Daniel Braun

5. Auflage

mitp

LET'S PLAY

# Programmieren lernen mit Java und Minecraft

Für Bukkit & Spigot unter Windows, Linux und macOS

## Plugins erstellen ohne Vorkenntnisse

Daniel Braun

# Let's Play Programmieren lernen mit Java und Minecraft

## Plugins erstellen ohne Vorkenntnisse

**Bibliografische Information der Deutschen Nationalbibliothek**
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-7475-0782-7
5. Auflage 2023

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082



Lektorat: Sabine Schulz, Nicole Winkel
Sprachkorrektorat: Petra Heubach-Erdmann, Knut Lorenzen
Coverbild: Daniel Braun
Satz: III-satz, Kiel, www.drei-satz.de

# Inhalt

# Einleitung

Liebe Leserinnen und Leser,

die Welt von Minecraft steckt voller Dinge, die es zu entdecken gilt. Verschiedene Landschaften, Hunderte verschiedene Gegenstände und allerlei Tiere und Monster sind nur einige der Dinge, die dich erwarten.

Irgendwann ist aber selbst diese Vielzahl an Möglichkeiten erschöpft und man hat das Gefühl, alles schon einmal gesehen oder gemacht zu haben. Wenn es dir so geht, dann ist dieses Buch genau das Richtige für dich. Denn im Verlaufe dieses Buches lernst du, wie man mithilfe von Java und dem Bukkit- oder Spigot-Server eigene Erweiterungen für Minecraft programmiert, sogenannte Plugins, die du dann zusammen mit deinen Freunden auf deinem eigenen Minecraft-Server ausprobieren kannst.

Egal ob du neue Crafting-Rezepte entwerfen möchtest, ganze Häuser mit einem einfachen Chat-Befehl bauen oder sogar einen eigenen Spielmodus programmieren möchtest, mit eigenen Plugins steckt die Welt von Minecraft wieder voller Herausforderungen und Dingen, die entdeckt werden wollen. Und ganz nebenbei lernst du auch noch zu programmieren – und wer weiß, vielleicht kommt das nächste Minecraft eines Tages von dir!

Bevor es so weit ist, liegt allerdings noch ein ordentliches Stück Weg vor dir. Die ersten beiden Kapitel dieses Buches beschäftigen sich deshalb zunächst einmal damit, wie du deinen Computer für das Programmieren und Testen eigener Plugins vorbereitest. Dazu wird dir erklärt, wie du den Bukkit- oder Spigot-Server installierst, der in diesem Buch verwendet wird, ihn nach deinen Wünschen konfigurierst und wie du deinen Computer so einrichtest, dass du Java-Programme schreiben kannst.

Direkt im Anschluss geht es im dritten Kapitel ohne Umschweife direkt los mit dem Programmieren deines ersten eigenen Plugins. Die ersten Schritte werden dir vielleicht noch etwas unspektakulär vorkommen, aber mit jedem der folgenden Kapitel wirst du immer mehr Möglichkeiten haben, um immer ausgeklügeltere Plugins zu programmieren. Schon im vierten Kapitel wirst du zum Beispiel lernen, wie du eigene Chat-Befehle programmieren und verwenden kannst.

In Kapitel 5 lernst du Eclipse kennen, einen Editor, der dich beim Programmieren von Plugins mit vielen nützlichen Funktionen unterstützen kann. Die Kapitel 6 bis 10 beschäftigen sich mit grundlegenden Konzepten des Programmierens im Allgemeinen und der Programmiersprache Java im Besonderen. Was du hier liest, wird dir nicht nur beim Programmieren von Minecraft-Plugins helfen, sondern beim Programmieren jedes Programms in jeder Programmiersprache. Trotzdem entstehen dabei natürlich auch einige praktische kleine Plugins wie zum Beispiel das Mauer-Plugin, das es dir erlaubt, mit einem einfachen Chat-Befehl auf die Schnelle eine Mauer zu bauen – wenn du möchtest, sogar aus purem Gold.

Das elfte Kapitel widmet sich dann ganz der Baukunst. Häuser, Schilder, Kreise und Kugeln – hier wird kein Block auf dem anderen gelassen. Und wenn du schon einmal versucht hast, eine Kugel in Minecraft von Hand zu bauen, dann wirst du ganz besonders die Dienste des Kugel-Plugins zu schätzen wissen, das dir auf Knopfdruck eine nahezu perfekte Kugel zaubern kann. Weiter geht es danach mit dem Bau von Schildern, denen das gesamte zwölfte Kapitel gewidmet ist.

Und wenn dir selbst ein Knopfdruck noch zu viel ist, dann wird dir das dreizehnte Kapitel besonders gefallen. Dort geht es nämlich um Plugins, die vollautomatisch auf Geschehnisse in der Spielwelt reagieren. Egal ob ein Creeper über die Karte schleicht, ein Spieler etwas isst oder ein Baum wächst: Hier lernst du, wie deinem Plugin nichts mehr von dem entgeht, was auf deinem Server passiert, und natürlich auch, wie du darauf reagieren kannst.

Falls du dich um die umherschleichenden Creeper aber doch lieber ganz manuell kümmern möchtest, kannst du die Informationen aus Kapitel 14 nutzen, um ganz eigene Waffen zu kreieren. In diesem Kapitel geht es nämlich um das Erstellen eigener Crafting-Rezepte und ein Beispiel, das dir dort begegnen wird, ist ein Rezept für ein Flammenschwert, das alles in Brand setzt, worauf es trifft.

Kapitel 15 ist dann wieder etwas technischer, aber nicht weniger nützlich. Hier lernst du nämlich, wie du Informationen dauerhaft speichern kannst, die auch dann erhalten bleiben, wenn der Server zwischenzeitlich ausgeschaltet wird. Das ist zum Beispiel praktisch, wenn du wie in Kapitel 16 eigene Spielmodi kreieren willst, also sozusagen ein Spiel im Spiel. Wie wäre es zum Beispiel mit einem Schneeballschlacht-Mod mit eigener Highscore-Liste, die die Treffer zählt? Oder lieber ein lustiges Suchspiel, bei dem der Gewinner mit Erfahrungspunkten oder wertvollen Gegenständen belohnt wird? Ganz wie du möchtest: Deiner Kreativität sind keine Grenzen gesetzt!

