Let's Play: Programmieren lernen mit Java und Minecraft

Kapitel 1

Java

Ob bewusst oder unbewusst, eine der wichtigsten Entscheidungen, die man auf dem Weg zum Programmierer zu treffen hat, hast du bereits getroffen: welche Programmiersprache du lernen möchtest. Mit diesem Buch hast du dich nämlich für die Programmiersprache Java entschieden. Bevor wir uns aber mit den Besonderheiten von Java beschäftigen und damit, warum es eine gute Entscheidung ist, Java zu lernen, soll es zunächst um die Frage gehen, was Programmiersprachen eigentlich sind und warum sie benötigt werden.

1.1 Programmiersprachen

Beim Programmieren geht es im Wesentlichen darum, dass der Programmierer dem Computer eine bestimmte Aufgabe gibt, die dieser erledigen soll. Damit er das kann, braucht der Computer eine genaue Handlungsvorschrift, die auch *Algorithmus* genannt wird. Auch im Alltag begegnen uns oft Handlungsvorschriften, zum Beispiel in Form eines Rezepts:

1. 250 Gramm Mehl in eine Schüssel geben
2. 500 Milliliter Milch dazugeben
3. 2 Eier hinzugeben
4. Mit einer Prise Salz würzen
5. Umrühren

Fertig ist der Crêpes-Teig! Damit eine Handlungsvorschrift korrekt ausgeführt werden kann, müssen sich beide Seiten auf eine gemeinsame Sprache einigen. Wenn dir jemand ein Rezept auf Chinesisch gibt, kannst du vermutlich nicht viel damit anfangen.

Computer »sprechen« in Einsen und Nullen, also in einer Sprache, mit der Menschen nicht besonders gut umgehen können. Unsere Sprache wiederum, egal ob es sich um Deutsch, Englisch oder Chinesisch handelt, ist für den Computer viel zu ungenau. Nehmen wir zum Beispiel den Satz: »Da vorne ist eine Bank.« Obwohl es sich dabei um einen vollkommen korrekten deutschen Satz handelt, ist doch nicht eindeutig klar, was mit dem Satz eigentlich gemeint ist. Steht da vorne eine Parkbank, auf die man sich setzen kann, oder ist dort die Filiale einer Bank, auf der man Geld einzahlen und abheben kann?

Es wäre ein recht kostspieliger Fehler, wenn dein Computer beim Online-Shoppen aus Versehen die Deutsche Bank statt einer Bank für den Garten kauft.

Algorithmen müssen deshalb nicht nur Handlungsvorschriften sein, sie müssen *eindeutige Handlungsvorschriften* sein. Auch mit Begriffen wie »eine Prise« kann ein Computer wenig anfangen. Aus diesem Grund nutzen wir Programmiersprachen, denn sie ermöglichen es uns, eindeutige Handlungsvorschriften festzulegen. Und obwohl sie auf den ersten Blick recht kompliziert scheinen, können wir sie doch leichter lernen als eine Sprache aus Nullen und Einsen.

Damit der Computer die Programmiersprache auch versteht, muss sie aber zunächst übersetzt werden, in die sogenannte *Maschinensprache*. Diese Übersetzung findet durch ein Programm statt, das *Compiler* genannt wird. Das Ergebnis sind dann sogenannte *Binärdateien*, die vom Computer ausgeführt werden können. Diese Binärdateien bestehen, wie in Abbildung 1.1 gezeigt, nur aus Nullen und Einsen.

