

```
} catch(Exception e){  
    getLogman().error  
}  
return true;
```



mitp

```
@Override  
public boolean disable() {  
    HashMap<String, Object> filter = new HashMap<String, Object>();  
    try {  
        Database.get().removeAll(new HighscoreTabelle(), filter);  
    }  
    catch (DatabaseWriteException e) {  
        getLogman().error("Fehler bei der Daten");  
    }  
  
    String[] spielerListe = highscore.toArray(new  
        String[highscore.size()]);  
  
    for (int i = 0; i < spielerListe.length; i++) {  
        HighscoreTabelle eintrag = new HighscoreTabelle();  
        eintrag.nummer = i;  
        eintrag.spieler = spielerListe[i];  
        eintrag.punkte = highscore.get(spielerListe[i]);  
        try {  
            Database.get().insert(eintrag);  
        }  
        catch (DatabaseWriteException e) {  
            getLogman().error("Fehler bei der Daten");  
        }  
    }  
}
```

Daniel Braun

3. Auflage



LET'S PLAY

Programmieren lernen

mit **Java** und **Minecraft**

Für
**Bukkit &
Spigot**
unter Windows, Linux und macOS

**Plugins erstellen
ohne Vorkenntnisse**

Kein offizielles Minecraft-Produkt. Nicht von Mojang genehmigt oder mit Mojang verbunden.

Inhalt

Einleitung	11
 Kapitel 1 Java	15
1.1 Programmiersprachen	15
1.2 Besonderheiten von Java	17
1.3 Installation und Einrichtung	18
1.3.1 Java-Compiler installieren	20
1.3.2 Ordner einrichten	22
1.4 Editor	23
1.5 Zusammenfassung	24
 Kapitel 2 Minecraft-Server	25
2.1 Installation	26
2.1.1 CraftBukkit	26
2.1.2 Spigot	28
2.2 Konfiguration	30
2.3 Befehle	36
2.4 Verbinden	37
2.5 Updates	41
 Kapitel 3 Das erste Plugin	43
3.1 Programmieren	43
3.2 Kompilieren	46
3.2.1 Fehler finden	47
3.2.2 Jar-Datei erstellen	49
3.3 Starten	50
3.4 Entdecken	52
3.5 Rätsel	52
3.6 Zusammenfassung	53

Kapitel 4	Chat-Befehle	55
4.1	Eigene Befehle definieren	56
4.2	Chat-Nachrichten versenden	60
4.3	Rätsel	61
4.4	Zusammenfassung	61
Kapitel 5	Eclipse installieren und einrichten	63
5.1	Installation	63
5.2	Einrichtung	64
5.3	Ein neues Projekt anlegen	65
5.4	Neue Dateien in einem Projekt anlegen	67
5.4.1	Java-Datei	68
5.4.2	Info-Datei	70
5.5	Kompilieren und packen	70
Kapitel 6	Variablen und Konstanten	73
6.1	Variablen	73
6.1.1	Zahlen	74
6.1.2	Zeichenketten	77
6.1.3	Konvertierung	82
6.1.4	Arrays	93
6.2	Konstanten	95
6.3	Rätsel	96
6.4	Zusammenfassung	97
Kapitel 7	Schleifen	101
7.1	Kürbis-Plugin	101
7.1.1	Positionierung	102
7.1.2	Blöcke platzieren	104
7.2	Die verschiedenen Schleifen	107
7.2.1	for-Schleife	108
7.2.2	while-Schleife	112
7.2.3	do-while-Schleife	117
7.2.4	Verschachtelte Schleifen	118

7.3	Rätsel	122
7.4	Zusammenfassung	124

Kapitel 8 Verzweigungen 127

8.1	if-Verzweigung	127
8.2	case-Verzweigung	134
8.3	Rätsel	136
8.4	Zusammenfassung	137

Kapitel 9 Funktionen 139

9.1	Deklaration von Funktionen	139
9.2	Rückgabewerte	140
9.3	Parameter	141
9.4	Anwendungsbeispiel	142
9.5	Rätsel	146
9.6	Zusammenfassung	147

Kapitel 10 Klassen und Objekte 149

10.1	Die ganze Welt ist ein Objekt	149
10.2	Erstellung einer eigenen Klasse	152
10.3	Funktionen in Klassen	155
10.4	Zugriffskontrolle	163
10.5	Vererbung	165
10.6	Abstrakte Methoden und Klassen	170
10.7	Bau-Plugin	173
10.8	Rätsel	178
10.9	Zusammenfassung	178

Kapitel 11 Bauen 181

11.1	Notunterkunft	181
11.1.1	Wände und Decke	182
11.1.2	Tür	187
11.1.3	Bett	191
11.1.4	Fackel	194

11.2	Runde Objekte	198
11.2.1	Kreise	198
11.2.2	Kugeln	203
11.3	Zusammenfassung	206

Kapitel 12 Schilder 207

12.1	Hängende Schilder	207
12.2	Stehende Schilder	208
12.3	Text festlegen	210
12.3.1	Farbe	211
12.3.2	Formatierung	213
12.4	Schilder-Plugin (Listen)	214
12.4.1	Listen-Grundlagen	215
12.4.2	Das Plugin	218
12.5	Rätsel	234
12.6	Zusammenfassung	235

Kapitel 13 Listener 237

13.1	Grundgerüst	237
13.2	Spieler-Events	238
13.3	Kreaturen-Events	245
13.4	Block-Events	249
13.5	Inventar-Events	252
13.6	Server-Events	253
13.7	Fahrzeug-Events	254
13.8	Wetter-Events	255
13.9	Welt-Events	255
13.10	Mehrere Listener in einem Plugin	256
13.11	Zusammenfassung	258

