

Daniel Braun

JAVA

Schnelleinstieg

Programmieren lernen in 14 Tagen

Einfach und ohne Vorkenntnisse

Zahlreiche
Praxisbeispiele
und Übungen



mitp

Inhalt

Einleitung

E.1	Programmieren lernen in 14 Tagen	13
E.2	Der Aufbau des Buches	13
E.3	Programmtexte und Lösungen zum Download	14
E.5	Fragen und Feedback	14



Erste Schritte mit Java

1.1	Programmiersprachen	15
1.2	Besonderheiten von Java	17
1.3	Installation und Einrichtung	18
1.3.1	Linux	19
1.3.2	macOS	19
1.3.3	Windows	19
1.3.4	Installation testen	21
1.4	Entwicklungsumgebungen	22
1.4.1	Eclipse	23
1.4.2	IntelliJ IDEA	31



Das erste Programm – »Hallo Welt!«

2.1	Datei erstellen	37
2.1.1	Eclipse	38
2.1.2	IntelliJ IDEA	38
2.2	Quelltext	40
2.2.1	Hallo Welt	40
2.2.2	Syntax	41
2.2.3	Kommentare	43

2.3	Kompilieren	44
2.3.1	Ohne IDE	44
2.3.2	Eclipse	44
2.3.3	IntelliJ IDEA	45
2.4	Fehler finden	46
2.5	Ausführen	49
2.5.1	Ohne IDE	49
2.5.2	Eclipse	50
2.5.3	IntelliJ IDEA	51
2.6	Übungen	55



Variablen

3.1	Funktionsweise von Variablen	57
3.2	Typen	60
3.2.1	Zahlen	60
3.2.2	Texte	63
3.2.3	Wahrheitswerte	64
3.3	Operatoren	65
3.3.1	Arithmetische Operatoren	65
3.3.2	String-Konkatenation	66
3.3.3	Boolesche Operatoren	67
3.4	Typumwandlung	69
3.5	Nutzereingaben verarbeiten	74
3.6	Übungen	76



Verzweigungen und Schleifen

4.1	Verzweigungen	77
4.1.1	if-Verzweigung	77
4.1.2	if-else und else if	80
4.1.3	Vergleichsoperatoren	83
4.1.4	case-Verzweigung	84

4.2	Schleifen	86
4.2.1	while-Schleife	88
4.2.2	do-while-Schleife	92
4.2.3	for-Schleife	93
4.2.4	Verschachtelung von Schleifen	94
4.3	Übungen	96



Funktionen

5.1	Grundlagen	99
5.2	Parameter	102
5.3	Rückgabewerte	103
5.4	Rekursion	104
5.5	Übungen	106



Objektorientierung

6.1	Die ganze Welt ist ein Objekt	107
6.1.1	Definition von Klassen	108
6.1.2	Instanzen und Objekte erzeugen	109
6.1.3	Attribute und Funktionen	111
6.2	Statische und nicht-statische Funktionen und Variablen	112
6.3	Verwendung mehrerer Klassen	114
6.4	Zugriffsrechte	116
6.5	Vererbung	118
6.6	Umwandlung zwischen verwandten Klassen (Typecasting)	121
6.7	Abstrakte Methoden und Klassen	123
6.8	Einlesen von Personendaten	125
6.9	Übungen	132



Arrays und Listen

7.1	Arrays	135
7.1.1	Deklaration	135
7.1.2	Lesen und Schreiben	136
7.1.3	Arrays und Schleifen	138
7.1.4	Personenverwaltung mit Arrays	139
7.1.5	Umwandlung zwischen Strings und Arrays	143
7.1.6	Mehrdimensionalität	145
7.1.7	Start-Parameter	147
7.2	Listen	149
7.2.1	Arten von Listen	150
7.2.2	Deklaration	150
7.2.3	Elemente hinzufügen	151
7.2.4	Elemente lesen	152
7.2.5	Elemente löschen	152
7.2.6	Elemente ändern	153
7.2.7	Listen durchlaufen	154
7.2.8	Unterschied zwischen ArrayList und LinkedList	156
7.2.9	Personenverwaltung mit Liste	159
7.3	Übungen	161



Fehlerbehandlung

8.1	Arten von Fehlern	163
8.2	Fehler abfangen	164
8.2.1	Umwandlungsfehler	166
8.2.2	Index-Fehler	169
8.2.3	Andere Fehler	169
8.2.4	Fehlerbehandlung bei Funktionen	171
8.3	Mehrere Arten von Fehlern abfangen	174
8.4	finally-Block	174
8.5	Übungen	175

9

Dateisystem

9.1	Textdateien schreiben	177
9.2	Textdateien lesen	180
9.3	Dateien löschen und umbenennen	181
9.4	Ordner	182
9.4.1	Erstellen	182
9.4.2	Löschen	183
9.4.3	Umbenennen	183
9.4.4	Inhalt anzeigen	183
9.5	Objekte speichern	184
9.5.1	CSV-Dateien	184
9.5.2	Binäre Dateien	190
9.6	Übungen	193

10

Externe Bibliotheken

10.1	JSON-Bibliothek	195
10.1.1	Das Format	196
10.1.2	Bibliothek einbinden	197
10.1.3	Bibliothek verwenden	199
10.2	Eigene .jar-Dateien erstellen	208
10.2.1	Bibliotheken	209
10.2.2	Ausführbare .jar-Dateien	215
10.3	Abhängigkeiten in großen Softwareprojekten	218
10.4	Übungen	219

11

Grafische Benutzerschnittstellen

11.1	Fenster	222
11.2	Textfelder	223
11.3	Buttons	224

11.4	Layout	225
11.4.1	BorderLayout	225
11.4.2	FlowLayout	227
11.4.3	GridLayout	228
11.5	Grafische Benutzerschnittstelle für die Personenverwaltung	230
11.5.1	Personendaten einlesen	230
11.5.2	Personendaten anzeigen	232
11.5.3	Personendaten bearbeiten	235
11.5.4	Personendaten exportieren	236
11.6	Übungen	238



Multitasking

12.1	Threads	240
12.2	Data Race	242
12.3	Synchronisierung	244
12.4	Locks	245
12.5	Conditions	247
12.6	Threads beenden	248
12.7	Übungen	250



Datenbanken

13.1	Datenbank einrichten	251
13.2	Verbindung herstellen	252
13.3	Tabelle erstellen	253
13.4	Daten hinzufügen	255
13.5	Daten lesen	255
13.6	Daten ändern	257
13.7	Nachteile und Alternativen	259
13.8	Übungen	259



REST-Schnittstellen

14.1	Funktionsweise von REST-Schnittstellen	264
14.2	Beispiele für das Ansprechen einer REST-Schnittstelle	264
14.2.1	GET	265
14.2.2	POST	265
14.2.3	PUT	266
14.2.4	DELETE	267
14.3	Implementierung in Java	267
14.3.1	Server-Konfiguration	267
14.3.2	Router	269
14.3.3	REST-Schnittstellen testen	271
14.4	Schnittstelle für Personendaten	273
14.4.1	Initialisierung	273
14.4.2	POST	276
14.4.3	GET	278
14.4.4	DELETE	280
14.4.5	PUT	281
14.5	Übungen	283
14.6	Nächste Schritte	283

Stichwortverzeichnis

Einleitung

E.1 Programmieren lernen in 14 Tagen

Mit diesem Buch haben Sie sich für einen einfachen, praktischen und fundierten Einstieg in die Welt der Programmierung entschieden. Sie lernen ohne unnötigen Ballast alles, was Sie wissen müssen, um Java effektiv für Projekte in Ihrem Berufs- und Interessensgebiet einzusetzen.

Wenn Sie Zeit genug haben, können Sie jeden Tag ein neues Kapitel durcharbeiten und so innerhalb von zwei Wochen Programmieren lernen. Alle Erklärungen sind leicht verständlich formuliert und setzen keine Vorkenntnisse voraus. Am besten lesen Sie das Buch neben der Computer-Tastatur und probieren die Programmbeispiele und Übungen gleich aus. Sie werden schnell erste Erfolge erzielen und Freude an der Programmierung finden.

