen übereinstimmen! Wenn Sie Ihr Programm mit `class Xyz` beginnen, dann müssen Sie die Datei unter dem Namen `Xyz.java` speichern.

  Eine übliche Schreibweise für längere Namen ist die sogenannte Upper-Camel-Case-Notation: Dabei beginnt jedes neue Wort mit einem Großbuchstaben – also z. B. `MeineLangeKlasse`.

- **Parametername:** Es ist Ihnen auch freigestellt, wie Sie den Parameter in der `main`-Methode nennen. `main(String[] meineparameter)` ist also absolut korrekt. Wie im Klassennamen müssen Sie auf Leer- und Sonderzeichen verzichten. Parameternamen haben üblicherweise einen *kleinen* Anfangsbuchstaben.

- **Abstände und Einrückungen:** Sie dürfen den Code beliebig einrücken, leere Zeilen zwischen den Anweisungen einfügen und an gewissen Stellen sogar Leerraum innerhalb von Anweisungen verwenden. Umgekehrt können Sie möglichst viele Anweisungen, Klammern etc. in einer einzigen Zeile unterbringen.

Die folgenden beiden Listings zeigen zwei Varianten des Hello-World-Programms. Beide sind syntaktisch korrekt, auch wenn der Code merkwürdig aussieht, vor allem in den Augen erfahrener Programmierer.

Die zwei Listings zeigen Ihnen noch ein Sprachelement von Java: // leitet einen *Kommentar* ein, der bis zum Ende der Zeile reicht. Damit können Sie Erläuterungen in den Code einbauen, die vom Java-Compiler ignoriert werden.

```
// Syntaxfreiheiten in Java, Beispiel 1
class Xyz
{
    public static
    void main(String[] abc)
      {
        System . out . println ( "Hello World!" )
        ;
}}
```

```
// Syntaxfreiheiten in Java, Beispiel 2
class Kurz{public static void main(String[] x){
System.out.println("Hello World!");}}
```

Java-Interna

Losgelöst von der Syntax von Java, die in diesem Buch im Vordergrund steht, möchte ich Ihnen an dieser Stelle drei besonders wichtige Begriffe aus der vielschichtigen Welt der Java-Interna vorstellen. Diese Begriffe sind zwar für das eigentliche Programmieren nicht relevant; wenn Sie sie kennen, werden Sie aber besser verstehen, wie Java hinter den Kulissen funktioniert.

- **Byte-Code:** Das Kommando `javac` wandelt den Quelltext Ihres Programms in eine binäre Darstellung um, die *Byte-Code* genannt wird. Diese Umwandlung bezweckt mehrere Dinge:
  - Erstens stellt eine erfolgreiche Umwandlung in Byte-Code sicher, dass Ihr Code syntaktisch korrekt ist.
  - Zweitens kann die nachfolgende Ausführung des Codes dank der optimierten Darstellung des Programms effizienter erfolgen.
  - Und drittens ist der Byte-Code plattformunabhängig. Das bedeutet, dass beispielsweise ein unter Windows kompiliertes Java-Programm später auch unter Linux ausgeführt werden kann.

  Java-Byte-Code wird in Dateien mit der Endung `.class` gespeichert.
- **Java Virtual Machine:** Für die eigentliche Ausführung des Byte-Codes ist das Kommando `java` verantwortlich. Es übergibt den Byte-Code an die sogenannte *Java Virtual Machine* (JVM). Der in der JVM integrierte *Just-in-Time-Compiler* (JIT-Compiler) wandelt den Byte-Code in Maschinen-Code der jeweiligen Hardware-Plattform um. Die JVM mit dem JIT-Compiler ist eine Grundkomponente des Java Runtime Environments (JRE).

  Die JVM stellt dem Java-Programm Hilfsfunktionen zur Verfügung und kümmert sich beispielsweise darum, dass nicht mehr benötigte Objekte durch den sogenannten *Garbage Collector* automatisch aus dem Speicher entfernt werden.
- **Java-Bibliothek:** Beim Programmieren in Java ist es nicht notwendig, das Rad ständig neu zu erfinden. Es existiert ein riesiges Fundament von oft benötigten Grundfunktionen, auf dem Sie aufbauen können. Unzählige Pakete mit Klassen und Methoden der von Sun bzw. Oracle entwickelten Java-Bibliothek warten darauf, von Ihnen entdeckt und eingesetzt zu werden. Die Methode `println` im Hello-World-Programm war dafür das erste Beispiel. Die vollständige Dokumentation der Java-Bibliothek finden Sie hier:

  https://docs.oracle.com/en/java/javase/11

  Damit Sie die Dokumentation richtig deuten können, müssen Sie allerdings zuerst Grundkenntnisse in Java erwerben.

