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-Programmcode 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.

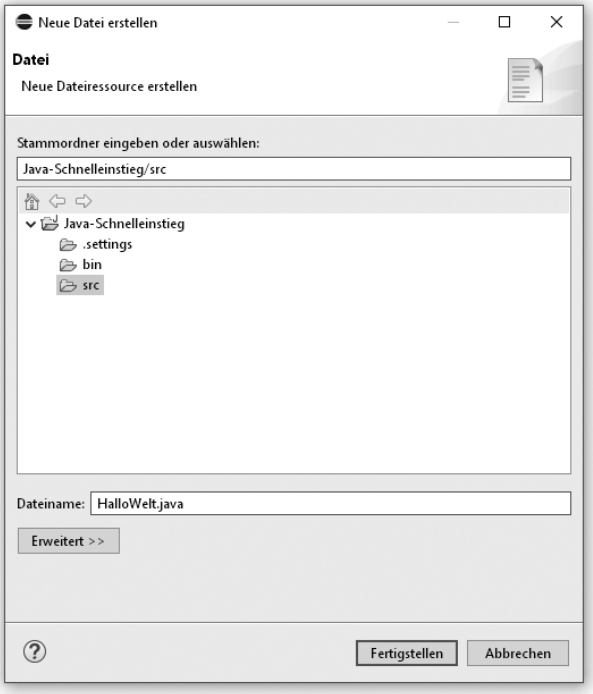

Abb. 2.1: Erstellen einer neuen Datei in Eclipse

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.

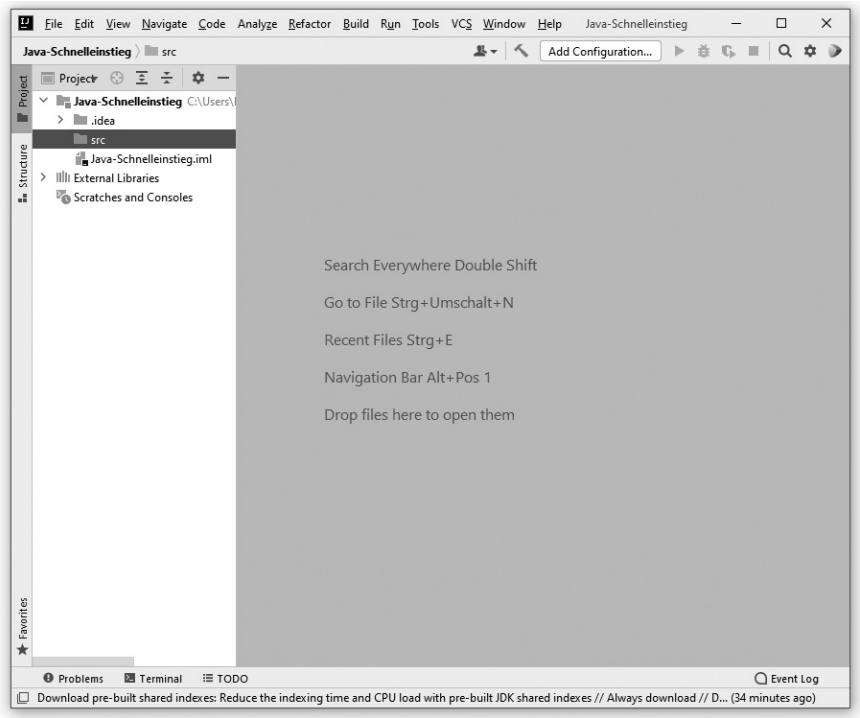

Abb. 2.2: Leeres Projekt mit ausgewähltem `src`-Ordner

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.

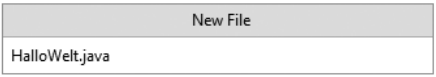

Abb. 2.3: Erstellen einer neuen Datei in IntelliJ IDEA

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 *Anweisungblöcke*, 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.

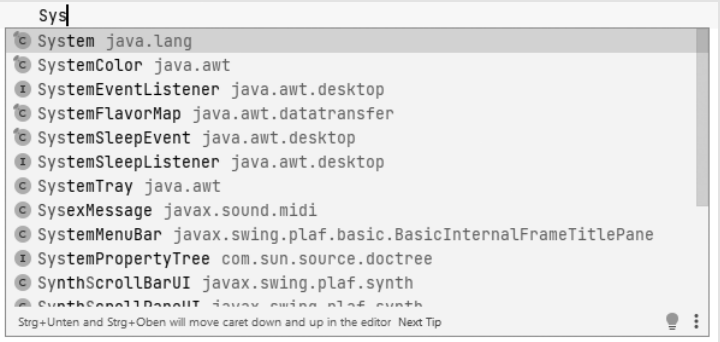

Abb. 2.4: Autovervollständigung in IntelliJ IDEA

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-Schnelleinsteig` 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.