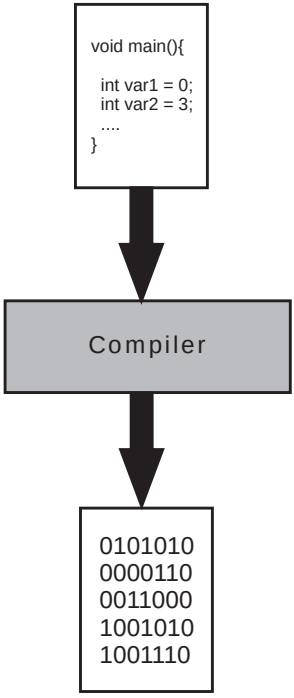

Abbildung 1.1: Funktionsweise eines Compilers

Der einfache Satz »Das ist ein Test.« wird so zum Beispiel zu einer 136 Zeichen langen Kette aus Nullen und Einsen, die du in Listing 1.1 sehen kannst.

```
01000100 01100001 01110011 00100000 01101001 01110011 01110100 00100000
01100101 01101001 01101110 00100000 01010100 01100101 01110011 01110100
00101110
```

Listing 1.1: Binärcodierung von »Das ist ein Test.«

Merke

- Ein *Algorithmus* ist eine eindeutige Handlungsvorschrift.
- Der *Compiler* übersetzt Programmiersprache in Maschinensprache.
- Eine *Binärdatei* besteht aus Nullen und Einsen.

1.2 Besonderheiten von Java

Verschiedene Programmiersprachen haben verschiedene Vor- und Nachteile. Einige sind besonders leicht zu erlernen, wie zum Beispiel Python, andere, wie zum Beispiel C, sind besonders für zeitkritische Anwendungen geeignet, also Anwendungen, bei denen es auf schnelle Reaktionszeiten ankommt, und wieder andere sind besonders universell einsetzbar, wie zum Beispiel Java. Die eine »richtige« oder »beste« Programmiersprache gibt es daher nicht – je nach Anwendungsfall kann der Einsatz einer anderen Programmiersprache sinnvoll sein.

Der Hauptgrund, warum wir Java zum Programmieren unserer Plugins verwenden, ist, dass sowohl Minecraft selbst als auch der Minecraft-Server in Java programmiert sind. Außerdem können Java-Programme, im Gegensatz zu vielen in anderen Programmiersprachen geschriebenen Programmen, problemlos auf allen gängigen Betriebssystemen ausgeführt werden, also insbesondere auf Windows, GNU/Linux und macOS.

Damit das möglich ist, funktioniert der Java-Compiler anders als andere Compiler. Er wandelt die Programmiersprache nicht sofort in Maschinencode um, sondern zunächst in den sogenannten *Java-Bytecode*. Dieser ist ein Zwischenschritt zwischen der für Menschen gut lesbaren Programmiersprache und dem für den Computer gut lesbaren Maschinencode. Erst die sogenannte *Java Virtual Machine* (*JVM*) wandelt das Programm in Maschinencode um.

Der Vorteil: Statt jedes Programm in Maschinencode für jedes Betriebssystem, also zum Beispiel Windows, macOS und GNU/Linux übersetzen zu müssen, muss nur ein Programm, nämlich die Java Virtual Machine, für jedes Betriebssystem übersetzt werden – und das bedeutet deutlich weniger Aufwand.

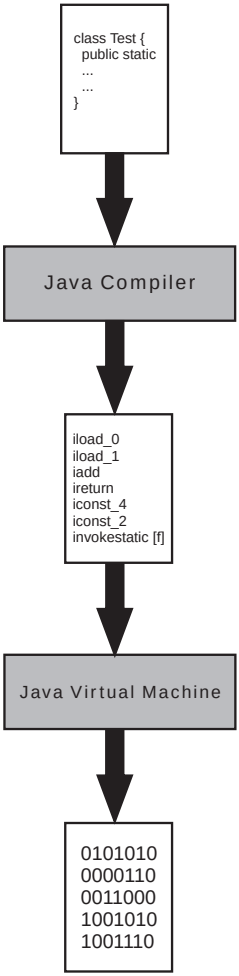

Abbildung 1.2: Funktionsweise des Java-Compilers

1.3 Installation und Einrichtung

Bevor du mit dem eigentlichen Programmieren loslegen kannst, musst du daher dafür sorgen, dass auf deinem Computer sowohl die Java Virtual Machine als auch der Java-Compiler installiert sind. Auf manchen Systemen, insbesondere GNU/Linux-Systemen,