1.7 Die Java-Shell

Java verfügt seit Version 9 über einen *Kommandointerpreter*, die sogenannte Java-Shell (*JShell*, siehe Abbildung 1.6). In der Fachsprache heißt dieses Feature auch *Read-Eval-Print-Loop* (REPL). Die Java-Shell wird durch die Ausführung von `jshell` in `cmd.exe` bzw. in einem Terminalfenster gestartet. Anschließend können Sie einzelne Java-Kommandos eingeben, die dann sofort ausgeführt werden. Um die Zeichenkette »Hello World!« auszugeben, müssen Sie nur das entsprechende `System.out.println`-Kommando eingeben – ohne die Definition einer Klasse und ohne `main`-Methode.

```
> jshell
  Welcome to JShell -- Version 11.0.1
  For an introduction type: /help intro

jshell> System.out.println("Hello World!")
  Hello World!
```

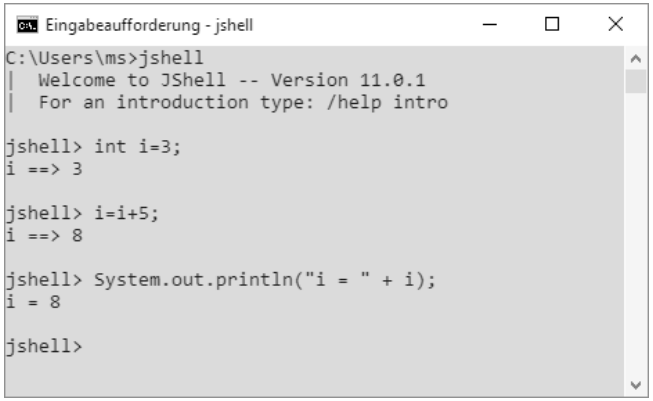

Abbildung 1.6 Die JShell erleichtert das unkomplizierte Testen von Java-Anweisungen.

Abweichend von den Java-Syntaxregeln ist es nicht erforderlich, einzelne Anweisungen mit einem Strichpunkt abzuschließen. Selbst auf

`System.out.println` können Sie in der Regel verzichten. Wenn eine Anweisung ein Ergebnis liefert, wird dieses in jedem Fall ausgegeben:

```
jshell> String s = "abcde"
  s ==> "abcde"

jshell> s.length()
  $8 ==> 5
```

Selbst mehrzeilige Konstrukte sind erlaubt. Das folgende Beispiel zeigt eine Schleife, in der die Variable `i` die Werte 0, 1 und 2 annimmt:

```
jshell> for(int i=0; i<3; i++) {
   ...>   System.out.println(i);
   ...> }
0
1
2
```

Die JShell ist gerade für Java-Einsteiger ungemein praktisch. Sie können damit Sprachfunktionen und Syntaxdetails ohne den Overhead eines vollständigen Java-Programms ausprobieren. Die sofortige Syntaxkontrolle für jede Zeile ist zudem eine große Hilfe.

Fortgeschrittene Funktionen

Wenn Sie längere Code-Passagen eingegeben haben, ist die nachträgliche Veränderung im Zeileneditor mühsam. Abhilfe schafft das Kommando `/edit`, das alle Ihre Eingaben in einem simplen Editor präsentiert. Dort können Sie Änderungen durchführen. Beim Verlassen des Editors mit Accept wird der gesamte Code nochmals durch die Java-Shell ausgeführt.

`/imports` listet auf, welche Bibliotheken »importiert« sind, d. h., aus welchen Paketen der Java-Standardbibliothek Sie unkompliziert Klassen verwenden können. Die JShell unterscheidet sich von normalem Java-Code auch dadurch, dass einige grundlegende Pakete standardmäßig importiert werden, z. B. `java.io.*`, `java.math.*` und `java.util.*`. (Was Importe

sind, erfahren Sie in Abschnitt 2.2, »Java-Syntax«, und noch detaillierter in Kapitel 19, »Pakete, Bibliotheken und Module«.)