E.2 Der Aufbau des Buches

Das Buch beginnt mit den Grundlagen: Installation von Java, Nutzung verschiedener Entwicklungsumgebungen und Formulierung einfacher Anweisungen. Die Kapitel bauen aufeinander auf. Sie lernen Schritt für Schritt, wie man Daten lädt, verarbeitet und speichert, sowohl in Dateien als auch in Datenbanken, und erhalten eine Einführung in die Verwendung von Funktionen, objektorientierte Programmierung, die Gestaltung von grafischen Benutzeroberflächen und Programmieren von nebenläufigen Anwendungen. Das letzte Kapitel schließlich beschäftigt sich mit der Implementierung einer Web-Schnittstelle in Java und zeigt Ihnen einige Möglichkeiten, wie Sie nach dem Schnelleinstieg Ihre Programmierkenntnisse weiterentwickeln können.

Ihr Tagespensum schließt mit praktischen Programmier-Übungen, in denen Sie Ihr neu gewonnenes Wissen vertiefen können. Die Lösungen zu diesen Übungen, die relativ viel Programmtext enthalten, stehen in einem Online-Kapitel zum Download zur Verfügung. Mehr dazu im nächsten Abschnitt.

Am Ende des Buches finden Sie ein Stichwortverzeichnis, das Ihnen hilft, bestimmte Themen im Buch schneller zu finden.

E.3 Programmtexte und Lösungen zum Download

In diesem Buch wird eine vollständige Personenverwaltung mit Anbindung an eine Datenbank, grafischer Benutzeroberfläche und Web-Schnittstelle entwickelt, um die gezeigten Techniken zu demonstrieren und anzuwenden.

Der Code der Personenverwaltung in den unterschiedlichen Entwicklungsstadien sowie die Lösungen zu den Übungen stehen Ihnen auf der Webseite des Verlags unter www.mitp.de/0392 zum Download zur Verfügung.

Dort finden Sie außerdem ein praktisches Glossar mit den wichtigsten Fachbegriffen.

E.5 Fragen und Feedback

Unsere Verlagsprodukte werden mit großer Sorgfalt erstellt. Sollten Sie trotzdem einen Fehler bemerken oder eine andere Anmerkung zum Buch haben, freuen wir uns über eine direkte Rückmeldung an lektorat@mitp.de.

Falls es zu diesem Buch bereits eine Errata-Liste gibt, finden Sie diese unter www.mitp.de/0392 im Reiter DOWNLOADS.

Wir wünschen Ihnen viel Erfolg und Spaß bei der Programmierung mit Java!

Daniel Braun und das mitp-Lektorat



Das erste Programm – »Hallo Welt!«

Nachdem Sie im letzten Kapitel alle Vorbereitungen getroffen und Ihre Entwicklungsumgebung eingerichtet haben, kann es nun endlich mit dem Programmieren losgehen. Wir starten mit einem denkbar einfachen Programm, dessen einzige Aufgabe es sein wird, den Text »Hallo Welt!« auf dem Bildschirm auszugeben.



»Hallo Welt!«-Programme haben unter Programmierern Tradition. Bereits seit 1974 werden sie als einfaches Beispiel für die Nutzung einer Programmiersprache verwendet.

2.1 Datei erstellen

Zunächst müssen Sie eine Datei anlegen, in der das Programm gespeichert werden soll. Dateien, die Java-Programmcodes enthalten, tragen üblicherweise die Endung `.java`. Der Dateiname darf dabei nur aus Buchstaben und Zahlen bestehen, Satz- oder Sonderzeichen sowie Umlaute sind nicht zulässig.

Wenn Sie keine IDE benutzen, können Sie in einem Texteditor Ihrer Wahl eine Datei anlegen, die zum Beispiel den Namen `HalloWelt.java` trägt. Im weiteren Verlauf werden wir der Einfachheit halber davon ausgehen, dass die Datei unter Windows im Ordner `C:\Java-Schnelleinstieg`, unter Linux im Ordner `/home/benutzername/Java-Schnelleinstieg` und unter macOS im Ordner `/Users/benutzername/Java-Schnelleinstieg` abgelegt ist. Sie können die Datei auch unter einem beliebigen anderen Pfad ablegen, müssen dann bei den nächsten Schritten aber darauf achten, jeweils den passenden verwendeten Pfad anzugeben.

2.1.1 Eclipse

In Eclipse können Sie in Ihrem leeren Projekt eine neue Datei erstellen, indem Sie in der oberen Menüleiste das Menü **DATEI** öffnen, dort den Eintrag **NEU** auswählen und dann auf **DATEI** klicken. Daraufhin öffnet sich der in Abbildung 2.1 gezeigte Dialog. Dort müssen Sie nun nur noch den Dateinamen angeben und nach einem Klick auf **FERTIGSTELLEN** wird die Datei erzeugt. Das Projekt beinhaltet bereits einen automatisch generierten Ordner mit den Namen `src`, in dem die Datei abgelegt wird. Was es mit diesem Namen auf sich hat, das erfahren Sie in Abschnitt 2.2.

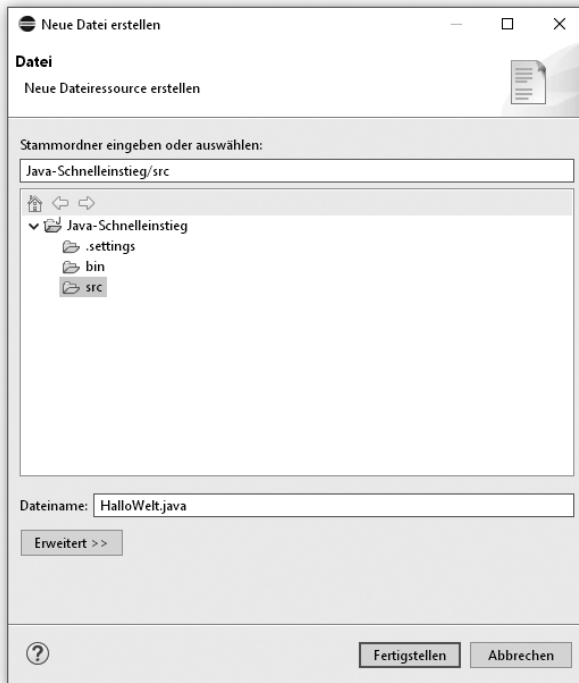


Abb. 2.1: Erstellen einer neuen Datei in Eclipse

2.1.2 IntelliJ IDEA

Um in IntelliJ IDEA eine neue Datei in Ihrem leeren Projekt zu erstellen, klicken Sie zunächst, wie in Abbildung 2.2 gezeigt, in der *Project*-Ansicht auf der linken Seite des Projekts auf den automatisch angelegten Ordner `src`. So wird

die neue Datei automatisch in diesem Ordner erstellt. Was es damit auf sich hat, dazu mehr in Abschnitt 2.2.