Im letzten Kapitel bekommst du dann noch einen kurzen Ausblick darauf, was du mit deinen neu gewonnenen Programmierfähigkeiten noch anstellen kannst, außer Minecraft-Plugins zu programmieren. Denn wenn du am Ende des Buches angelangt bist, hört der Spaß noch lange nicht auf, denn dann hast du alle Werkzeuge und alles Wissen, das du benötigst, um ganz eigene Plugins, ganz nach deinen Vorstellungen zu entwerfen. Dabei helfen dir einige Listen im Anhang des Buches, in denen du Befehle und besonders häufig benötigte Dinge schnell nachschlagen kannst. Denn egal wie erfahren man als Programmierer ist, alles kann und muss man nicht auswendig können, man muss nur wissen, wo man es nachschlagen kann – und genau dazu dient der Anhang dieses Buches.

Für Fragen, Kritik oder Anregungen zum Buch oder generell zu Minecraft-Plugins, kannst du mich gerne jederzeit kontaktieren. Du erreichst mich per Mail an `info@daniel-braun.com` oder über meine Website *www.daniel-braun.com*.

## Downloads zum Buch

Unter der Webadresse *buch.daniel-braun.com* findest du:

- Links zu allen Downloads, die du benötigst
- Alle Plugins, die du im Rahmen des Buches programmieren wirst, falls du den Code nicht aus dem Buch abtippen möchtest

Mein besonderer Dank gilt Karl-Heinz Barzen, der den Entstehungsprozess dieses Buches unermüdlich mit zahlreichen hilfreichen Kommentaren und Anmerkungen begleitet und damit einen wichtigen Beitrag dazu geleistet hat, dass dieses Buch möglichst verständlich und einsteigerfreundlich wird.

Nun wünsche ich dir aber vor allem viel Spaß beim Lesen, Programmieren und Entdecken!

Daniel Braun

# 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.

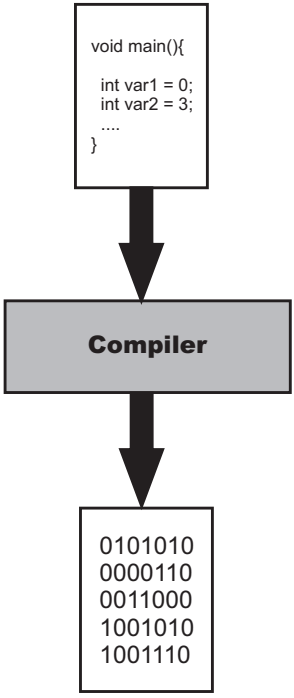


**Abbildung 1.1:** Funktionsweise eines Compilers

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.

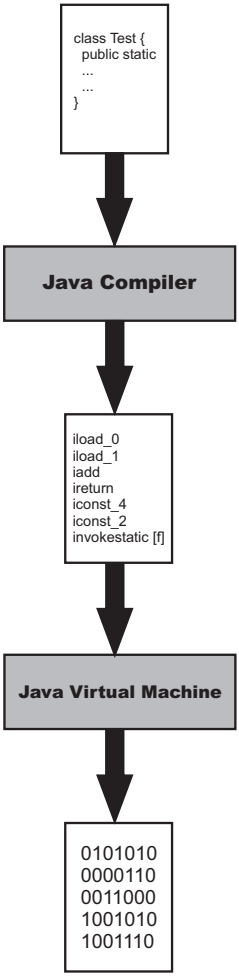


**Abbildung 1.2:** Funktionsweise des Java-Compilers

## 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.

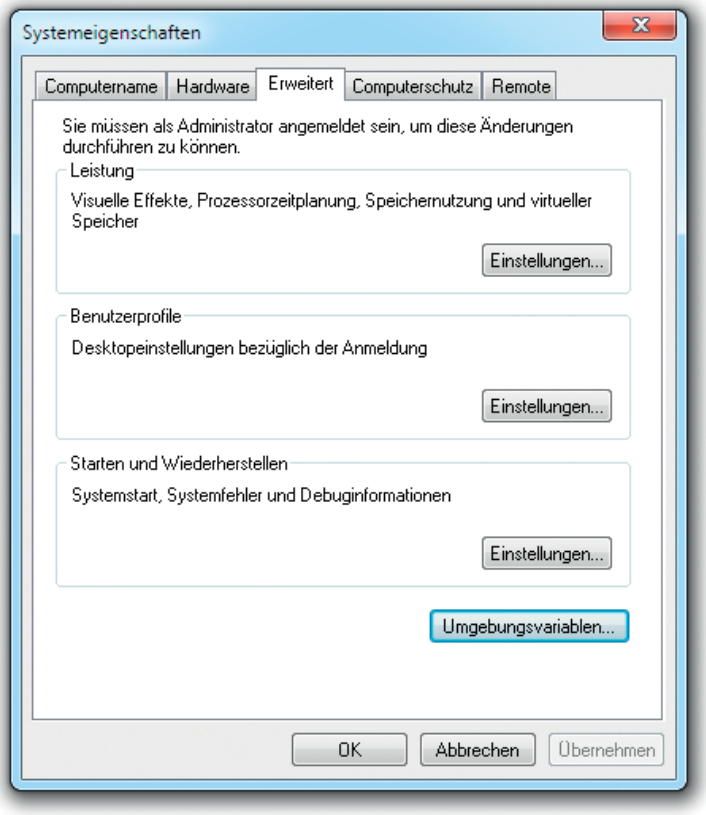


**Abbildung 1.4:** Erweiterte Systemeinstellungen

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.