sind beide Programme schon vorinstalliert. Um zu testen, ob das bei deinem System der Fall ist, musst du zunächst die Eingabeaufforderung (Windows) beziehungsweise das Terminal (GNU/Linux, macOS) öffnen, denn im Gegensatz zu den meisten modernen Programmen, wie wir sie heute kennen, hat der Java Compiler keine grafische Oberfläche, sondern wird komplett über die Eingabeaufforderung bedient. Unter Windows findest du die Eingabeaufforderung entweder, indem du den Namen einfach in das Suchfeld im Startmenü eingibst, oder ebenfalls im Startmenü unter ZUBEHÖR. Unter macOS findest du das Terminal im Ordner `/Programme/Dienstprogramme` oder indem du in die Suche `Terminal` eingibst. In der Eingabeaufforderung beziehungsweise im Terminal gibst du dann den Befehl `javac` ein und bestätigst die Eingabe mit der Enter-Taste. Ist danach eine Ausgabe wie in Abbildung 1.3 zu sehen, ist der Java-Compiler bereits korrekt auf deinem Computer installiert und du kannst direkt weiter zu Abschnitt 1.3.2 springen. Bekommst du dagegen eine Meldung wie `Der Befehl "javac" ist entweder falsch geschrieben oder konnte nicht gefunden werden.` oder Ähnliches, so muss der Java-Compiler noch auf deinem Computer installiert werden.

```
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are u
sed
  -classpath <path>          Specify where to find user class files and annotati
on processors
  -cp <path>                 Specify where to find user class files and annotati
on processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compil
ation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors
to run; bypasses default discovery process
  -processorpath <path>      Specify where to find annotation processors
  -parameters                Generate metadata for reflection on method paramete
rs
  -d <directory>             Specify where to place generated class files
  -s <directory>             Specify where to place generated source files
  -h <directory>             Specify where to place generated native header file
s
  -implicit:{none,class}     Specify whether or not to generate class files for
implicitly referenced files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release

  -target <release>          Generate class files for specific VM version
  -profile <profile>         Check that API used is available in the specified p
rofile
  -version                   Version information
  -help                      Print a synopsis of standard options
  -Akey[=value]              Options to pass to annotation processors
  -X                         Print a synopsis of nonstandard options
  -J<flag>                   Pass <flag> directly to the runtime system
  -Werror                    Terminate compilation if warnings occur
  @<filename>                Read options and filenames from file
```

Abbildung 1.3: Ausgabe bei korrekt installiertem Java-Compiler

1.3.1 Java-Compiler installieren

Der Java-Compiler ist, wie auch die Java Virtual Machine, Teil des *Java Development Kit* (JDK) und kann kostenlos heruntergeladen werden. Einen Link zum Download findest du auf `buch.daniel-braun.com`.

Unter GNU/Linux kannst du Java direkt über den Paketmanager deiner Wahl installieren. Unter macOS und Windows lädst du zunächst ein gepacktes Verzeichnis herunter, das nach dem Download entpackt werden muss. Dieses Verzeichnis, das je nach Version zum Beispiel den Namen `jdk-20.0.2` trägt, kopierst du unter macOS in den Ordner `/library/Java/JavaVirtualMachines/` und unter Windows in ein beliebiges Verzeichnis deiner Wahl, zum Beispiel direkt in `C:\`. Unter Windows musst du dieses Verzeichnis dann noch zur sogenannte PATH-Variable hinzufügen. Dazu musst du zunächst die erweiterten Systemeinstellungen deines Computers öffnen.

Unter Windows 8, 10 und 11 kannst du die erweiterten Systemeinstellungen öffnen, indem du den Begriff einfach direkt in die Suche eingibst. Alternativ kannst du auch zunächst mit der rechten Maustaste auf das Windows-Logo in der unteren linken Ecke klicken und dort dann auf SYSTEM und in dem sich öffnenden Fenster wieder auf ERWEITERTE SYSTEMEINSTELLUNGEN.