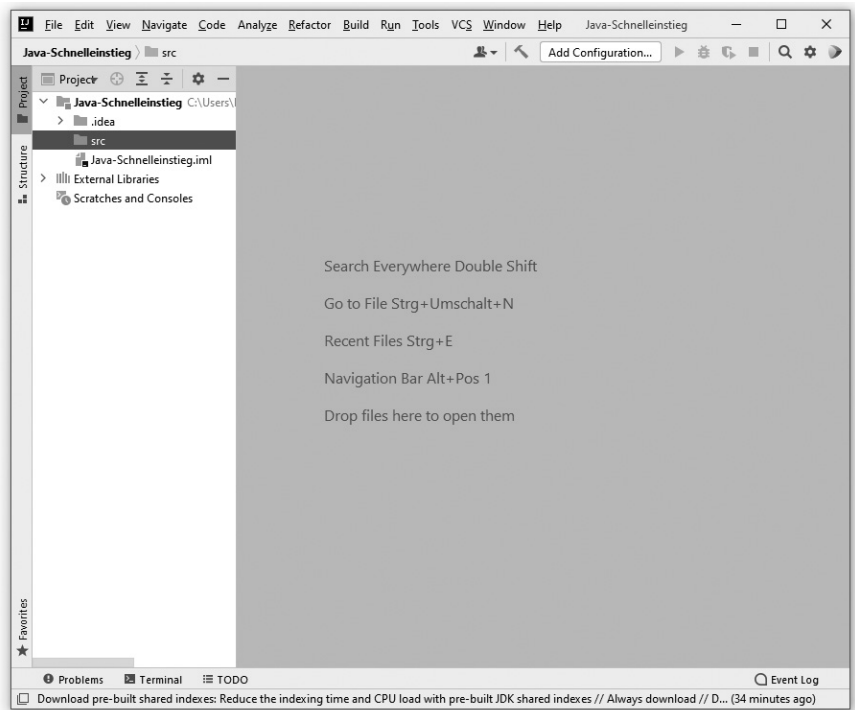


Abb. 2.2: Leeres Projekt mit ausgewähltem `src`-Ordner

Nun öffnen Sie das **FILE**-Menü in der oberen Menüleiste, wählen dort den Eintrag **NEW** aus und klicken dann auf **FILE**. Daraufhin öffnet sich ein kleines Dialogfenster, das in Abbildung 2.3 gezeigt ist. Hier müssen Sie nun nur den Dateinamen eingeben und mit Drücken der **[Enter]**-Taste bestätigen. Danach wird die Datei angelegt.

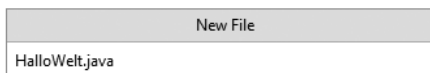


Abb. 2.3: Erstellen einer neuen Datei in IntelliJ IDEA

2.2 Quelltext

Nun können Sie die neu erstellte Datei mit dem Programmcode für Ihr erstes Java-Programm befüllen. Gerade einmal fünf Zeilen umfasst der Text des Programms, den Sie in Listing 2.1 finden. Der Text, aus dem ein Programm besteht, wird auch *Quelltext* oder *Quellcode* genannt. Im Englischen wird er als *source code* bezeichnet, was häufig auch mit *src* abgekürzt wird. Die von der IDE automatisch erstellten *src*-Ordner dienen also als Speicherplatz für den Quelltext des Programms.



Der Umfang eines Programms beziehungsweise seines Quelltexts wird häufig in (Code-)Zeilen angegeben. Im Englischen ist auch die Abkürzung *LoC* für *Lines of Code* gebräuchlich.

2.2.1 Hallo Welt

Das »Hallo Welt!«-Programm in Listing 2.1 besteht zwar aus fünf Zeilen, aber es gibt eigentlich nur eine Anweisung, die ausgeführt wird, und die befindet sich in der dritten Zeile. Diese Anweisung lautet `System.out.println()`, und was sie tut, ist, den Text auszugeben, der innerhalb der Klammern in Anführungszeichen steht. Beendet wird die Zeile, wie die meisten Anweisungen in Java, mit einem Semikolon.

Die anderen Zeilen bilden die Grundstruktur eines Java-Programms, die, unabhängig vom eigentlichen Zweck des Programms, immer gleich oder zumindest sehr ähnlich ist.

```
001 class HalloWelt {
002     public static void main(String[] args) {
003         System.out.println("Hallo Welt!");
004     }
005 }
```

Listing 2.1: »Hallo Welt!«-Programm

In Zeile 1 befindet sich der Namen des Programms, genauer der Name der Klasse, aber dazu später mehr. Schon jetzt wichtig ist aber, dass dieser Name immer auch dem Dateinamen entsprechen muss. Heißt die Datei also `HalloWelt.java`, so muss die erste Zeile `class HalloWelt {` lauten. Heißt die Datei dagegen `Kundenverwaltung.java`, so muss die erste Zeile `class Kundenverwaltung {` heißen. Entsprechend gelten hier bei der Namensvergabe dieselben Regeln wie beim Dateinamen: nur Buchstaben und Zahlen, keine Leer-, Sonder- oder Satzzeichen.

Auffällig im Quellcode sind die geschweiften Klammern (`{, }`). Sie markieren in Java sogenannte *Anweisungsblöcke*, also Teile des Quelltexts, die zusammengehören. Das Programm `HalloWelt` startet zum Beispiel an der geöffneten geschweiften Klammer und endet an der geschlossenen geschweiften Klammer.

Listing 2.1 enthält noch einen zweiten Anweisungsblock, der in der zweiten Zeile beginnt. Vereinfacht ausgedrückt teilt dieser dem Computer mit, wo der Anfang des Programms ist. Was genau es damit auf sich hat, werden wir uns zu einem späteren Zeitpunkt im Detail anschauen.

Automatische Vervollständigung

Eine der Komfortfunktionen, die Eclipse und IntelliJ IDEA beim Programmieren bieten, ist die automatische Vervollständigung von Befehlen. Sobald Sie zum Beispiel die Zeichen »Sys« eingeben, öffnet sich in beiden IDEs ein Fenster, wie in Abbildung 2.4, das mögliche Vervollständigungen vorschlägt. Das hilft, Tippfehler zu vermeiden, und zeigt, welche Befehle überhaupt zur Verfügung stehen.



Abb. 2.4: Autovervollständigung in IntelliJ IDEA

2.2.2 Syntax

All diese Regeln – also wie ein Befehl geschrieben wird, wo ein Semikolon verwendet werden muss und wo eine Klammer – bilden die *Syntax* der Programmiersprache Java.

Das menschliche Gehirn ist sehr gut darin, Syntaxfehler in unserer Sprache automatisch zu beheben. Sie kennen vielleicht das Spiel mit dem vertauschten Bauchstaben: »In wlecehr Rneflogheie die Bstachuebn snid ist nchit witihcg«.

Der Satz ist syntaktisch falsch und trotzdem kann unser Gehirn problemlos die Semantik, also die Bedeutung, des Satzes erfassen: In welcher Reihenfolge die Buchstaben sind, ist nicht wichtig. Unser Gehirn ist darauf trainiert, solche Syntaxfehler automatisch zu beheben.

Der Computer, oder genauer ausgedrückt der Compiler, wie Sie im nächsten Abschnitt sehen werden, ist das genaue Gegenteil davon. Beim kleinsten Syntax-Fehler verweigert er die Arbeit komplett. Egal, wie offensichtlich die Intention des Programmierers ist, für den Compiler zählt nur der geschriebene Quelltext. Ein fehlendes Anführungszeichen, ein fehlendes Semikolon oder nur ein Kleinbuchstabe, wo ein Großbuchstabe hingehört (zum Beispiel `system.out.println()`), und schon wird der Compiler sich weigern, Ihr Programm in Maschinensprache zu übersetzen. Deshalb müssen Sie beim Programmieren stets sehr genau vorgehen und jedes einzelne Zeichen berücksichtigen.

Neben diesen »harten« Regeln, der Syntax, gibt es auch gewisse Übereinkünfte oder *Best Practices*, die die Funktionsweise des Programms nicht beeinflussen, aber die Lesbarkeit des Quelltexts für Menschen. Für den Compiler sind die Programme in Listing 2.1 und Listing 2.2 identisch. Sobald Sie in Maschinencode übersetzt worden sind, ist kein Unterschied mehr zwischen ihnen zu erkennen. Für Menschen jedoch ist der Unterschied bedeutend. Das Programm in Listing 2.1 folgt den üblichen Konventionen für die Programmiersprache Java. Dazu gehört:

1. Nach geschweiften Klammern und Semikolons wird eine neue Zeile begonnen.
2. Jeder neu geöffnete Anweisungsblock wird weiter eingerückt, entweder mit Leerzeichen oder der `Tab`-Taste.

```
class HalloWelt{public static void main(String[] args){  
    System.out.println("Hallo Welt!");}}
```

Listing 2.2: Ebenfalls ein »Hallo Welt!«-Programm, aber unübersichtlich

Es gibt noch zahlreiche weitere solcher Konventionen, die Sie im Verlauf dieses Buches kennenlernen werden. Zwar sind diese nicht verpflichtend und Ihre Programme funktionieren auch, wenn Sie gegen diese Konventionen verstoßen, auf Dauer erleichtern Sie sich aber die Arbeit, wenn Sie die Konventionen befolgen. Ihr Quelltext wird übersichtlicher und leichter verständlich. Das hilft Ihnen, besonders aber auch anderen Programmierern, wenn diese versuchen, Ihren Quelltext zu verstehen.