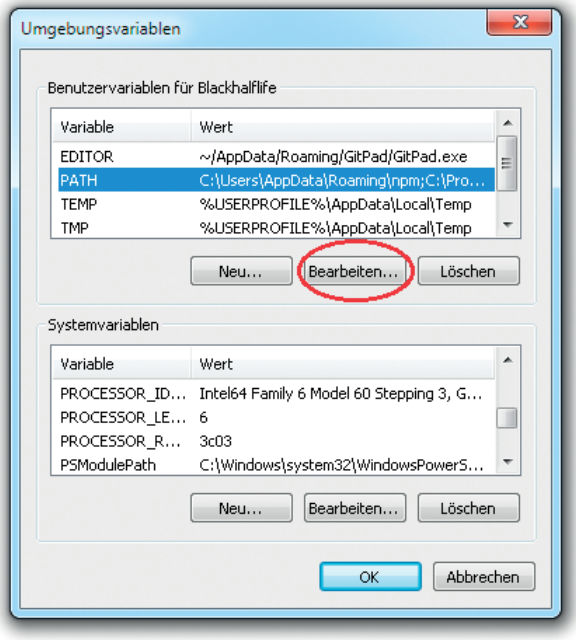


**Abbildung 1.5:** Umgebungsvariablen

## 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.

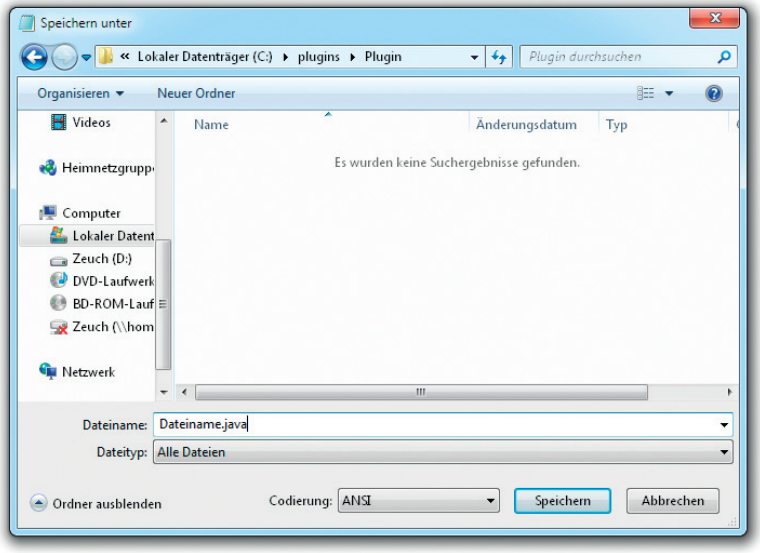


**Abbildung 1.6:** Speichern von Dateien mit dem Windows-Editor

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`.

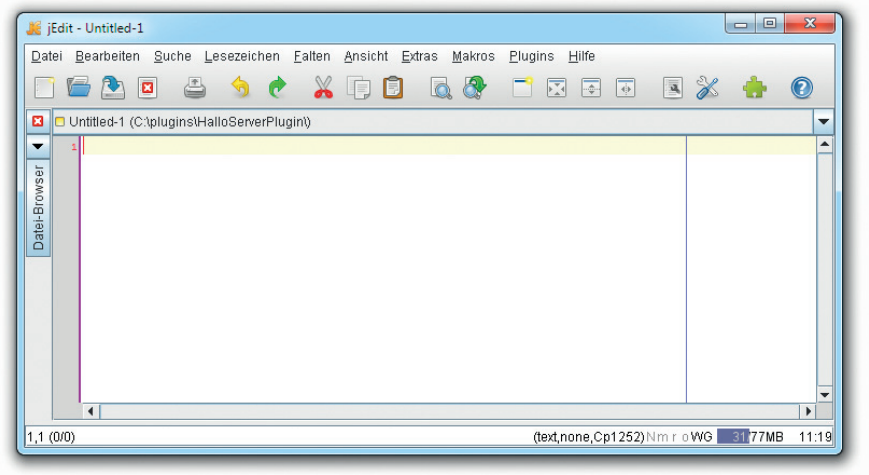


**Abbildung 1.7:** jEdit

## 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. |

# Kapitel 2

# Minecraft-Server

Alleine Minecraft zu spielen, kann schon jede Menge Spaß machen, noch lustiger wird es aber, wenn du dich mit anderen Spielern zusammentust, um mit ihnen oder auch gegen sie zu spielen. Dazu kannst du dir entweder einen der Hunderten öffentlichen Server aussuchen, die du überall im Internet findest, oder du kannst deinen eigenen Server nutzen – dann hast du die volle Kontrolle über alle Einstellungen. Noch mehr Spaß wird dir dein eigener Server machen, wenn du im Laufe des Buches lernst, immer ausgefeiltere Plugins für ihn zu programmieren, mit denen du Minecraft nach deinen Vorstellungen erweitern kannst.

Um deinen eigenen Server zu betreiben, benötigst du neben dem normalen Minecraft-Spiel, das auch **Client** genannt wird, noch ein weiteres Programm, nämlich den Minecraft-**Server**. Den »normalen« Minecraft-Server, manchmal auch »Vanilla-Server« genannt, kannst du auf der offiziellen Minecraft-Webseite *www.minecraft.net* herunterladen. Neben dieser Version gibt es aber auch noch zahlreiche sogenannte Mods, also Modifikationen des Original-Servers. Als Mods oder Modifikationen bezeichnet man im Zusammenhang mit Spielen Versionen eines Spiels, die in irgendeiner Form verändert, also modifiziert wurden. Diese meist von Fans entwickelten Mods bieten häufig viele zusätzliche Funktionen und Annehmlichkeiten, über die der Vanilla-Server nicht verfügt, wie zum Beispiel auch die Möglichkeit, eigene Plugins zu programmieren.

**Merke**

Das normale Minecraft-Spiel, das du auch startest, wenn du alleine spielst, wird **Client** genannt. Das Programm, das wir in diesem Kapitel installieren werden, das du benötigst, um mit Freunden zusammen spielen zu können, heißt hingegen **Server**.

Dieses Buch ist für gleich zwei der beliebtesten Server ausgelegt. Du kannst dich entscheiden zwischen dem **CraftBukkit**-Server, häufig auch einfach nur Bukkit genannt, und dem **Spigot**-Server. Da der Spigot- auf dem Bukkit-Server aufbaut, funktionieren alle Plugins, die wir im Rahmen dieses Buches programmieren werden, auf beiden Servern. Der einzige Unterschied liegt in der Administration der Server, hier bietet Spigot mehr Möglichkeiten, ist dafür in der Bedienung aber auch etwas komplexer. Außerdem ist der Spigot-Server etwas effizienter, was bedeutet, dass er insbesondere etwas weniger Arbeitsspeicher (RAM) benötigt. Für Anfänger, die zum ersten Mal einen eigenen Server betreiben, ist es daher ratsam, zunächst auf Bukkit zu setzen; wer schon Erfah-

rung mit der Verwaltung eines Minecraft-Servers hat, kann sich auch an Spigot herantrauen. Ein Wechsel ist ohnehin jederzeit möglich.

# 2.1 Installation

An dieser Stelle musst du dich nun entscheiden, welchen Server du zum Testen deiner Plugins verwenden möchtest. Wenn du dich für den Bukkit-Server entscheidest, kannst du in Abschnitt 2.1.1 weiterlesen; möchtest du lieber den Spigot-Server verwenden, dann kannst du direkt zu Abschnitt 2.1.2 springen.

## 2.1.1 CraftBukkit

Einen Link zum Download der neuesten Version des Bukkit-Servers findest du auf der Website zum Buch unter *buch.daniel-braun.com*. Dabei handelt es sich um eine einzelne sogenannte Jar-Datei, die, je nach Version, zum Beispiel den Namen `craftbukkit-1.20.1.jar` trägt. Zunächst solltest du einen leeren Ordner anlegen, in den du diese Datei kopierst. Prinzipiell kannst du diesen Ordner nennen, wie du möchtest, im Verlaufe des Buches werden wir davon ausgehen, dass der Ordner den Namen `server` trägt und in `C:\server` unter Windows, `/home/Benutzername/server` unter GNU/Linux beziehungsweise `/Users/Benutzername/server` unter macOS abgelegt ist.