Nun solltest du, unabhängig von deiner verwendeten Windows-Version, das in Abbildung 1.4 gezeigte Fenster sehen. Dort findest du in der rechten unteren Ecke einen Button mit der Beschriftung UMGEBUNGSVARIABLEN. Bei einem Klick darauf öffnet sich das in Abbildung 1.5 gezeigte Fenster.

Dort wählst du dann, wie in Abbildung 1.5 gezeigt, den Eintrag PATH aus und klickst anschließend auf BEARBEITEN. Sollte der Eintrag nicht vorhanden sein, so kannst du direkt zum nächsten Absatz springen. Danach öffnet sich ein langes Textfeld, in dem es schon zahlreiche Einträge gibt, die auf keinen Fall geändert werden dürfen. Stattdessen solltest du am Ende, abgetrennt durch ein Semikolon, den Pfad angeben, an dem du zuvor das Java Development Kit installiert hast. Standardmäßig sähe das so aus:

```
C:\jdk-20.0.2\bin;
```

Je nachdem, welche Java-Version du installiert hast, kann der Pfad aber, insbesondere bei der Versionsnummer, leicht abweichen. Daher solltest du unbedingt darauf achten, den tatsächlichen Installationspfad zu nutzen. Danach musst du die Änderungen nur noch mit OK und ÜBERNEHMEN bestätigen.

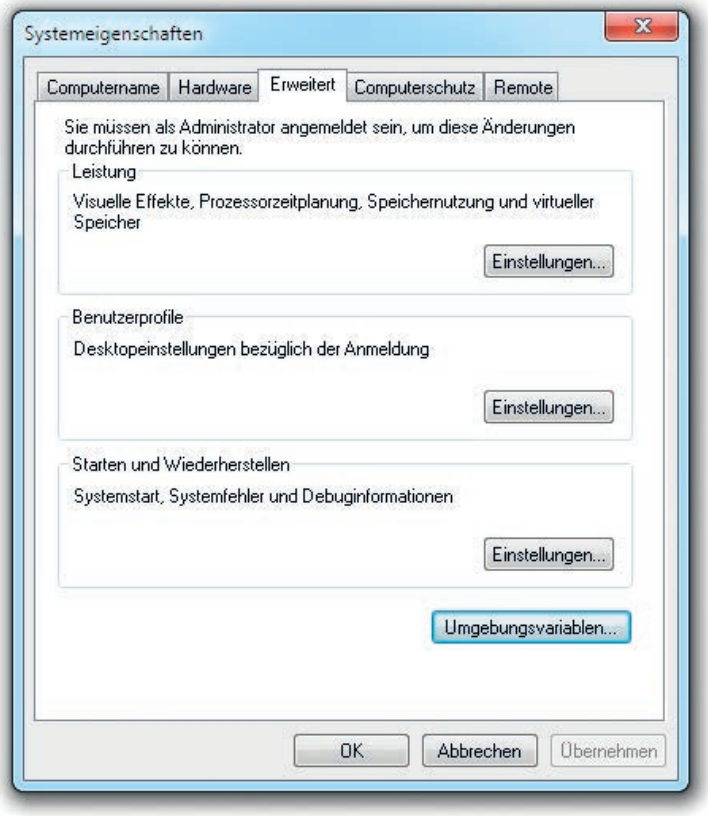

Abbildung 1.4: Erweiterte Systemeinstellungen

Sollte es bei dir noch keinen Eintrag mit dem Namen PATH geben, so kannst du diesen ganz einfach selbst anlegen. Dazu klickst du statt auf BEARBEITEN einfach auf NEU. Im Fenster, das sich daraufhin öffnet, gibst du als NAME DER VARIABLEN das Wort PATH ein und als WERT DER VARIABLEN den Pfad zur Installation, beendet durch ein Semikolon, und bestätigst deine Eingabe mit OK.

Anschließend sollte der `javac`-Befehl dann in der Eingabeaufforderung funktionieren.