Mit `/save dateiname` können Sie alle bisherigen Eingaben in einer Textdatei speichern. In einer späteren JShell-Sitzung können Sie diese Eingaben mit `/open dateiname` wieder laden und ausführen. Einen Überblick über noch mehr eingebaute JShell-Kommandos erhalten Sie mit `/help`. Noch mehr Möglichkeiten dokumentiert das JShell-Handbuch:

https://docs.oracle.com/en/java/javase/11/jshell

Kürzel	Bedeutung
Cursortasten	Blättert durch die bisherigen Eingaben.
⇆	Vervollständigt die Eingabe.
⇧+⇆	Zeigt die Online-Hilfe zum Schlüsselwort.
Strg+D	Beendet die JShell.

Tabelle 1.1 Wichtige JShell-Tastenkürzel

1.8 Hello IntelliJ IDEA

Dies ist ein Java-Buch, kein IntelliJ-IDEA-Buch! Auch wenn sich dieser Abschnitt primär um Kommandos, Dialoge und Fenster der IntelliJ-Entwicklungsumgebung dreht, werde ich mich im weiteren Verlauf des Buchs auf Java konzentrieren und die IntelliJ IDEA nur noch am Rande erwähnen. Für alle, die IntelliJ als Entwicklungsumgebung verwenden, fasst Anhang A, »Crashkurs IntelliJ IDEA«, die wichtigsten Begriffe und Techniken zusammen.

Lassen Sie sich nicht von der Länge dieses Abschnitts abschrecken. Die Beschreibung der Grundfunktionen der IntelliJ IDEA nimmt zwar viel Platz in Anspruch, aber sobald Sie sich einmal an das Programm gewöhnt haben, können Sie das komplette Hello-World-Programm in 30 Sekunden erstellen und ausführen. (Nicht gelogen, ich habe es mit der Stoppuhr getestet!)

JDK einrichten

Wenn Sie im Startdialog von IntelliJ Create New Project ausführen, müssen Sie im nächsten Dialog das passende Project SDK für das neue Projekt auswählen (siehe Abbildung 1.7, links). Je nach Installation kann es sein, dass das Listenfeld bereits automatisch den richtigen Eintrag enthält (z. B. »11« für die Java-Version 11). Wenn ein entsprechender Eintrag aber fehlt, weiß IntelliJ nicht, wo sich die JDK auf Ihrer Festplatte oder SSD befindet.

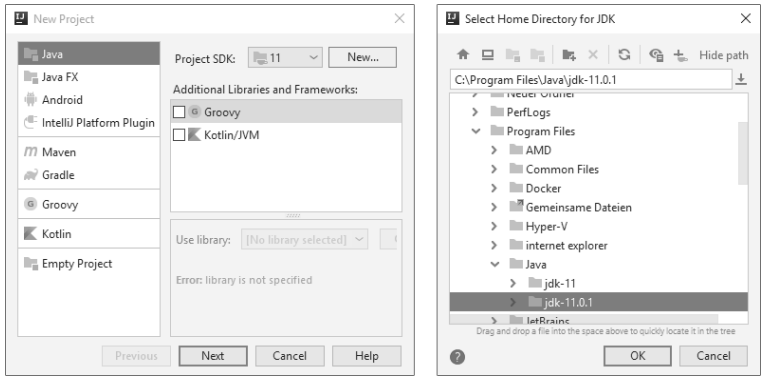

Abbildung 1.7 Beim Start des ersten Projekts muss das SDK eingerichtet werden.

In diesem Fall müssen Sie einmalig den Button New und dann den Listeneintrag JDK anklicken und im nächsten Dialog das richtige Verzeichnis auswählen (siehe Abbildung 1.7, rechts). Die üblichen Orte sind:

- Windows: `C:\Program Files\Java\jdk-11`
- macOS: `/Library/Java/JavaVirtualMachines/jdk-11.jdk/Contents/Home`
- Linux: `/usr/lib/jvm/java-11.0-openjdk-amd64` oder `/usr/lib/jvm/jre-11-openjdk`

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.