Um den Server nun zum ersten Mal zu starten, musst du zunächst wieder die Eingabeaufforderung beziehungsweise ein Terminal öffnen und in den Server-Ordner wechseln. Das kannst du mithilfe des Befehls `cd`. Die englische Abkürzung steht für »change directory«, also »Ordner wechseln«, und genau das, also zwischen verschiedenen Ordnern hin- und herwechseln, kann man mit diesem Befehl auch tun. Unter Windows gibst du also zum Beispiel `cd C:\server` ein und unter GNU/Linux `cd /home/Benutername/server`. Bist du erst einmal im richtigen Ordner, so kannst du den Server mit dem Befehl `java -jar craftbukkit-1.20.1.jar` starten. Beim ersten Starten wirst du aber zunächst einmal nur die in Abbildung 2.1 gezeigten Warnhinweise sehen.

**Merke**

Der Server wird mit dem Befehl `java -jar craftbukkit-1.20.1.jar` gestartet. Achte darauf, die Versionsnummer im Befehl an die von dir verwendete Server-Version anzupassen.

Dort steht im Wesentlichen, dass du zunächst den Nutzungsbedingungen zustimmen musst, bevor du den Server verwenden kannst. Wenn du jetzt einen Blick in deinen Server-Ordner wirfst, dann wird dir auffallen, dass es dort, wie in Abbildung 2.2, nun zwei weitere Dateien und einen Ordner gibt.

```
C:\server>java -jar craftbukkit-1.20.1.jar
Unbundling libraries to C:\server\bundler
Starting server
Loading libraries, please wait...
[17:53:40] [ServerMain/INFO]: You need to agree to the EULA in order to run the server. Go to eula.txt for more info.
```

**Abbildung 2.1:** Ausgabe nach dem ersten Starten des Servers


**Abbildung 2.2:** Inhalt des Server-Ordners nach dem ersten Start

Um den Nutzungsbedingungen zuzustimmen, musst du die dort nun vorhandene Datei `eula.txt` öffnen. In dieser Datei findest du auch einen Link, unter dem du die Bedingungen lesen kannst. Wenn du diesen Link öffnest, wirst du auf die offizielle Seite des Minecraft-Herstellers Mojang geleitet, wo du die Nutzungsbedingungen glücklicherweise auch auf Deutsch vorfindest. Dort wird geregelt, was du mit dem Spiel und dem Server machen darfst – und was nicht. Außerdem steht dort auch explizit, dass du, solltest du unter 18 sein, die Zustimmung eines gesetzlichen Vertreters einholen musst, also zum Beispiel eines Elternteils. Auf jeden Fall solltest du die Bedingungen sorgfältig lesen.

Den Inhalt der `eula.txt` findest du auch in Listing 2.1. Bist du mit den Bedingungen einverstanden, so kannst du dies kenntlich machen, indem du die letzte Zeile der Datei von `eula=false` zu `eula=true` änderst. Nur wenn du das tust, kannst du den Server benutzen. Genau das wird in der ersten Zeile der Datei auf Englisch erklärt.

```
#By changing the setting below to TRUE you are indicating your agreement
to our EULA (https://account.mojang.com/documents/minecraft_eula).
#Mon Apr 01 13:37:00 BST 2021
eula=false
```

**Listing 2.1:** Inhalt der Datei `eula.txt`

Wenn du die Änderungen gespeichert hast, kannst du wieder versuchen, den Server mit dem Befehl `java -jar craftbukkit-1.20.1.jar` zu starten. Der Startvorgang wird dieses Mal wahrscheinlich eine Weile dauern und es werden sehr viele Zeilen relativ schnell über den Bildschirm laufen. Wichtig ist besonders die letzte Zeile. Steht dort so etwas wie `Done (13,370s)! For help, type "help" or "?"`, dann bedeutet das, dass dein Server nun problemlos läuft. Ein erneuter Blick in den Server-Ordner wird dir zeigen, dass es dort nun, wie in Abbildung 2.3 zu sehen, noch einmal deutlich mehr Dateien gibt.

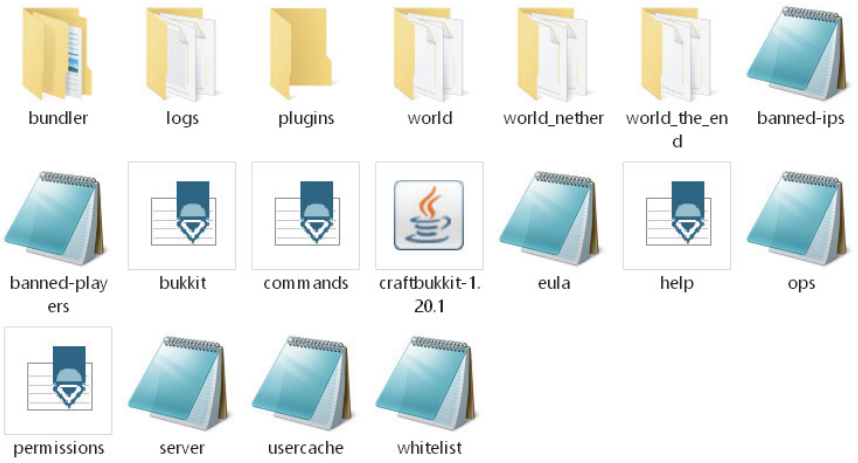


**Abbildung 2.3:** Inhalt des Server-Ordners nach erfolgreichem Starten des Servers

| Hinweis |
| --- |
| Der Server läuft nur, solange das entsprechende Fenster der Eingabeaufforderung beziehungsweise des Terminals geöffnet bleibt. Schließt du das Fenster, so wird auch der Server geschlossen. |

## 2.1.2 Spigot

Einen Link zum Download der neuesten Version des Spigot-Servers findest auf der Website zum Buch unter *buch.daniel-braun.com*. Dabei handelt es sich um eine einzelne sogenannte Jar-Datei, die, je nach Version, zum Beispiel den Namen `spigot-1.20.1.jar` trägt. Zunächst solltest du einen leeren Ordner anlegen, in den du diese Datei kopierst. Prinzipiell kannst du diesen Ordner nennen, wie du möchtest, im Verlaufe des Buches werden wir davon ausgehen, dass der Ordner den Namen `server` trägt und unter Windows in `C:\server`, unter GNU/Linux in `/home/Benutzername/server` beziehungsweise unter macOS in `/Users/Benutzername/server` abgelegt ist.