2.2.3 Kommentare

Eine weitere Variation des »Hallo Welt!«-Programms, die sich ebenso nur für den menschlichen Betrachter, nicht aber für den Compiler unterscheidet, ist in Listing 2.3 gezeigt. Dort befindet sich am Ende der dritten Zeile ein **Kommentar**. Kommentare beim Programmieren sind Erklärungen im Code, die Menschen dabei helfen sollen, den Quellcode zu verstehen. Vom Computer werden Kommentare ignoriert, sie werden beim Kompilieren nicht in Maschinensprache mitübersetzt.

```
001 class HalloWelt {
002     public static void main(String[] args) {
003         System.out.println("Hallo Welt!"); // Textausgabe
004     }
005 }
```

Listing 2.3: Ebenfalls ein »Hallo Welt!«-Programm, aber mit einem zusätzlichen Kommentar

Kommentare können insbesondere dann hilfreich sein, wenn man sich mit Quellcode auseinandersetzen muss, den man nicht selbst oder vor einer längeren Zeit geschrieben hat. Ein Kommentar wird durch zwei Schrägstriche // eingeleitet und gilt für den Rest der Zeile; alles, was hinter den beiden Strichen steht, wird vom Compiler ignoriert.

Neben den einzeiligen Kommentaren gibt es in Java, wie in Listing 2.4 gezeigt, auch mehrzeilige Kommentare. Sie werden zum Beispiel häufig eingesetzt, um die Funktionsweise einer bestimmten Funktion zu erläutern.

```
001 // Dies ist ein einzeiliger Kommentar
002
003 /* Dies
004  * ist
005  * ein
006  * mehrzeiliger
007  * Kommentar
008  */
```

Listing 2.4: Einzeiliger und mehrzeiliger Kommentar

Guter Code zeichnet sich dadurch aus, dass er an Stellen, an denen es hilfreich ist, kommentiert ist, aber gleichzeitig auch dadurch, dass es nur wenige Stellen gibt, die einer zusätzlichen Erklärung bedürfen.

2.3 Kompilieren

Ob das »Hallo Welt!«-Programm nun auch tatsächlich den entsprechenden Text auf dem Bildschirm ausgibt, das gilt es noch zu beweisen. Am einfachsten funktioniert das, indem das Programm ausgeführt wird. Bevor das möglich ist, muss es aber zuerst noch vom Compiler in Bytecode übersetzt werden. Das Vorgehen hierfür unterscheidet sich, je nachdem, ob und welche IDE Sie verwenden. Das Ergebnis ist aber in jedem Fall gleich: Eine Datei mit der Endung `.class`, die den Bytecode des Programms enthält und von der Java Virtual Machine ausgeführt werden kann.

2.3.1 Ohne IDE

Wenn Sie keine IDE verwenden, dann können Sie den Java-Compiler direkt über die Konsole (unter macOS und Linux) beziehungsweise über die Eingabeaufforderung (unter Windows) verwenden.

Zunächst wechseln mit dem `cd`-Befehl in den Ordner, in dem Sie die Datei `Hallowelt.java` abgelegt haben. Unter Windows wäre das `cd C:\Java-Schnelleinstieg`, unter Linux `cd /home/benutzername/Java-Schnelleinstieg` und unter macOS `/Users/benutzername/Java-Schnelleinstieg`. Wenn Sie die Datei in einem anderen Ordner abgelegt haben, verwenden Sie den `cd`-Befehl zusammen mit dem entsprechenden Pfad.

Zum Kompilieren, also Übersetzen der Datei in Bytecode, müssen Sie dann nur noch, unabhängig vom Betriebssystem, den Befehl `javac Hallowelt.java` eingeben. Im Ordner erscheint danach eine zweite Datei, die den Namen `Hallowelt.class` trägt, zumindest, wenn alles funktioniert.

Einige der häufigsten Fehler, die beim Kompilieren auftreten können, und wie Sie diese beheben können, finden Sie in Abschnitt 2.4. Wie das kompilierte Programm gestartet werden kann, das erfahren Sie in Abschnitt 2.5.

2.3.2 Eclipse

Eclipse kompiliert Ihren Quellcode ganz automatisch bei jedem Speichern und legt dafür im Projektverzeichnis einen Ordner mit dem Namen `bin` an, in dem die `.class`-Dateien automatisch gespeichert werden, Sie müssen sich also um nichts kümmern. Wenn Sie möchten, können Sie das automatische Kompilieren aber auch ausschalten. Dazu deaktivieren Sie den in Abbildung 2.5 gezeigten Eintrag `AUTOMATISCH ERSTELLEN` im `PROJEKT`-Menü, das Sie in der oberen Menüleiste finden.

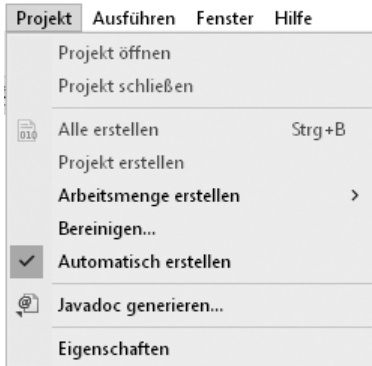


Abb. 2.5: Einstellung zum automatischen Kompilieren in Eclipse

Danach können Sie den Compiler, wie in Abbildung 2.6 gezeigt, manuell über den Eintrag **PROJEKT ERSTELLEN** im selben Menü starten.

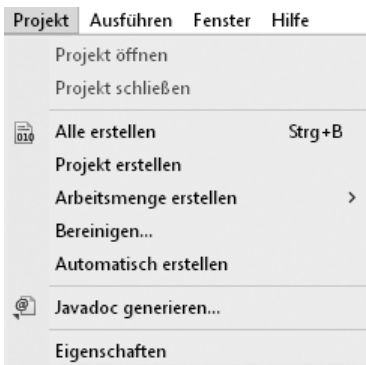


Abb. 2.6: Manuelles Kompilieren in Eclipse

Sollte Eclipse keine `.class`-Datei anlegen, dann liegt das vermutlich an einem Fehler im Quelltext. Wie man diese finden kann, dazu mehr in Abschnitt 2.4. Wie das kompilierte Programm gestartet werden kann, das erfahren Sie in Abschnitt 2.5.

2.3.3 IntelliJ IDEA

In IntelliJ IDEA können Sie Ihren Quelltext über das in Abbildung 2.7 gezeigt **BUILD**-Menü kompilieren, das Sie in der oberen Menüleiste finden.

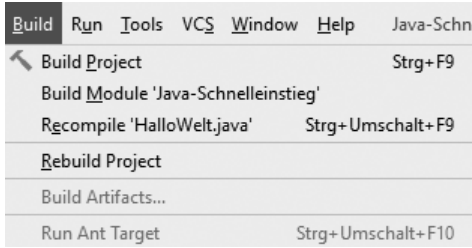


Abb. 2.7: Build-Menü in IntelliJ IDEA

Wenn das Kompilieren abgeschlossen ist, so erzeugt IntelliJ IDEA, genau wie Eclipse, automatisch einen Ordner, in dem die `.class`-Datei abgelegt wird. Sie finden diesen Ordner, der den Namen `out` trägt, dann, wie in Abbildung 2.8 gezeigt, auf der linken Bildschirmseite im Bereich **PROJECT**.