Die SDK-Einstellung ist notwendig, weil die IntelliJ IDEA neben Java/JDK auch andere Sprachen mit deren *Software Development Kits* unterstützt. Die Verwaltung der SDKs erfolgt mit File • Other Settings • Default Project Structure. Dieses Buch berücksichtigt aber nur Java.

Neues Projekt starten

In der IntelliJ IDEA gilt jedes Java-Programm als eigenes »Projekt«. Seine Dateien werden in einem Projektverzeichnis gespeichert. Jedes Projekt hat also sein eigenes Verzeichnis.

Um ein neues Projekt zu beginnen, klicken Sie im Startdialog auf Create New Project, wählen den Projekttyp Java und das Java-SDK in der gewünschten Version.

Im nächsten Dialogblatt aktivieren Sie die Option Create project from template. Diese Option bewirkt, dass die Entwicklungsumgebung den Großteil des Hello-World-Codes bereits in die Code-Datei des neuen Projekts einfügt. Sie sparen sich also etwas Tipparbeit.

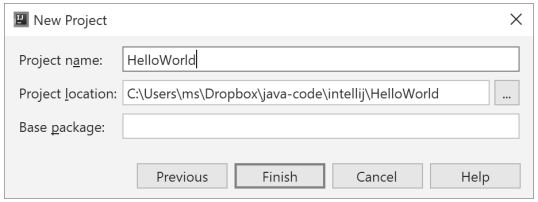

Abbildung 1.8 Projekteinstellungen

Im dritten und letzten Dialogblatt müssen Sie noch drei Parameter einstellen: den Projektnamen, den Speicherort und das Base package (siehe Abbildung 1.8). Die ersten zwei Parameter bedürfen keiner Erklärung. Das Base package gibt einen Namen an, der Ihren eigenen Klassen vorangestellt wird. Ich empfehle Ihnen, den Parameter für alle Projekte aus diesem Buch einfach leer zu lassen und die folgende Info-Box vorerst zu ignorieren!

Hintergründe zu Base Packages

Bei einem großen Projekt mit vielen Klassen und externen Bibliotheken ist das Base package sehr wohl zweckmäßig: Dann schafft die Einstellung Klarheit, welche Klasse zu welchem Teil des Projekts gehört.

Die Angabe des Base package gilt als »guter Java-Stil«. Wenn Sie sich dieser Konvention unterwerfen möchten, geben Sie die Zeichenkette ohne Leer- und Sonderzeichen (einzig _ ist erlaubt) in dieser Form an:

```
de.meine_domain.mein_projekt
```

Die ersten der durch Punkte getrennten Bestandteile entsprechen dem Domänennamen Ihrer Webseite (oder dem Ihrer Firma oder Schule) in umgekehrter Reihenfolge. Danach folgt der Projektname. Das Ziel dieser Angabe ist es, dass sich Ihre Klassen aufgrund des Base-Package-Namens weltweit von allen anderen gleichnamigen Klassen unterscheiden lassen.

Die Angabe der Base-Package-Zeichenkette hat zwei Konsequenzen: Zum einen werden die Code-Dateien Ihres Projekts in Unterverzeichnissen gespeichert (z. B. in `src/de.meine_domain.mein_projekt` anstatt einfach in `src`), zum anderen enthält jede Code-Datei die Anweisung `package de.meine_domain.mein_projekt`. Mehr Details zu Java-Paketen und -Bibliotheken folgen in Kapitel 19.

Endlich Code

Wenn Sie beim Einrichten des neuen Projekts wie empfohlen die Option Create project from template aktiviert haben, dann enthält das neue Projekt bereits die Klassendatei `Main.java` mit allen für *Hello World!* erforderlichen Code-Zeilen außer der `System.out.println`-Anweisung. Diese fügen Sie hinzu (siehe Abbildung 1.9). Anschließend kompilieren und starten Sie das Programm durch einen Klick auf den grünen Pfeil-Button bzw. durch das Menükommando Run • Run Main.