Um den Server zum ersten Mal zu starten, musst du zunächst wieder die Eingabeaufforderung beziehungsweise ein Terminal öffnen, und in den Server-Ordner wechseln. Das kannst du mithilfe des Befehls `cd`. Unter Windows gibst du also zum Beispiel `cd C:\server` ein und unter GNU/Linux `cd /home/Benutzername/server`. Bist du erst einmal im richtigen Ordner, so kannst du den Server mit dem Befehl `java -jar spigot-1.20.1.jar` starten. Beim ersten Starten wirst du aber zunächst einmal nur die in Abbildung 2.4 gezeigten Warnhinweise sehen.

```
C:\server>java -jar spigot-1.20.1.jar
Unbundling libraries to C:\server\bundler
Starting server
Loading libraries, please wait...
[17:47:41] [ServerMain/INFO]: You need to agree to the EULA in order to run the server. Go to eula.txt for more info.
```

**Abbildung 2.4:** Ausgabe nach dem ersten Starten des Servers

### Merke

Der Server wird mit dem Befehl `java -jar spigot-1.20.1.jar` gestartet. Achte darauf, die Versionsnummer im Befehl an die von dir verwendete Server-Version anzupassen.

### Tipp

Sollte beim Starten des Servers folgender Hinweis angezeigt werden:

```
*** Error, this build is outdated ***
*** Please download a new build as per instructions from
    https://www.spigotmc.org/go/outdated-spigot ***
*** Server will start in 20 seconds ***
```

dann bedeutet das, dass du nicht die aktuellste Version des Servers benutzt. Du kannst den Server trotzdem weiterhin wie gewohnt nutzen, oder eine neuere Version aus dem Internet herunterladen.

Dort steht im Wesentlichen, dass du zunächst den Nutzungsbedingungen zustimmen musst, bevor du den Server verwenden kannst. Wenn du jetzt einen Blick in deinen Server-Ordner wirfst, dann wird dir auffallen, dass es dort, wie in Abbildung 2.5, nun zwei weitere Dateien und einen Ordner gibt.

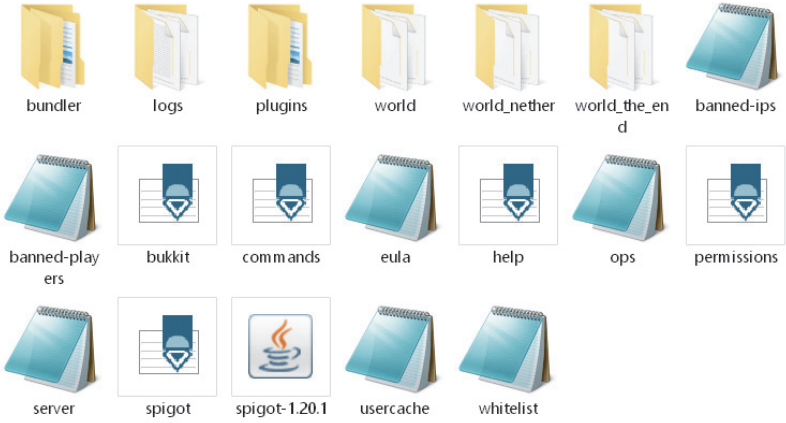


**Abbildung 2.5:** Inhalt des Server-Ordners nach dem ersten Start

Um den Nutzungsbedingungen zuzustimmen, musst du die dort nun vorhandene Datei `eula.txt` öffnen. In dieser Datei findest du auch einen Link, unter dem du die Bedingungen lesen kannst. Wenn du diesen Link öffnest, wirst du auf die offizielle Seite des Minecraft-Herstellers Mojang geleitet, wo du die Nutzungsbedingungen glücklicherweise auch auf Deutsch vorfindest. Dort wird geregelt, was du mit dem Spiel und dem Server machen darfst – und was nicht. Außerdem steht dort auch explizit, dass du, solltest du unter 18 sein, die Zustimmung eines gesetzlichen Vertreters einholen musst, also zum Beispiel eines Elternteils. Auf jeden Fall solltest du die Bedingungen sorgfältig lesen.

Den Inhalt der `eula.txt` findest du auch in Listing 2.1. Bist du mit den Bedingungen einverstanden, so kannst du dies kenntlich machen, indem du die letzte Zeile der Datei von `eula=false` zu `eula=true` änderst. Nur wenn du das tust, kannst du den Server benutzen. Genau das wird in der ersten Zeile der Datei auf Englisch erklärt.

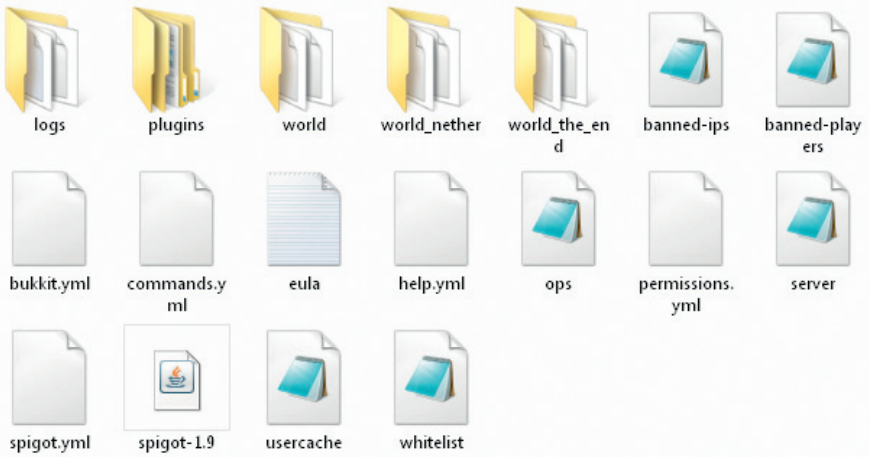


**Abbildung 2.6:** Inhalt des Server-Ordners nach erfolgreichem Starten des Servers

Wenn du die Änderungen gespeichert hast, kannst du wieder versuchen, den Server mit dem Befehl `java -jar spigot-1.20.1.jar` zu starten. Der Startvorgang wird dieses Mal wahrscheinlich eine Weile dauern und es werden sehr viele Zeilen relativ schnell über den Bildschirm laufen. Wichtig ist besonders die letzte Zeile. Steht dort so etwas wie `Done (13,370s)! For help, type "help" or "?"`, dann bedeutet das, dass dein Server nun problemlos läuft. Ein erneuter Blick in den Server-Ordner wird dir zeigen, dass es dort nun, wie in Abbildung 2.6 zu sehen, noch einmal deutlich mehr Dateien gibt.