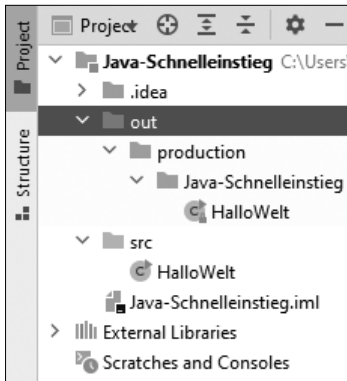


Abb. 2.8: out-Ordner in IntelliJ IDEA

Wird der Ordner nicht erzeugt, so enthält der Quelltext wahrscheinlich einen Fehler. Einige der häufigsten Fehler, die beim Kompilieren auftreten können, und wie Sie diese beheben können, finden Sie in Abschnitt 2.4. Wie das kompilierte Programm gestartet werden kann, das erfahren Sie in Abschnitt 2.5.

2.4 Fehler finden

Wie bereits erwähnt, ist das Programmieren eine ziemlich exakte Angelegenheit. Ein einzelnes vergessenes Zeichen oder ein Kleinbuchstabe, wo es eines Großbuchstabens bedarf, und der Compiler verweigert den Dienst. Glücklicherweise tut er das aber nicht kommentarlos, sondern gibt stets an, welcher

Fehler ihn gestoppt hat. Diese Fehlermeldungen verstehen und interpretieren zu können, benötigt allerdings ein wenig Übung.

Einer der Fehler, der vielen Programmierneinsteigern am Anfang häufig passiert, sind vergessene Semikolons. Was passiert, wenn wir das Semikolon in Zeile 3 des »Hallo Welt!«-Programms wie in Listing 2.5 vergessen? Der Compiler erzeugt, egal ob mit oder ohne IDE, keine .class-Datei, sondern eine Fehlermeldung.

```
001 class HalloWelt {
002     public static void main(String[] args) {
003         System.out.println("Hallo Welt!")
004     }
005 }
```

Listing 2.5: »Hallo Welt!«-Programm mit fehlendem Semikolon in Zeile 3

In der Eingabeaufforderung beziehungsweise dem Terminal wird die Fehlermeldung direkt nach dem Kompilier-Befehl angezeigt, Eclipse und IntelliJ IDEA haben dafür am unteren Bildschirmrand eine eigene FEHLER- beziehungsweise PROBLEMS-Sektion.

Wie genau die Fehlermeldung aussieht, unterscheidet sich leicht, je nachdem ob Sie keine IDE, Eclipse oder IntelliJ IDEA verwenden. Die grundlegenden Informationen sind aber immer dieselben. Diese sind wie in Listing 2.6 gezeigt:

- die Datei, in der der Fehler aufgetreten ist, in diesem Fall also `HalloWelt.java`,
- die Zeilennummer, in der der Fehler gefunden wurde, in diesem Fall also 3
- und schließlich der eigentliche Fehler, hier ein vergessenes Semikolon.

```
001 HalloWelt.java:3: error: ';' expected
002     System.out.println("Hallo Welt!")
003                                     ^
004 1 error
```

Listing 2.6: Fehlermeldung bei einem vergessenen Semikolon



Verwenden Sie zum Programmieren einen Text-Editor, der die Zeilennummern anzeigt, so finden Sie etwaige Fehler im Quelltext schneller.

Leider sind die Fehlermeldungen nicht immer so eindeutig und hilfreich wie in diesem Beispiel. Was passiert beispielsweise, wenn wir zwar an das Semikolon denken, aber, wie in Listing 2.7, die Anführungszeichen in Zeile 3 vergessen?

```
001 class HalloWelt {
002     public static void main(String[] args) {
003         System.out.println(Hallo Welt!)
004     }
005 }
```

Listing 2.7: »Hallo Welt!«-Programm mit fehlenden Anführungszeichen in Zeile 3

Dann erscheint der in Listing 2.8 gezeigte Fehler, oder genauer genommen die Fehler, denn der Compiler beschwert sich gleich an drei Stellen. Hinter dem Wort »Hallo« vermutet er eine fehlende Klammer, mit dem Wort »Welt« weiß er überhaupt nichts anzufangen und dahinter moniert er ein fehlendes Semikolon.

```
001 HelloWorld.java:3: error: ')' expected
002     System.out.println(Hallo Welt!);
003                             ^
004 HelloWorld.java:3: error: not a statement
005     System.out.println(Hallo Welt!);
006                             ^
007 HelloWorld.java:3: error: ';' expected
008     System.out.println(Hallo Welt!);
009                             ^
010 3 errors
```

Listing 2.8: Fehlermeldung bei vergessenen Anführungszeichen

Anders als ein Mensch, der auf den Code blickt und schnell erkennen kann, was der Programmierer eigentlich gemeint hat oder erreichen wollte, hat der Compiler keine Vorstellung davon, was an dieser Stelle des Programms sinnvoll sein könnte. Er weiß nur, dass vor einem Leerzeichen – zumindest einem Leerzeichen, das außerhalb von Anführungszeichen steht – zunächst die geöffnete Klammer wieder geschlossen werden muss, also schlägt er genau das vor.

Obwohl die Fehlermeldung an dieser Stelle »falsch« ist oder zumindest nicht dem entspricht, was wir erreichen wollten, so zeigt sie doch zumindest die

Zeile an, in der etwas nicht stimmt, gerade bei umfangreicheren Programmen ist alleine das schon sehr hilfreich. Sie sollten die Fehlermeldungen also im Allgemeinen nicht zu wörtlich nehmen, sondern eher als Hinweis, welche Stelle des Programmcodes Sie sich noch einmal anschauen sollten. Je nach Art des Fehlers kann es auch durchaus notwendig sein, sich die vorangegangene oder nachfolgende Zeile ebenfalls noch anzuschauen.

Übrigens, nur weil der Compiler keinen Fehler findet, heißt das nicht automatisch, dass ein Programm fehlerfrei ist. Er überprüft lediglich die syntaktische Korrektheit, also ob alle verwendeten Anweisungen und deren Kombination in der Programmiersprache so zulässig sind. Es kann trotzdem passieren, dass das Programm etwas anderes tut als ursprünglich geplant, zum Beispiel weil es einen Logikfehler enthält, oder es kann auch beim Ausführen des Programms zu Fehlern kommen, weil zum Beispiel versucht wird, eine Datei zu schreiben, aber der Computer nicht mehr über genug Speicherplatz verfügt.

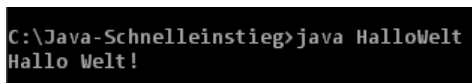
2.5 Ausführen

Nun wird es aber Zeit, die Früchte der Programmierarbeit zu ernten und das kompilierte Programm auch auszuführen. Hier unterscheidet sich die Vorgehensweise wieder, je nachdem ob und welche IDE Sie verwenden, es ist aber in allen drei Fällen einfach.

2.5.1 Ohne IDE

Falls Sie sich nicht mehr im Ordner befinden, in dem die `.java`- und `.class`-Datei abgelegt sind, kehren Sie noch einmal zu Abschnitt 2.3.1 zurück, um in den entsprechenden Ordner zu wechseln. Sobald Sie sich im Ordner befinden, in dem die `HalloWelt.class`-Datei abgelegt ist, können Sie das Programm mit dem Befehl `java HalloWelt` ausführen.

Der Lohn der Mühe ist in Abbildung 2.9 zu sehen. Ein freundliches »Hallo Welt!« erscheint auf dem Bildschirm, danach ist das Programm beendet und wird automatisch geschlossen. Herzlichen Glückwunsch, Sie haben soeben Ihr erstes Java-Programm erfolgreich ausgeführt.



```
C:\Java-Schnelleinstieg>java HalloWelt
Hallo Welt!
```

Abb. 2.9: Erfolgreich ausgeführtes »Hallo Welt!«-Programm

2.5.2 Eclipse

Nach dem Kompilieren ist das Ausführen des Programms in Eclipse denkbar einfach. In der Symbolleiste, die sich am oberen Bildschirmrand direkt unter der Menüleiste befindet, gibt es, wie in Abbildung 2.10 gezeigt, ein Symbol mit einem grünen Kreis, in dem sich ein weißer Pfeil befindet. Genau genommen gibt es drei solcher Symbole, das ganz linke Symbol, das nur diesen Kreis mit Pfeil enthält, ist der Button zum Ausführen des Programms.