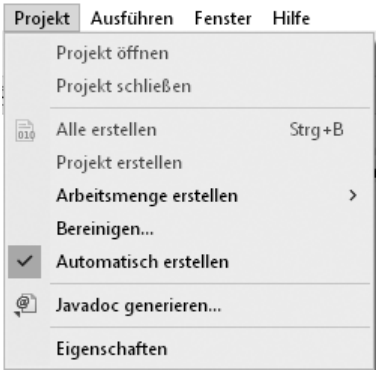

Abb. 2.5: Einstellung zum automatischen Kompilieren in Eclipse

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

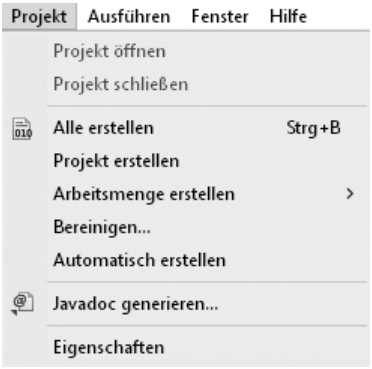

Abb. 2.6: Manuelles Kompilieren in Eclipse

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.

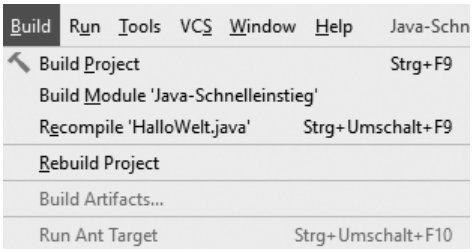

Abb. 2.7: Build-Menü in IntelliJ IDEA

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.

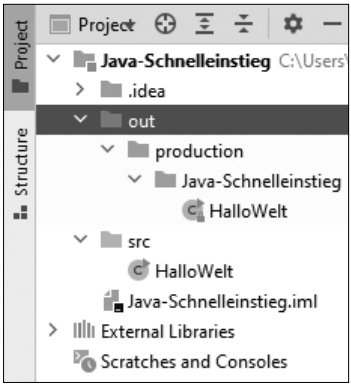

Abb. 2.8: out-Ordner in IntelliJ IDEA

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 Programmiereinsteigern 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 HalloWelt.java:3: error: ')' expected
002         System.out.println(Hallo Welt!);
003                                ^
004 HalloWelt.java:3: error: not a statement
005         System.out.println(Hallo Welt!);
006                                 ^
007 HalloWelt.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, achtes 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 `HalloWelt.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

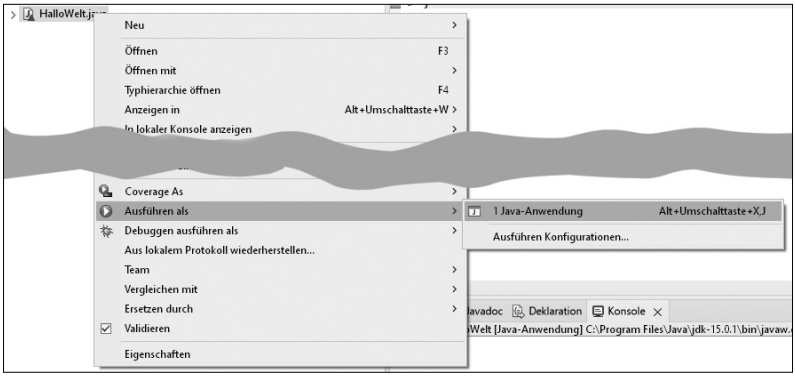

Abb. 2.12: Ausführen über Auswahl der Datei in Eclipse

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.

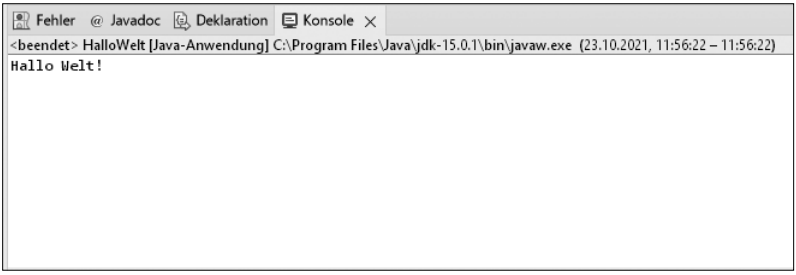

Abb. 2.13: Eclipse-Konsole mit »Hallo Welt!«

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.

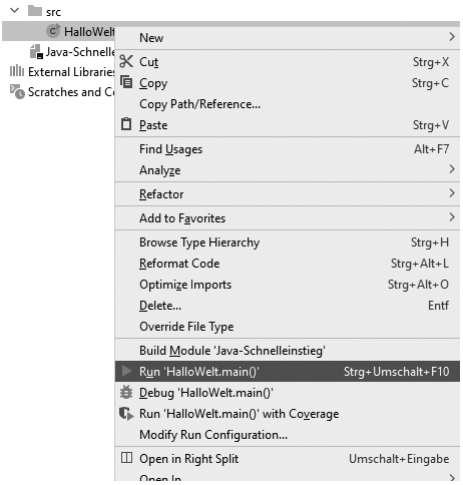

Abb. 2.14: Ausführen über Auswahl der Datei in IntelliJ IDEA

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.