**Tipp**

Wenn beim Versuch, den Server zu starten, die Fehlermeldung `java.lang.OutOfMemoryError` erscheint und der Server wieder heruntergefahren wird, dann steht der JVM nicht genug Speicher zur Verfügung. Wenn du beim Starten vor dem Dateinamen `java -Xms2048M -Xmx2048M -jar` eingibst statt nur java -jar, dann wird der Speicher erhöht und der Server kann starten.

## 2.2 Konfiguration

Die Konfiguration der Server funktioniert für beide Versionen sehr ähnlich. Größter und offensichtlicher Unterschied ist es hier, dass der Spigot-Server über eine zusätzliche Datei, die `spigot.yml`, verfügt.

In den Ordnern `world`, `world_nether` und `world_the_end` werden, unabhängig vom verwendeten Server, Informationen über die Spielwelt gespeichert. Im Ordner `logs` werden sogenannte **Log-Dateien** gespeichert, diese Dateien enthalten im Wesentlichen alle Informationen, die dir auch in der Eingabeaufforderung beziehungsweise dem Terminal angezeigt werden. Das ist besonders später beim Programmieren von Plugins praktisch, denn sollte es einmal zu einem Fehler kommen, so kannst du die genaue Fehlermeldung hier in Ruhe nachlesen. Der Ordner `plugins` ist zu Beginn noch leer, hier werden wir später unsere selbst geschriebenen Plugins speichern.

Zunächst einmal interessieren uns aber vor allem die zahlreichen `.properties`-, `.json`- und `.yml`-Dateien, die erzeugt wurden. Mit diesen kannst du deinen Server nämlich konfigurieren und ihn nach deinen Wünschen anpassen.

### Die Datei server.properties

Die wichtigsten Grundeinstellungen findest du in der Datei `server.properties`. 35 verschiedene Einstellungen kannst du hier insgesamt vornehmen. Welche das sind, kannst du in Tabelle 2.1 sehen. Am Anfang kannst du aber ruhig auch alle Einstellungen unverändert lassen, dann wird dein Server auf jeden Fall problemlos funktionieren.

| Einstellung | Erklärung |
|---|---|
| `spawn-porection=16` | Legt fest, in welchem Radius um den Spawn-Punkt Blöcke unzerstörbar sind. |
| `generator-settings=` | Ist der Welttyp FLAT oder CUSTOMIZED (s. `level-type`), können hier Optionen für die Generierung festgelegt werden. Für den Welttyp FLAT erzeugt zum Beispiel `3;minecraft:bedrock, 2*minecraft: dirt,minecraft:grass;1;village` eine Ebene mit Dörfern. |

**Tabelle 2.1:** Einstellungsmöglichkeiten der `server.properties`

| Einstellung | Erklärung |
|---|---|
| `op-permission-level=4` | Bestimmt, welche Rechte Nutzer mit dem Status *Operator* haben (1 = können geschützten Spawnbereich verändern, 2 = können Befehlsblöcke editieren und Chat-Befehle ausführen, 3 = können Spieler verbannen, kicken und zum Operator ernennen, 4 = können den Server stoppen). |
| `allow-nether=true` | Aktiviert (true) oder deaktiviert (false) Nether-Portale. |
| `level-name=world` | Der Name des Ordners, in dem sich die Welt-Datei befindet. |
| `enable-query=false` | Aktiviert (true) oder deaktiviert (false) die Schnittstelle zum Abfragen von Server-Informationen. |
| `allow-flight=false` | Erlaubt (true) oder verbietet (false) Spielern, im Überlebensmodus zu fliegen. |
| `announce-player-achievements=true` | Aktiviert (true) oder deaktiviert (false) Nachrichten an alle Spieler, wenn ein Spieler ein Achievement erzielt. |
| `server-port=25565` | Legt den Port des Servers fest. |
| `max-world-size=29999984` | Legt die Größe der Welt fest (maximal 30.000.000, größere Werte werden ignoriert). |
| `level-type=DEFAULT` | Legt den Welttyp fest (DEFAULT = Standardwelt, FLAT = komplett flache Welt, LARGEBIOMES = große Biome, AMPLIFIED = Welt mit extremen Höhenunterschieden, CUSTOMIZED = individuelle Welt nach den Einstellungen in `generator-settings`). |
| `enable-rcon=false` | Aktiviert (true) oder deaktiviert (false) den Fernzugriff auf die Server-Konsole. |
| `level-seed=` | Erlaubt die manuelle Eingabe eines Startwerts (Seed) für die Generierung der Welt. |
| `force-gamemode=false` | Legt fest, ob Spieler beim Betreten in den Spielmodus zurückkehren, in dem sie den Server verlassen haben (false), oder immer im Standardmodus (true) starten. |
| `server-ip=` | Soll der Server nur unter einer bestimmten IP erreichbar sein, so kann diese hier eingetragen werden. |
| `network-compression-threshold=256` | Legt die Kompressionsstärke der Datenübertragung fest. |
| `max-build-height=256` | Legt die maximale Bauhöhe fest. |
| `spawn-npcs=true` | Aktiviert (true) oder deaktiviert (false) die Generierung von Dorfbewohnern. |
| `white-list=false` | Legt fest, ob nur Spieler, die sich auf der Whitelist befinden, den Server betreten dürfen (true) oder alle Spieler, die nicht verbannt sind (false). |

**Tabelle 2.1:** Einstellungsmöglichkeiten der `server.properties` (Forts.)

| Einstellung | Erklärung |
|---|---|
| `spawn-animals=true` | Aktiviert (true) oder deaktiviert (false) die Generierung von Tieren. |
| `hardcore=false` | Aktiviert (true) oder deaktiviert (false) den Hardcore-Modus (Spieler werden dauerhaft gebannt, sobald sie sterben). |
| `snooper-enabled=true` | Aktiviert (true) oder deaktiviert (false) das Senden von anonymisierten Server-Daten an Mojang. |
| `resource-pack-sha1=` | Prüfsumme des Ressourcenpakets, kann genutzt werden, um zu überprüfen, dass das Paket nicht verändert wurde. |
| `online-mode=true` | Gleicht verbundene Spieler mit der Datenbank von Mojang ab, falls aktiviert (true). Verhindert Fake-Accounts. |
| `resource-pack=` | Legt das empfohlene Ressourcenpaket des Servers fest. |
| `pvp=true` | Legt fest, ob sich Spieler gegenseitig angreifen können (true) oder nicht (false). |
| `difficulty=1` | Legt den Schwierigkeitsgrad fest, von 0 (friedlich) bis 3 (schwer). |
| `enable-command-block=false` | Aktiviert (true) oder deaktiviert (false) Befehlsblöcke. |
| `gamemode=0` | Legt den Spielmodus fest (0 = Überlebensmodus, 1 = Kreativmodus, 2 = Abenteuermodus, 3 = Zuschauermodus). |
| `player-idle-timeout=0` | Legt fest, nach wie vielen Minuten inaktive Spieler vom Server gekickt werden (0 = überhaupt nicht). |
| `max-players=20` | Legt die Zahl der maximal auf dem Server erlaubten Spieler fest. |
| `max-tick-time=60000` | Schaltet den Server automatisch ab, wenn zwischen zwei Aktualisierungen (Ticks) mehr als die angegebene Zahl von Millisekunden vergeht (-1 = deaktiviert). |
| `spawn-monsters=true` | Aktiviert (true) oder deaktiviert (false) die Generierung von Monstern. |
| `generate-structures=true` | Aktiviert (true) oder deaktiviert (false) die Generierung von Dörfern, Tempeln und anderen Gebäuden. |
| `view-distance=10` | Legt die Sichtweite fest. |
| `motd=A Minecraft Server` | Text, der in der Serverliste als Beschreibung angezeigt wird. |