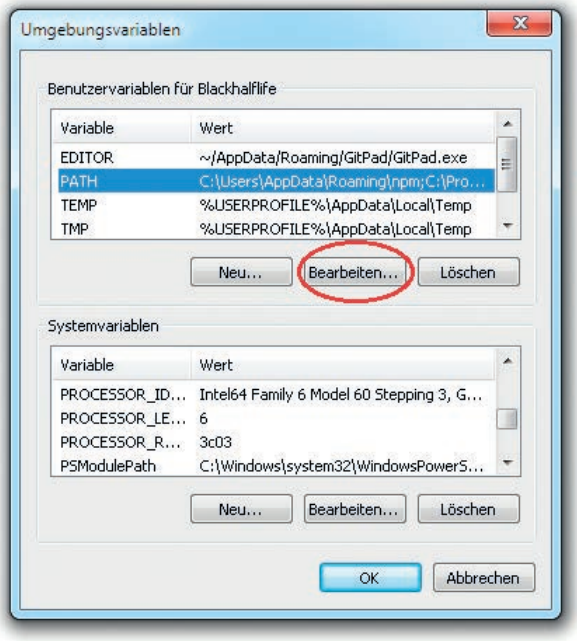

Abbildung 1.5: Umgebungsvariablen

1.3.2 Ordner einrichten

Im Verlaufe des Buches wirst du zahlreiche Plugins programmieren, einige davon auch in verschiedenen Versionen. Damit du darüber später leichter den Überblick behalten kannst, solltest du jetzt schon vorsorgen.

Am besten legst du einen eigenen Ordner an, in dem du später alle Projekte aus dem Buch speicherst. Prinzipiell kannst du diesen Ordner natürlich, wie den des Servers, wieder speichern, wo du möchtest. Es wird dir später aber das Leben erleichtern, wenn du ihn im selben Verzeichnis wie den Server-Ordner platzierst, unter Windows also zum Beispiel unter `C:\` und unter GNU/Linux und macOS unter `/home/Benutzername` beziehungsweise `/Benutzer/Benutzername`. Als Namen für den Ordner kannst du zum Beispiel einfach `plugins` wählen.

1.4 Editor

Damit sind auch fast alle Vorbereitungen abgeschlossen, die nötig sind, bevor es mit dem Programmieren losgehen kann. Was dir jetzt noch fehlt, ist ein Programm zum Schreiben deiner zukünftigen Plugins. Grundsätzlich kannst du dazu nahezu jedes Programm verwenden, mit dem man Texte verfassen kann. Der mit Windows mitgelieferte EDITOR, den du im Startmenü unter ZUBEHÖR findest, ist zum Beispiel völlig ausreichend. macOS bringt das Programm TEXTEDIT mit, das sich im Ordner `Programme` befindet oder über die Suche gefunden werden kann. Solltest du dich für den Windows-Editor entscheiden, so musst du beim SPEICHERN darauf achten, dass du als DATEITYP den Eintrag ALLE DATEIEN auswählst. Beim Programm TEXTEDIT sollte nach dem Neuanlegen eines Dokuments der Menüpunkt FORMAT|IN REINEN TEXT UMWANDELN ausgewählt werden. Und unabhängig davon, welches Programm du verwendest, solltest du beim Speichern an den Dateinamen die Endung `.java` anhängen, damit dein Computer weiß, dass es sich bei der gespeicherten Datei um Java-Code handelt.