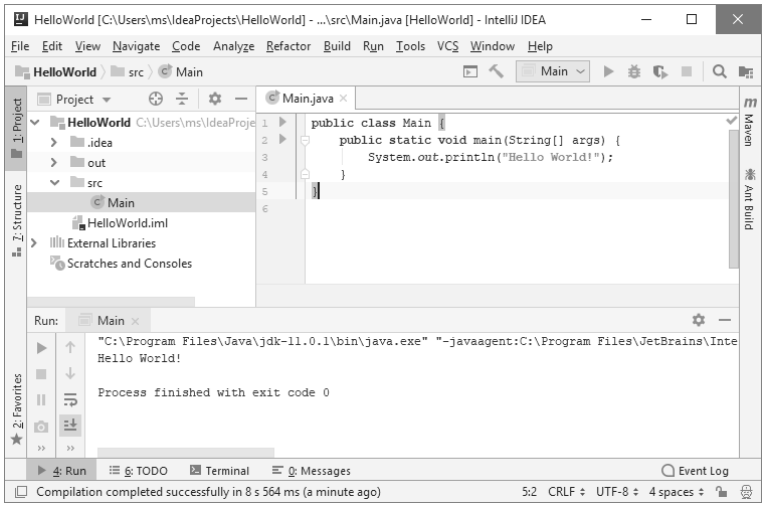

Abbildung 1.9 Ein typisches Setup in der IntelliJ IDEA: links die Projektübersicht, rechts eine Code-Datei, unten die Ausgaben eines Programms.

Beispieldateien

Alle Beispieldateien zu diesem Buch gibt es in zweifacher Ausfertigung: einmal als pure `*.java`-Dateien zum direkten Kompilieren und einmal als IntelliJ-Projekte für die Java-Version 11. Wenn auf Ihrem Rechner eine andere Java-Version installiert ist (älter oder neuer), dann zeigt IntelliJ nach dem Laden eines Projekts die Fehlermeldung *Projekt SDK not defined* an. Komponenten des Projekts sind mit roten Wellenlinien unterlegt, der Code kann nicht ausgeführt werden.

Dieses Problem lässt sich leicht lösen: Mit FILE • PROJEKT STRUCTURE gelangen Sie in einen Dialog, in dem Sie diverse Projekteinstellungen verändern können (siehe Abbildung 1.10). Dort stellen Sie als PROJECT SDK die auf Ihrem Rechner installierte Java-Version ein. Den PROJEKT LANGUAGE LEVEL können Sie auf Version 11 belassen – es sei denn, Sie verwenden eine ältere Version: In diesem Fall stellen Sie dieselbe Version wie beim SDK ein. (Die meisten Beispielprogramme dieses Buchs sind auch zu älteren Java-Versionen kompatibel.)

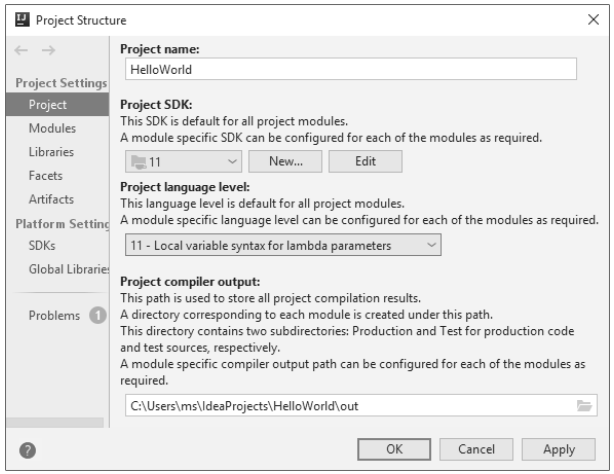

Abbildung 1.10 In den Projekteinstellungen können Sie die Java-Version einstellen.

IntelliJ(gent) oder nicht – das ist hier die Frage …

Zuletzt noch eine persönliche Einschätzung: Aus meiner Sicht ist eine Entwicklungsumgebung wie die IntelliJ IDEA eher ein notwendiges Übel als die ideale Umgebung für Java-Einsteiger.

Klar, die IntelliJ IDEA bietet wie die anderen Java-Entwicklungsumgebungen viele Eingabeerleichterungen und vermeidet eine Menge Fehler schon im Vorfeld. Die IntelliJ IDEA ist aber ein sehr komplexes Programm, und gerade Einsteiger verlieren sich leicht in den riesigen Menüs und den unzähligen Teilfenstern, die wie von Geisterhand auftauchen und verschwinden. Es besteht die Gefahr, dass Sie viel Zeit mit den Eigentümlichkeiten der IntelliJ IDEA vergeuden und sich nicht auf das eigentliche Erlernen von Java konzentrieren.