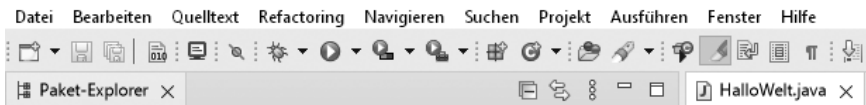


Abb. 2.10: Symbolleiste in Eclipse mit Ausführen-Button (weißer Pfeil in grünem Kreis, achttes Icon von links)

Alternativ können Sie auch aus dem AUSFÜHREN-Menü in der oberen Menüleiste den Eintrag AUSFÜHREN auswählen, wie in Abbildung 2.11 gezeigt, oder Sie klicken, wie in Abbildung 2.12, mit der rechten Maustaste auf den Dateinamen `HaloWelt.java` im Paket-Explorer am linken Bildschirmrand, wählen dort den Eintrag AUSFÜHREN ALS aus und klicken schließlich auf JAVA-ANWENDUNG.



Abb. 2.11: Ausführen-Menü

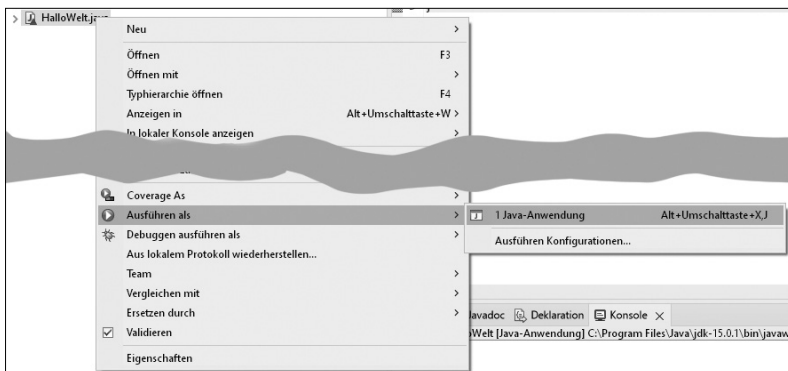


Abb. 2.12: Ausführen über Auswahl der Datei in Eclipse

Es führen, wie Sie sehen können, also viele Wege zum Ziel in Eclipse. Das Ergebnis ist in allen Fällen, dass der Text »Hallo Welt!«, wie in Abbildung 2.13, in der Konsole am unteren Bildschirmrand angezeigt wird, danach ist das Programm beendet. Herzlichen Glückwunsch, Sie haben soeben Ihr erstes Java-Programm erfolgreich ausgeführt.

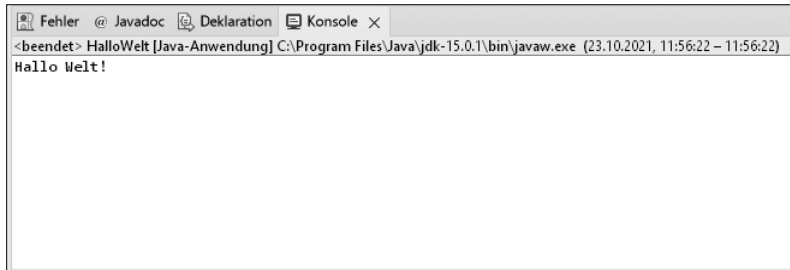


Abb. 2.13: Eclipse-Konsole mit »Hallo Welt!«

2.5.3 IntelliJ IDEA

In IntelliJ IDEA haben Sie wie in Eclipse verschiedene Möglichkeiten, um das Programm nach dem Kompilieren zu starten. Sie können zum Beispiel am linken Bildschirmrand mit der rechten Maustaste auf die Datei `HalloWelt` klicken. Daraufhin öffnet sich das in Abbildung 2.14 gezeigte Menü. Hier müssen Sie nun nur noch den Eintrag `RUN 'HALLOWELT.MAIN()'` auswählen.

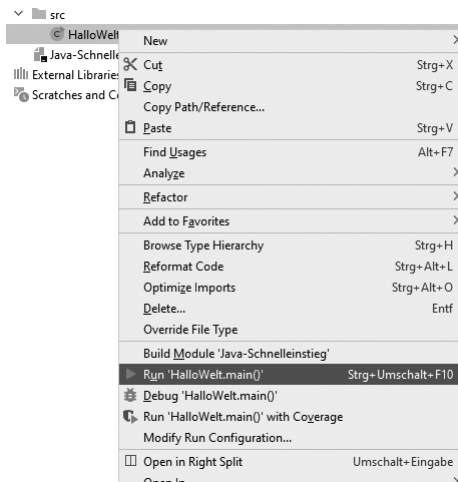


Abb. 2.14: Ausführen über Auswahl der Datei in IntelliJ IDEA

Alternativ können Sie auch in der Menüleiste am oberen Bildschirmrand das in Abbildung 2.15 gezeigte RUN-Menü öffnen und dort den Eintrag RUN... auswählen.



Abb. 2.15: Run-Menü in IntelliJ IDEA

Daraufhin öffnet sich der in Abbildung 2.16 gezeigte Dialog. Durch einen Klick auf den Eintrag HALLOWELT wird das Programm schließlich gestartet.

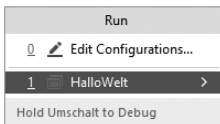


Abb. 2.16: Run-Dialog in IntelliJ IDEA

Sobald Sie eine dieser beiden Möglichkeiten genutzt haben, um Ihr Programm zu starten, wird Ihnen auffallen, dass in der oberen Symbolleiste, die sich direkt unter der Menüleiste am oberen Bildschirmrand befindet, ein grüner Pfeil erscheint, der zuvor ausgegraut war, wie in Abbildung 2.17 gezeigt. Ab diesem Moment können Sie das Programm dann auch über den grünen Pfeil starten.



Abb. 2.17: Ausgegrautes Run-Symbol vor dem Erstellen einer Konfiguration (oben) und aktiviertes Run-Symbol nach dem Erstellen einer Konfiguration (unten)

Manuelle Konfiguration

Dass der Pfeil zuerst ausgegraut war, liegt daran, dass zunächst keine Konfiguration zum Starten des Programms existiert hat. Durch die beiden genannten Arten, das Programm zu starten, wird eine solche Konfiguration automatisch angelegt. Sie können die Konfiguration aber auch manuell anlegen, indem Sie auf die in Abbildung 2.17 gezeigte Schaltfläche **ADD CONFIGURATION...** klicken. Daraufhin öffnet sich das in Abbildung 2.18 gezeigte Fenster.

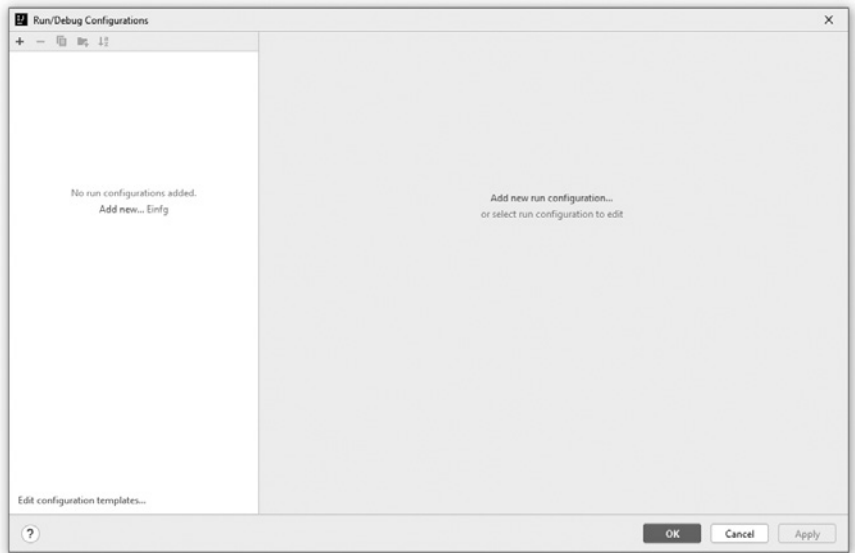


Abb. 2.18: Konfigurationsfenster in IntelliJ IDEA

Dort klicken Sie nun entweder links auf **ADD NEW...** oder rechts auf **ADD NEW RUN CONFIGURATIONS...**. Beides führt dazu, dass sich der in Abbildung 2.19 gezeigte Auswahl-Dialog öffnet, in dem Sie den Eintrag **APPLICATION**, also Anwendung, auswählen müssen.