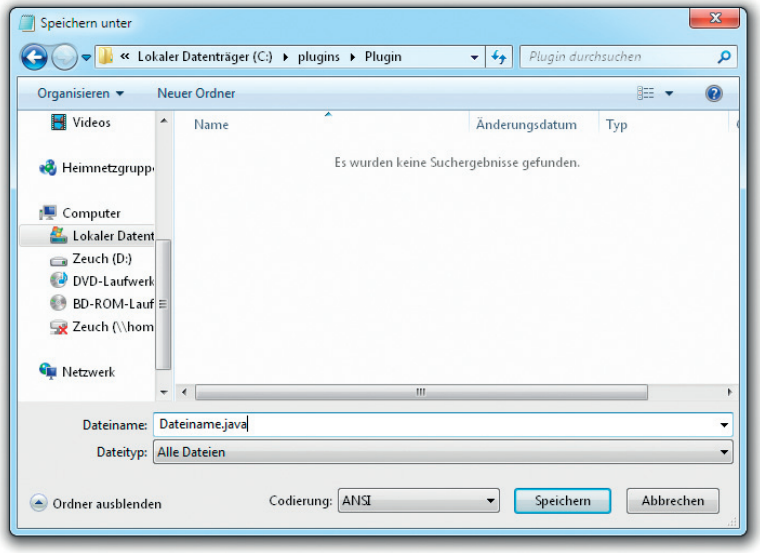

Abbildung 1.6: Speichern von Dateien mit dem Windows-Editor

Wenn du es gerne etwas komfortabler hättest, kannst du aber auch einen Editor wählen, der speziell dafür entwickelt wurde, Java-Programme zu schreiben. Solche Editoren bieten dir in der Regel zahlreiche Komfortfunktionen wie das automatische Einfärben von Quellcode, automatisches Einrücken oder sogar eine automatische Vervollständigung

an. Einige Editoren, die besonders viele solcher Zusatzfunktionen mitbringen, nennt man auch **integrierte Entwicklungsumgebungen**, oder englisch *Integrated Development Environment*, kurz **IDE**s. Zu den bekanntesten Java-IDEs gehören zum Beispiel Eclipse und NetBeans.

Diese sind mitunter allerdings sehr komplex zu bedienen. Fürs Erste solltest du daher vielleicht einen etwas weniger umfangreichen Editor wählen, da du die meisten Funktionen der großen IDEs anfangs ohnehin nicht nutzen wirst. Der Editor jEdit, der kostenlos für alle gängigen Betriebssysteme erhältlich ist, bietet sich dafür zum Beispiel an. Den Link zum Download sowie eine Installationsanleitung findest du ebenfalls unter `buch.daniel-braun.com`.

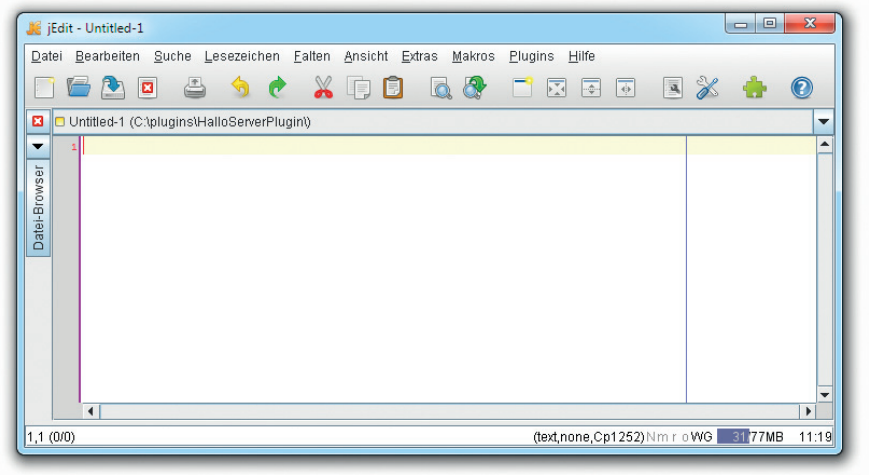

Abbildung 1.7: jEdit

1.5 Zusammenfassung

Begriff	Bedeutung
Algorithmus	Eine eindeutige Handlungsvorschrift, die festlegt, was genau zum Beispiel ein Programm tun soll.
Compiler	Ein Programm, das Programmiersprache in Maschinensprache übersetzt.
Binärdatei	Eine Datei, die Maschinensprache enthält, die nur aus Nullen und Einsen besteht.