**Tabelle 2.1:** Einstellungsmöglichkeiten der `server.properties` (Forts.)

### Die Datei bukkit.yml

Die zweite wichtige Datei mit Einstellungen, die, trotz des Namens, sowohl bei Bukkit als auch bei Spigot vorhanden ist, ist die `bukkit.yml`. Sie bietet noch einmal 24 weitere Einstellmöglichkeiten, die du in Tabelle 2.2 finden kannst.

| Einstellung | Erklärung |
|---|---|
| `allow-end: true` | Aktiviert (true) oder deaktiviert (false) Endportale. |
| `warn-on-overload: true` | Aktiviert (true) oder deaktiviert (false) Warnhinweis bei Überlastung des Servers. |
| `permissions-file: permissions.yml` | Dateiname der Datei, die die Berechtigungen festlegt. |
| `update-folder: update` | Legt den Ordner (im Plugin-Ordner) fest, in dem Updates für Plugins gespeichert werden. |
| `plugin-profiling: false` | Aktiviert (true) oder deaktiviert (false) den Befehl `/timings`. |
| `connection-throttle: 4000` | Zeit in Millisekunden, bevor ein Client sich nach einer Trennung wieder verbinden darf. |
| `query-plugins: true` | Aktiviert (true) oder deaktiviert (false) den Remote-Zugriff auf die Plugin-Liste. |
| `deprecated-verbose: default` | Aktiviert (true) oder deaktiviert (false) Warnhinweis bei Plugins, die veraltete Methoden verwenden. |
| `shutdown-message: Server closed` | Legt die Nachricht fest, die beim Schließen des Servers an die Spieler gesendet wird. |
| `monsters: 70` | Legt die Zahl der Monster fest, die in der Welt spawnen können. |
| `animals: 15` | Legt die Zahl der Tiere fest, die in der Welt spawnen können. |
| `water-animals: 5` | Legt die Zahl der Wassertiere fest, die in der Welt spawnen können. |
| `ambient: 15` | Legt die Zahl der »Ambient«-Kreaturen fest, die in der Welt spawnen können (zurzeit nur Fledermäuse). |
| `period-in-ticks: 600` | Legt fest, in welchen Abständen (in Ticks) geprüft wird, ob Chunks aus dem Speicher entfernt werden können. |
| `load-threshold: 0` | Zahl der Chunks, die geladen sein müssen, bevor versucht wird, ältere Chunks aus dem Speicher zu entfernen. |

**Tabelle 2.2:** Einstellungsmöglichkeiten `bukkit.yml`

| Einstellung | Erklärung |
|---|---|
| `animal-spawns: 400` | Legt fest, in welchen Abständen (in Ticks) der Server versucht, Tiere zu spawnen. |
| `monster-spawns: 1` | Legt fest, in welchen Abständen (in Ticks) der Server versucht, Monster zu spawnen. |
| `autosave: 6000` | Legt die Zahl von Ticks fest, nach denen die Inhalte des Servers gespeichert werden (6000 entspricht ca. alle 5 Minuten). |
| `aliases: now-in-commands.yml` | Gibt an, in welcher Datei alternative Namen für Befehle festgelegt werden. |
| `username: bukkit` | Legt den Nutzernamen für Datenbankzugriff fest. |
| `isolation: SERIALIZABLE` | Datenbankeinstellung, die nicht verändert werden sollte. |
| `driver: org.sqlite.JDBC` | Verwendeter Treiber für die Verbindung zur Datenbank. |
| `password: walrus` | Legt das Passwort für Datenbankzugriff fest. |
| `url: jdbc:sqlite:{DIR}{NAME}.db` | Adresse der Datenbank. |

**Tabelle 2.2:** Einstellungsmöglichkeiten `bukkit.yml` (Forts.)

### Die Datei spigot.yml

Wem diese fast 60 Einstellungsmöglichkeiten noch nicht kompliziert genug sind, der kann in der `spigot.yml` noch fast 100 weitere Einstellungen vornehmen, vorausgesetzt, man verwendet den Spigot-Server, denn nur der verfügt über diese Datei. Das sind so viele, dass an dieser Stelle nicht einzeln auf alle eingegangen werden kann. Zudem handelt es sich bei den meisten Optionen um Detaileinstellungen, die du vermutlich niemals benötigen wirst. Folgende sechs Einstellmöglichkeiten könnten aber durchaus interessant für dich sein: `whitelist`, `unknown-command`, `server-full`, `outdated-client`, `outdated-server` und `restart`. Mit diesen sechs Befehlen kannst du die Nachrichten festlegen, die an einen Spieler geschickt werden, wenn er sich nicht auf der Whitelist befindet, einen unbekannten Befehl eingibt, der Server voll ist, der Client des Spielers veraltet ist, der Server veraltet ist oder der Server neu gestartet wird. Mit eigenen Nachrichten kannst du deinem Server schnell und unkompliziert eine persönliche Note verleihen.

### Die Dateien banned-ips.json, banned-players.json, ops.json, whitelist.json

Die Dateien `banned-ips.json`, `banned-players.json`, `ops.json`, `whitelist.json` gibt es wieder unabhängig davon, welchen Server du verwendest. In ihnen wird gespei-

chert, welche IPs und Spieler vom Server verbannt wurden, welche Spieler Administratoren oder genauer ausgedrückt Operatoren sind und welche Spieler sich auf der Whitelist befinden.