Sie machen nichts verkehrt, wenn Sie Ihre ersten Experimente in der Java-Shell durchführen. Erste Progamme verfassen Sie mit einem simplen Editor und kompilieren diese dann manuell. Ein gut geeigneter Editor ist Notepad++. Wenn Sie etwas Vertrauen zu Java gefasst haben, ist es immer

noch früh genug, um sich mit der IntelliJ IDEA oder einer anderen Entwicklungsumgebung anzufreunden.

1.9 Wiederholungsfragen

- **W1:** Wozu brauche ich die IntelliJ IDEA?
- **W2:** Worin besteht der Unterschied zwischen JRE und JDK?
- **W3:** Was ist der Unterschied zwischen `java` und `javac`?
- **W4:** Wo ist die EXE-Datei meines Java-Programms?
- **W5:** Wozu dient die Java Virtual Machine?
- **W6:** Wozu dient die Java-Shell?

Kapitel 4

Operatoren

Im Ausdruck `a = b + c` gelten die Zeichen `=` und `+` als Operatoren. Dieses Kapitel stellt Ihnen alle Java-Operatoren vor – von den simplen Operatoren für die Grundrechenarten bis hin zu etwas diffizileren Operatoren zur Bitarithmetik.

4.1 Überblick

Java kennt Operatoren zur Zuweisung von Variablen, zum Vergleich von Werten, zur Berechnung mathematischer Ausdrücke etc. (siehe Tabelle 4.1). Seit Java 8 dienen `->` zur Formulierung von Lambda-Ausdrücken sowie `::` zur Angabe von Referenzen auf Methoden. Diese Operatoren stelle ich Ihnen in Kapitel 14, »Lambda-Ausdrücke«, näher vor.

Priorität	Operator	Bedeutung
1 →	`()`	Methodenaufruf
	`[]`	Zugriff auf Felder (Arrays)
	`.`	Zugriff auf Objekte, Methoden etc.
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 4.1 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 4.1 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 lesbarer!

4.2 Details und Sonderfälle

Natürlich müssen Sie die Operator-Tabelle 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;
java.awt.Point p;
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 Integer-Berechnungen 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;
System.out.println(i % 5);    // liefert 2
double d = 22.3;
System.out.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:

http://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 sowie für Berechnungen.

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

Vergleiche

Zwei Zahlen können Sie mit `==`, `!=`, `<`, `<=`, `>` und `>=` vergleichen (gleich, ungleich, kleiner, kleiner-gleich, größer, größer-gleich). Derartige Vergleiche benötigen wir in Kapitel 5, »Verzweigungen und Schleifen«. Vorweg ein Beispiel:

```
int a=3, b=5;
if(a == b) {
  System.out.println("a und b sind gleich groß.");
} else {
  System.out.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 11.

Die Operatoren <, <=, > und >= stehen für allgemeine Objekte nicht zur Verfügung. Wenn eine Klasse die Schnittstelle `Comparable` implementiert, können Sie mit `a.compareTo(b)` feststellen, welches Objekt *kleiner* bzw. *größer* ist. Was eine Schnittstelle ist und wie Sie selbst `Comparable` implementieren können, lernen Sie in Kapitel 12, »Vererbung und Schnittstellen«.

```
if(a.compareTo(b)==0) { ... }  // wenn a und b gleich
                               // groß sind
if(a.compareTo(b)<0)  { ... }  // wenn a kleiner b gilt
if(a.compareTo(b)>0)  { ... }  // wenn a größer b gilt
```

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, aber fast immer falsch. Der Inhalt der Zeichenketten wird dabei nicht berücksichtigt. Es ist durchaus möglich, dass zwei Zeichenketten (`String`-Objekte) dieselben Zeichen enthalten, sich aber an unterschiedlichen Orten im Speicher befinden.

```
String s1 = "abc";
String s2 = "abc".toUpperCase();
System.out.println(s2 == "ABC");       // liefert false
System.out.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 4.2 Verknüpfung von Bedingungen

Java sieht für die Verknüpfung mehrerer Bedingungen eine ganze Palette boolescher Operatoren vor (siehe Tabelle 4.2). 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` muss 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. Gleichzeitig vermeidet sie auch 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 links (entspricht einer Division durch 2^n), << verschiebt entsprechend nach rechts (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 5, »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 11, »Klassen«.