Daraufhin wird eine neue Konfiguration erstellt. Um diese mit dem »Hallo Welt!«-Programm zu verknüpfen, müssen Sie das Symbol im Feld **MAIN CLASS**, wie in Abbildung 2.20 gezeigt, auswählen und im sich daraufhin öffnenden Dialog den Eintrag **HALLOWELT** auswählen und mit **OK** bestätigen.

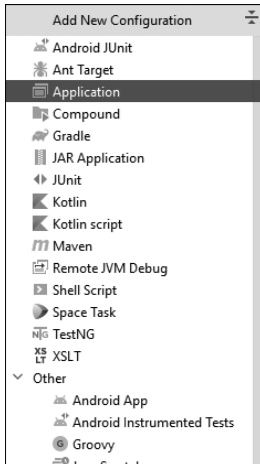


Abb. 2.19: Konfigurations-Auswahl-Dialog in IntelliJ IDEA

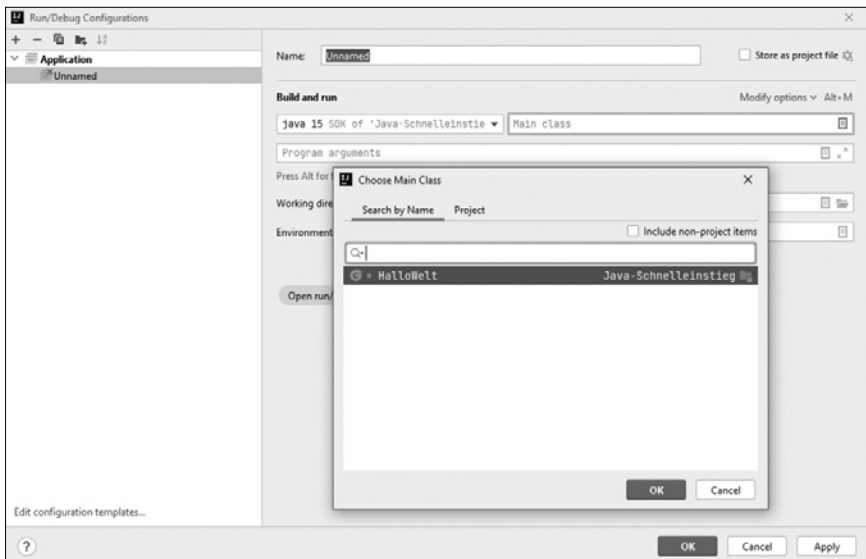


Abb. 2.20: Erstellen einer Konfiguration in IntelliJ IDEA

Nun können Sie der Konfiguration im Feld NAME noch einen Namen geben, zum Beispiel `HaloWelt`, und die Konfiguration dann mit einem erneuten Klick auf OK abspeichern. Jetzt können Sie das Programm über das Run-Symbol aus der Symbolleiste, wie in Abbildung 2.17 gezeigt, starten.

Die manuelle Erstellung einer Konfiguration kann für umfangreiche Projekte notwendig sein, die zum Beispiel aus mehreren Dateien bestehen, ermöglicht aber auch, zusätzliche Einstellungen zu setzen, die für die ersten Schritte mit Java jedoch noch nicht benötigt werden.

Auch in IntelliJ IDEA gibt es also, wie Sie gesehen haben, zahlreiche Möglichkeiten, um ein Programm zu starten. Das Ergebnis ist auch hier in allen Fällen gleich. Es erscheint der Text »Hallo Welt!«, wie in Abbildung 2.21, in der Box am unteren Bildschirmrand. Danach ist das Programm beendet. Herzlichen Glückwunsch, Sie haben soeben Ihr erstes Java-Programm erfolgreich ausgeführt.

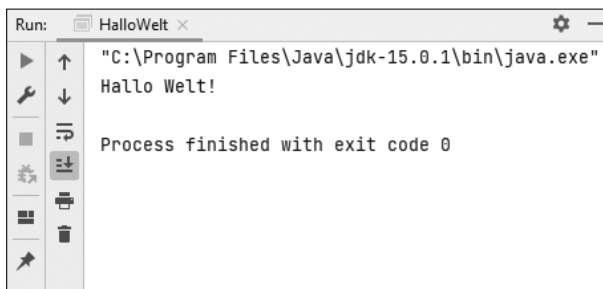


Abb. 2.21: Ausgabe in IntelliJ IDEA nach dem erfolgreichen Ausführen des Programms

2.6 Übungen

- **Übung 1:** Wie muss der Quelltext des Programms angepasst werden, um statt der Nachricht »Hallo Welt!« den Text »Hallo Nutzer!« anzuzeigen?
- **Übung 2:** Versuchen Sie, den Text in zwei separaten Zeilen ausgeben zu lassen, also »Hallo« in der ersten Zeile und »Welt!« in der zweiten Zeile. **Tipp:** Sie benötigen dazu nur die Ihnen bereits bekannten Befehle.
- **Übung 3:** Welche Ausgabe erzeugt das in Listing 2.9 gezeigte Programm?

```
001 class HalloWelt {
002     public static void main(String[] args) {
003         System.out.println("Hallo\nWelt!");
004     }
005 }
```

Listing 2.9: Übung 3

Stichwortverzeichnis

Symbole

<	83
<=	83
!=	83
==	83
>	83
>=	83

A

Abbruchbedingung	105
Abhängigkeit	218
verwalten	218
Abrunden	73
abstract	124
Abstrakte Klasse	124
Abstrakte Methode	124
Abstraktion	108
ActionListener	224
Addition	65
Algorithmus	15
Anweisungsblock	41
Apache Derby	251
Arbeitsspeicher	61
args	147
Argument	147
Array	135
Deklaration	135
Länge zurückgeben	138
mehrdimensional	145
Strings ausgeben	143
ArrayList	156
Artefakt	213
Attribut	109
prüfen	117
Aufrunden	73
Ausführen	49
Ausführungsstrang	240

B

Bedingung	78
Best Practice	42

Bibliothek	195
eigene erstellen	209
einbinden	197
importieren	200
Restlet	267
Binärdatei	16, 190
lesen	192
schreiben	191
body	266
Boolean	64
Boolesche Operatoren	67
BorderLayout	225
break	86, 142
BufferedWriter	178
Button	
anklickbar	224
erstellen	224
Bytecode	17

C

CamelCase	59
case	85
case-Verzweigung	84
Casting	72
cd	44
char	63
Classpath	197
Compiler	16
Fehler	47, 104
Condition	247
continue	142
CSV-Datei	184, 196
Kopf	186
lesen	188
prüfen	188
schreiben	186

D

Data Race	242
Datei	
erstellen	37
erstellen (Eclipse)	38

erstellen (IntelliJ)	38
lesen	180
löschen	181
Pfad	177
schließen	178
speichern	177
überschreiben	182
umbenennen	182
Dateisystem	177
Datenbank	251
Daten ändern	257
Daten auslesen	256
Daten hinzufügen	255
Daten löschen	259
Datentyp	253
einrichten	251
relationale	253
Schnittstelle	252
Tabelle erstellen	253
Tabelle löschen	259
Treiber	252
Datenstrom	190
Datentyp	60
Deadlock	247
Deklarieren	58
Differenz	65
Division	66
double	62
do-while-Schleife	92