Wie so eine Datei aussehen kann, zeigt Listing 2.2. Die dort dargestellte `whitelist.json` würde es nur einem Spieler, dem mit dem Namen »notch«, erlauben, auf dem Server zu spielen.

```
1 [
2   {
3     "uuid": "8d15678-a7f3-1234-8d11-c2ab1234dc9",
4     "name": "notch"
5   }
6 ]
```

**Listing 2.2:** Beispielinhalt `whitelist.json`

Alle vier Dateien sind nach diesem Prinzip aufgebaut und können theoretisch auch von Hand verwaltet werden, vorausgesetzt, du kennst die `uuid` des Spielers, also seine eindeutige Benutzeridentifizierung, den du zu einer Liste hinzufügen möchtest. Allerdings ist das überhaupt nicht notwendig, denn viel bequemer lassen sich all diese Listen durch die Eingabe von Befehlen im Server verwalten. Wie genau das funktioniert, darum soll es im nächsten Abschnitt gehen.

## 2.3 Befehle

Einige Befehle kennst du vermutlich schon aus dem Einzelspielermodus von Minecraft. Wenn du im Spiel mit der T-Taste den Chat öffnest, stehen dir verschiedene Befehle oder Cheats zur Verfügung, mit denen du die Welt beeinflussen kannst. Mit `/weather rain` kannst du es zum Beispiel regnen lassen, mit `/time set day` kannst du die Nacht zum Tag machen.

Alle Befehle, die du bereits aus dem Einzelspielermodus kennst, funktionieren auch auf deinem Server. Du kannst sie sogar direkt in dein geöffnetes Server-Fenster eingeben, dann allerdings ohne den Schrägstrich am Anfang, also zum Beispiel `weather rain` statt `/weather rain`. Wie das aussieht, kannst du in Abbildung 2.7 sehen.

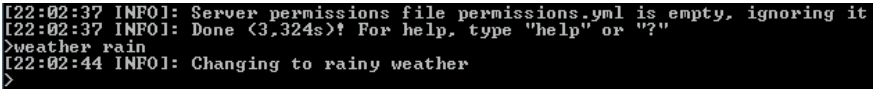


**Abbildung 2.7:** Befehlseingabe im Server-Fenster

Darüber hinaus stehen dir aber noch weitere Befehle zur Verfügung, die dir bei der Verwaltung deines Servers helfen. Mit dem Befehl `help` bekommst du eine Liste aller ver-

fügbaren Befehle angezeigt, die wichtigsten von ihnen findest du in alphabetischer Reihenfolge in Tabelle 2.3.

| Befehl | Beschreibung |
|---|---|
| `/ban <spielername>` | Verbannt einen Spieler dauerhaft vom Server. |
| `/ban-ip <ip>` | Verbannt eine IP-Adresse dauerhaft vom Server. |
| `/kick <spielername>` | Wirft einen Spieler temporär vom Server. |
| `/op <spielername>` | Gibt einem Spieler Administrationsrechte. |
| `/pardon <spielername>` | Hebt die Verbannung eines Spielers auf. |
| `/pardon-ip <ip>` | Hebt die Verbannung einer IP-Adresse auf. |
| `/restart` | Startet den Server neu. |
| `/say <nachricht>` | Sendet eine Nachricht an alle Spieler. |
| `/spawnpoint <x> <y> <z>` | Setzt den Spawnpunkt an die angegebene Stelle. |
| `/stop` | Schaltet den Server ab. |
| `/tell <spielername> <nachricht>` | Sendet eine private Nachricht an einen Spieler. |
| `/version` | Zeigt die Versionsnummer des Servers an. |
| `/whitelist on` | Erlaubt nur Spieler auf dem Server, die auf der Whitelist stehen. |
| `/whitelist off` | Erlaubt alle Spieler auf dem Server, die nicht verbannt sind. |
| `/whitelist add <spielername>` | Fügt der Whitelist einen Spieler hinzu. |
| `/whitelist remove <spielername>` | Entfernt einen Spieler von der Whitelist. |

**Tabelle 2.3:** Liste der Server-Befehle

**Merke**

Wenn du Befehle direkt ins Server-Fenster eingibst, muss der Schrägstrich am Anfang des Befehls weggelassen werden.

Spieler, die Operatoren sind, also mit op `<spielername>` zur Liste der Operatoren hinzugefügt wurden, können diese Befehle auch direkt im Spiel, wie gewohnt über den Chat, verwenden.

## 2.4 Verbinden

Inzwischen ist dein Server perfekt eingerichtet und konfiguriert, deshalb wird es jetzt langsam Zeit, ihn endlich einmal zu testen, indem du dich mit deinem Minecraft-Client darauf verbindest. Bevor du das machst, solltest du noch einmal überprüfen, dass du alle vorherigen Schritte ausgeführt hast und dein Server auch läuft.

**Merke**

Bevor du weiterliest, solltest du noch einmal überprüfen, ob du alle nötigen Installationsschritte ausgeführt hast:

1. Installation von Java
2. Herunterladen der Server-Datei von *buch.daniel-braun.com*
3. Neuen Ordner `server` anlegen und die Datei dorthin kopieren
4. Datei mit `java -jar` starten
5. Nutzungsbedingungen lesen und akzeptieren
6. Server erneut starten
7. Server-Fenster geöffnet lassen

Nachdem du das erledigt hast, kannst du Minecraft wie gewohnt starten. Im Hauptmenü wählst du dort dann den Eintrag MEHRSPIELER aus. Daraufhin öffnet sich das in Abbildung 2.8 gezeigte Menü.

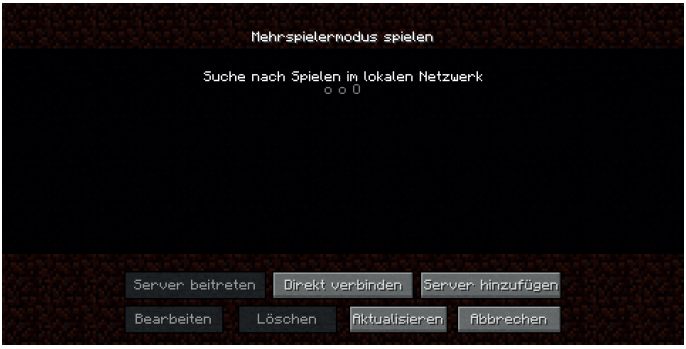


**Abbildung 2.8:** Mehrspieler-Menü

Dort klickst du nun auf den Button mit der Beschriftung DIREKT VERBINDEN, worauf sich die in Abbildung 2.9 gezeigte Maske öffnet.

Wenn du den Server auf demselben Computer laufen hast, auf dem auch der Client läuft, so muss als SERVERADRESSE immer `127.0.0.1` angegeben werden. Damit sagst du