4.3 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 Code-Lösungen immer um Lösungs*vorschläge* handelt. Zu fast allen Aufgabenstellungen gibt es viele Lösungswege.

B.1 Kapitel 1, »Hello World!«

W1: Wozu IntelliJ IDEA?

Sie brauchen die IntelliJ IDEA gar nicht, insbesondere nicht für die in diesem Buch präsentierten Beispiele. Die IntelliJ IDEA ist eine Entwicklungsumgebung, die das Verfassen, Testen und Dokumentieren von Java unterstützt. Insofern macht IntelliJ Ihnen das Leben als Java-Entwickler in vielerlei Hinsicht einfacher – und das umso mehr, je größer Ihre Projekte werden.

Leider ist die Komplexität von IntelliJ für viele Programmiereinsteiger abschreckend. Wenn Sie IntelliJ nicht einsetzen möchten, können Sie alle Java-Programme aus diesem Buch in einem beliebigen Editor verfassen, manuell mit `javac` kompilieren und mit `java` ausführen.

Anstelle von IntelliJ können Sie auch andere Entwicklungsumgebungen verwenden, z. B. Eclipse oder NetBeans.

W2: JRE versus JDK

Das *Java Runtime Environment* (JRE) erlaubt nur das Ausführen von fertigen Java-Programmen.

Um selbst Java-Programme entwickeln und kompilieren zu können, benötigen Sie das *Java Development Kit* (JDK).

W3: »java« versus »javac«

Das Kommando `java` dient zur Ausführung kompilierter Java-Programme.

Mit dem Compiler `javac` wandeln Sie Ihren Java-Quelltext in ein binäres, plattformunabhängiges Format um, den sogenannten Byte-Code.

W4: Wo ist die EXE-Datei?

Der Compiler `javac` erzeugt keine EXE-Dateien, sondern Byte-Code. Dieser Code muss durch den Java-Interpreter `java` ausgeführt werden.

Diese auf den ersten Blick umständliche Vorgehensweise hat den großen Vorteil, dass ein einmal kompiliertes Java-Programm unter *jedem* von Java unterstützten Betriebssystem ausgeführt werden kann. EXE-Dateien wären auf die Windows-Welt beschränkt.

W5: Java Virtual Machine

Die JVM ist für die Ausführung von Java-Byte-Code zuständig. Der Code wird dabei zuerst mit einem JIT-Compiler (Just-In-Time-Compiler) in Maschinen-Code für die CPU des Rechners umgewandelt.

W6: Java-Shell

Die Java-Shell ermöglicht es, einfache Java-Anweisungen auszuprobieren, ohne gleich ein ganzes Programm zu schreiben, das den Anforderungen der Java-Syntax entspricht. Die Java-Shell ist damit gerade für Einsteiger ein praktisches Hilfsmittel.

B.2 Kapitel 2, »Java-Crashkurs«

W1: Methoden

Methoden sind in sich abgeschlossene Code-Einheiten. 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, um mit einer bestimmten Art von Daten zu arbeiten. 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: Regeln für ».java«-Dateinamen

Der Dateiname ergibt sich aus dem Namen der Klasse und der Endung `.java`. Es ist üblich, den Namen von Klassen und anderen Typen (Schnittstellen, Enumerationen) mit einem Großbuchstaben zu beginnen. Diese Konvention gilt somit auch für die Namen der Code-Dateien.

W5: 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()`.

W6: 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 Code-Blocks wird durch geschwungene Klammern markiert.

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

W7: 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.

W8: import

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

B.3 Kapitel 3, »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 diese 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` infrage (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;
System.out.println(x/y);
```

W4: Variablen müssen initialisiert werden

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


```
int x, y, z;
x = 3;
y = x + z;
z = 5;
System.out.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;
System.out.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-kap03-quadrat`.

B.4 Kapitel 4, »Operatoren«

W1: Restwertoperator

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

```
System.out.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 Postix-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 `rechenfunktion(x)` den Wert 0 oder eine negative Zahl liefert, dann wird `rechenfunktion(y)` nicht aufgerufen. Das ist nicht notwendig, weil `&&` nur dann `true` liefern kann, wenn beide Teilergebnisse `true` sind.


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

Auf einen Blick

Inhalt