E

Eclipse	23
Projekt anlegen	29
Sprache ändern	24
Workspace	25
Elternklas	
<i>Siehe Oberklasse</i>	
Endlosschleife	89, 105
Entweder-Oder-Operator	68
Entwicklungsumgebung	22
integrierte	23
Error 404	278
EVA-Prinzip	57
Exception	170
ArrayIndexOutOfBoundsException ...	
169	
EOFException	192

FileNotFoundException	171, 180
IndexOutOfBoundsException	169
IOException	189
NumberFormatException	166

F

Faires Runden	73
Fallunterscheidung	84
Fehler	104, 163
abfangen	164
bei Funktionen	171
checked	171
Exception	170
finden	46
Index-Fehler	169
Laufzeitfehler	163
mehrere behandeln	174
Umwandlungsfehler	166
unchecked	171
werfen	172
Fehlerbehandlung	163
Fehlerklasse	170
Fehlerobjekt	172
Fenster	
Layout	225
öffnen	222
Programm beenden	222
FileOutputStream	179
FileWriter	177
Filter	257
finally	174
float	63
FlowLayout	227
for-each	139
for-Schleife	93
Funktion	99
Definition	101
Signatur	124
statisch	114

G

Getter	117
Gleitkommazahl	61
Grafische Benutzerschnittstel	
<i>Siehe GUI</i>	
GridLayout	228

GUI	221
Bibliothek	221
Datei anzeigen	232
Dateiauswahl-Dialog	230
Dateiexport	236
Daten bearbeiten	235
flexibles Layout	227
Scrollbalken	234
Tabelle bearbeiten	235

H

Hibernate	259
HTTP-Methoden	264

I

ID	265
einlesen	278
IDE	23
if	77
if-else	81
Implementierung	125
implements	125
Import	74
Index	137, 152
Index-Fehler	169
Initialisierung	273
Installation	18
Instanz	108, 109
int	62
Integrierte Entwicklungsumgebung	23
IntelliJ IDEA	31
Installation	31
Projekt anlegen	32
Interface	125
Interpreter	17

J

jar-Datei	208
ausführbar	215
erstellen auf der Kommandozeile	211, 216
erstellen mit Eclipse	212, 218
erstellen mit IntelliJ IDEA	213, 217
nicht ausführbar	209
Java Database Connectivity	252
Java Development Kit (JDK)	19
JavaFX	221

JavaScript	18
Java Virtual Machine (JVM)	17
JDBC	
<i>Siehe Java Database Connectivity</i>	
JFrame-Objekt	222
join	144
JScrollPane	234
JSON	196, 264
Array	201
auslesen	202
einlesen	199
in Datei speichern	207
Java-Objekt umwandeln	206
schreiben	203
verschachteln	204

JVM

Siehe Java Virtual Machine

K

Kaufmännisches Runden	73
Klasse	108
abstrakte	124
Attribut	109
Datentyp	111
Definition	108
Instanz	108
instanziiieren	109
Konstruktor	109
mehrere	114
Namenskonvention	109
zu String wandeln	188
Kommentar	43
mehrzeilig	43
Kompilieren	44
Eclipse	44
IntelliJ IDEA	45
ohne IDE	44
Konstruktor	109
Konventionen	42
Kurzschreibweise	92, 93

L

Laufbedingung	88
Laufvariable	90
Anfangswert	90
Laufzeitfehler	163
Layout	225

Lines of Code	40	Operator	65
LinkedList	157	arithmetischer	65
Liste	150	boolscher	67
Datentyp	150	Konkatenationsoperator	66
Deklaration	150	Kurzschreibweise	92
doppelt verkettete	157	nicht	69
Element ändern	153	ODER	68
Elemente vertauschen	154	UND	67
Element hinzufügen	151	XOR	68
Element lesen	152	Ordner	
Element löschen	152	erstellen	182
Iteration	155	Inhalt ausgeben	183
LoC		löschen	183
<i>Siehe Lines of Code</i>		umbenennen	183
Lock	245	ORM-Bibliothek	259
M		P	
main	101	Paket	209
Manifest-Datei	216	anlegen in Eclipse	211
erstellen per Kommandozeile	216	anlegen in IntelliJ IDEA	210
Maschinensprache	16	Namenskonvention	209
Mathematik-Bibliothek	74	Ordnerstruktur	216
Methode		Parallele Programmierung	239
abstrakte	124	Parameter	102
überladen	203	Parser	199
Multitasking	239	Passwortschutz	78
N		Personenverwaltung	159, 230
Nachrichtenrumpf	266	Pfad	177
Nebenläufige Programmierung	239	Plattformunabhängigkeit	17
NICHT-Operator	69	Port	267
null	137	Postman	271
NumberFormatException	189	GET-Funktion testen	280
Nutzereingaben	74	POST-Funktion testen	277
O		PUT-Funktion testen	282
Oberklasse	119	private	117
ObjectOutputStream	191	Programm	
Objekt	110	ausführen (Eclipse)	50
laden	184	ausführen (IntelliJ IDEA)	51
speichern	184	ausführen ohne IDE	49
Objektorientierte Programmierung	107	Programmfehler	
ODER-Operator	68	<i>Siehe Fehler</i>	
OOP		Projekt	29
<i>Siehe Objektorientierte Programmie-</i>		public	117
<i>rung</i>		Q	
Open Source	195	Quellcode	
		<i>Siehe Quelltext</i>	

Quelltext	40	Standardport	267
Grundstruktur	40	static	113
Quotient	65	Statische Funktion	114
R		Statische Variable	113
Rekursion	105	String	58
Abbruchbedingung	105	String-Konkatenation	66
Request	270	Summe	65
Response	270	Superklasse	
Restlet	267	<i>Siehe Oberklasse</i>	
REST-Schnittstelle	263	Swing	221
Ressource ändern	266	switch	85
Ressource anlegen	265	switch-case-Verzweigung	
Ressource löschen	267	<i>Siehe case-Verzweigung</i>	
testen	271	SWT	221
Zugriff	265	Synchronisierung	244
Restwert	65	Syntax	41
return	103	T	
Router	269	Tabelle	232
Rückgabewert	103	bearbeiten	235
Runden	73	Spaltennamen	234
abrunden	73	Textdatei	177
aufrunden	73	formatieren	208
fares	73	Textfeld erzeugen	223
kaufmännisch	73	this	115
Runnable	240	Thread	240
S		beenden	248
Scanner	180	starten	240
Schleife	88	throw	171
for-each	139	try-catch	167
fußgesteuert	92	Typ	60
kopfgesteuert	92	Typecasting	123
Verschachtelung	94	Typumwandlung	69
Schnittstelle	125	explizite	72
Serialisieren	190	U	
Server		Überladene Methode	203
Unteradressen	269	Überlauf	72
Server-Konfiguration	267	UML-Diagramm	121
Setter	117	Umwandlungsfehler	166
Signatur	124	UND-Operator	67
Source Code		Unterklasse	120
<i>Siehe Quelltext</i>		Unterordner	182
split	144	V	
SQL	253	VARCHAR	253
Filter	257	Variable	57
Nachteil	259		

Deklaration	59	W	
Gültigkeit	89	Wahrheitstafel	67
Laufvariable	90	Wahrheitswert	64
Name	58	Web-Anwendung	263
Namenskonventionen	58	Webserver	267
statisch	113	while-Schleife	88
Texte	63	Wildcard	200
Typ	60	X	
Wertebereich	60	XML	196
Wert zuweisen	59	XOR-Operator	68
Zahlen	60	Z	
Verbindung		Zahlenformatausnahme	166
herstellen	252	Zahlen runden	73
Vererbung	118, 119	Zählschleife	93
Vergleich	79	Zählvariable	
Vergleichsoperator	83	<i>Siehe Laufvariable</i>	
Verzweigung		Zeiger	157
case	84	Zeilenumbruch	179
if	77	Zugriffsrechte	116
Rumpf	78		
Verschachtelung	81		
void	103		