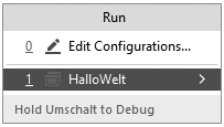

Abb. 2.16: Run-Dialog in IntelliJ IDEA

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.

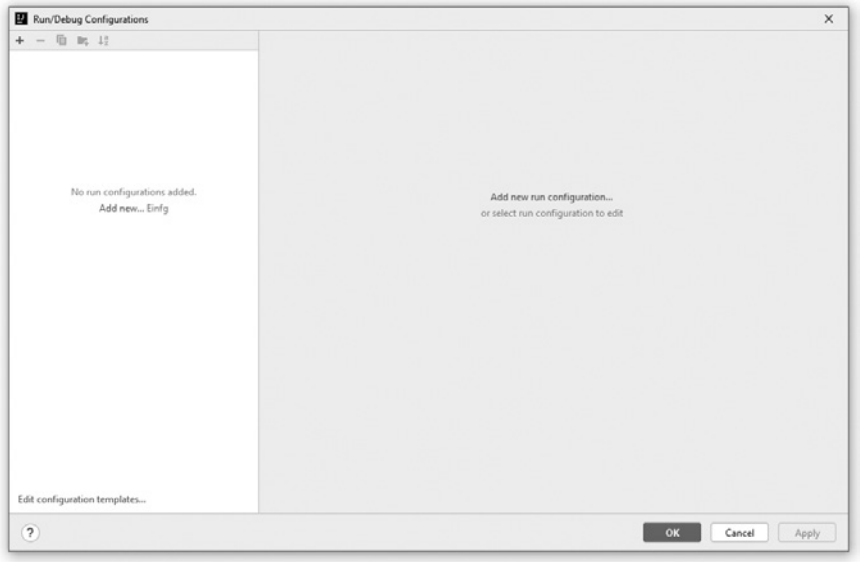

Abb. 2.18: Konfigurationsfenster in IntelliJ IDEA

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.

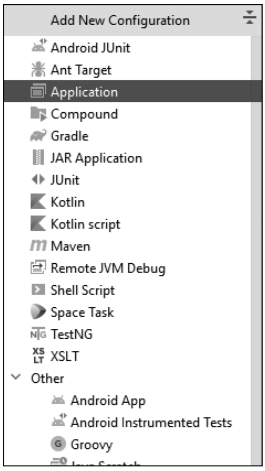

Abb. 2.19: Konfigurations-Auswahl-Dialog in IntelliJ IDEA

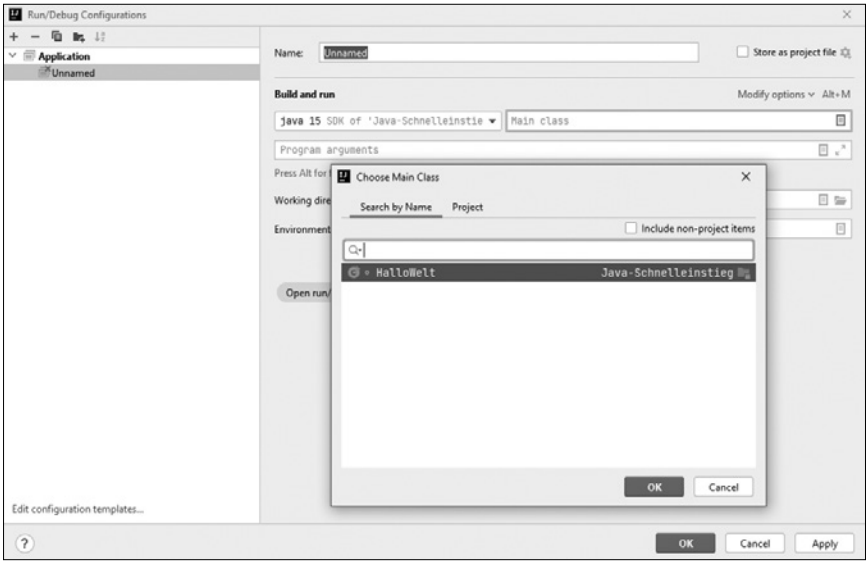

Abb. 2.20: Erstellen einer Konfiguration in IntelliJ IDEA

Nun können Sie der Konfiguration im Feld NAME noch einen Namen geben, zum Beispiel `HalloWelt`, 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.

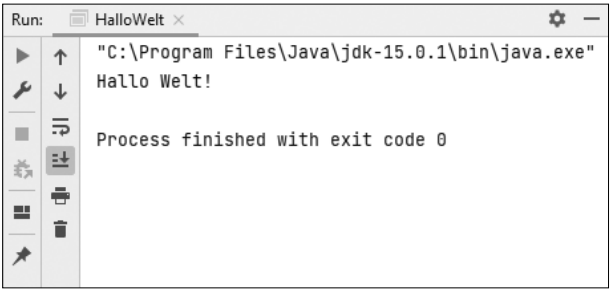

Abb. 2.21: Ausgabe in IntelliJ IDEA nach dem erfolgreichen Ausführen des Programms

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