Kapitel 14 Crafting-Rezepte 259

14.1	Rezepte festlegen	259
14.2	Eigene Rezepte entwerfen	262
14.3	Feuerschwert	263
14.4	Enderbogen	267

14.5	Rätsel	270
14.6	Zusammenfassung	270

Kapitel 15 Informationen dauerhaft speichern 273

15.1	Konfigurationsdateien	273
15.1.1	Lesen	273
15.1.2	Schreiben	276
15.2	Objekte in Dateien speichern	279
15.3	Zusammenfassung	294

Kapitel 16 Eigene Spielmodi entwickeln 297

16.1	Schneeballschlacht	297
16.1.1	Schneebälle verteilen	298
16.1.2	Schneebälle automatisch auffüllen	300
16.1.3	Punkte zählen	301
16.1.4	Highscore-Liste anzeigen	304
16.1.5	Vollständiger Quellcode	306
16.2	Sammelspiel	308
16.2.1	Aufbau des Plugins	308
16.2.2	Plugin starten	309
16.2.3	Spieler betritt den Server	311
16.2.4	Gegenstände zählen	312
16.2.5	Auftrag anzeigen	313
16.2.6	Vollständiger Quellcode	314
16.3	Rätsel	315
16.4	Zusammenfassung	316

Kapitel 17 Eigenständige Java-Programme 319

17.1	Grundgerüst	319
17.2	Statische Variablen und Funktionen	320
17.3	Ein- und Ausgabe	322
17.3.1	»Hallo Welt!«-Programm	322
17.3.2	Eingaben	323
17.4	Quiz programmieren	324

Anhang A	Rätsel-Lösungen	331
Anhang B	Befehlsreferenz	339
Anhang C	Materialien	357
Index	371

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, über meine Facebook-Seite www.facebook.com/AutorDanielBraun 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

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-Shopping 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.

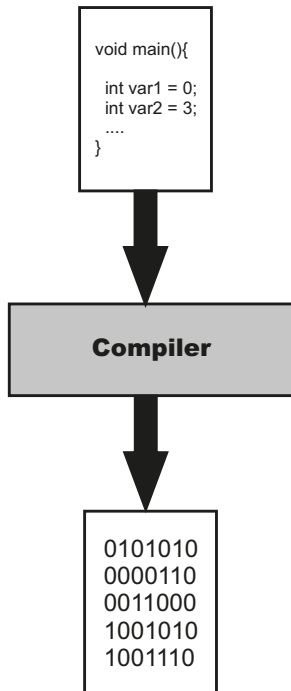


Abbildung 1.1: Funktionsweise eines Compilers

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.

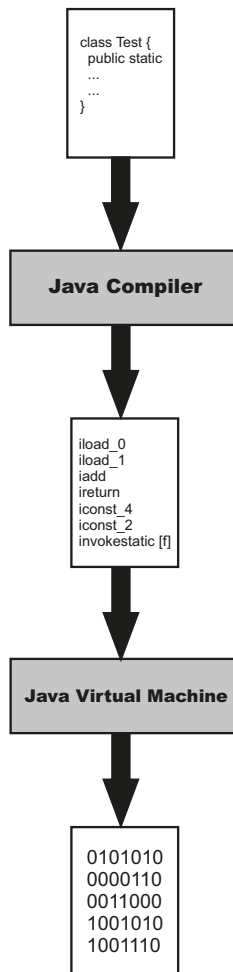


Abbildung 1.2: Funktionsweise des Java-Compilers

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 used
  -classpath <path>   Specify where to find user class files and annotations
  -cp <path>          Specify where to find user class files and annotations
  -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 compilation 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 parameters
  -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 files
  -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 profile
  -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 auf der Website von *Oracle* heruntergeladen werden. Einen Link zum Download findest du auf `buch.daniel-braun.com`.

Auf der Seite von Oracle musst du dann die für dein Betriebssystem passende Datei herunterladen. Die Dateien sind nach dem Schema benannt `jdk-$u$$$-betriebssystem-x64`, wobei die `$`-Zeichen der Versionsnummer entsprechen und `betriebssystem` dem Namen des Betriebssystems, das du verwendest. Nach dem Herunterladen kannst du die Datei einfach mit einem Doppelklick öffnen und der Installation bis zum Ende folgen.

Sobald diese abgeschlossen ist, solltest du die in Abschnitt 1.3 beschriebenen Schritte wiederholen, um zu testen, ob der Java-Compiler nun korrekt installiert ist. Solltest du Windows verwenden, kann es sein, dass der Befehl noch nicht funktioniert. Das bedeutet wahrscheinlich nicht, dass es einen Fehler bei der Installation gegeben hat, sondern lediglich, dass du noch eine sogenannte Umgebungsvariable einstellen musst.

Dazu musst du zunächst die erweiterten Systemeinstellungen deines Computers öffnen.

Windows XP, Vista und 7: Unter Windows XP, Vista und 7 öffnest du dafür zunächst das Startmenü und dann die SYSTEMSTEUERUNG. Dort wählst du aus der Kategorie SYSTEM UND SICHERHEIT den Eintrag SYSTEM aus und im sich danach öffnenden Fenster den Eintrag ERWEITERTE SYSTEMEINSTELLUNGEN.

Windows 8 und 10: Unter Windows 8 kannst du diese öffnen, indem du den Begriff einfach direkt in die Suche eingibst. Unter Windows 10 dagegen musst du 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:\Program Files\Java\jdk1.8.0_121\bin;
```

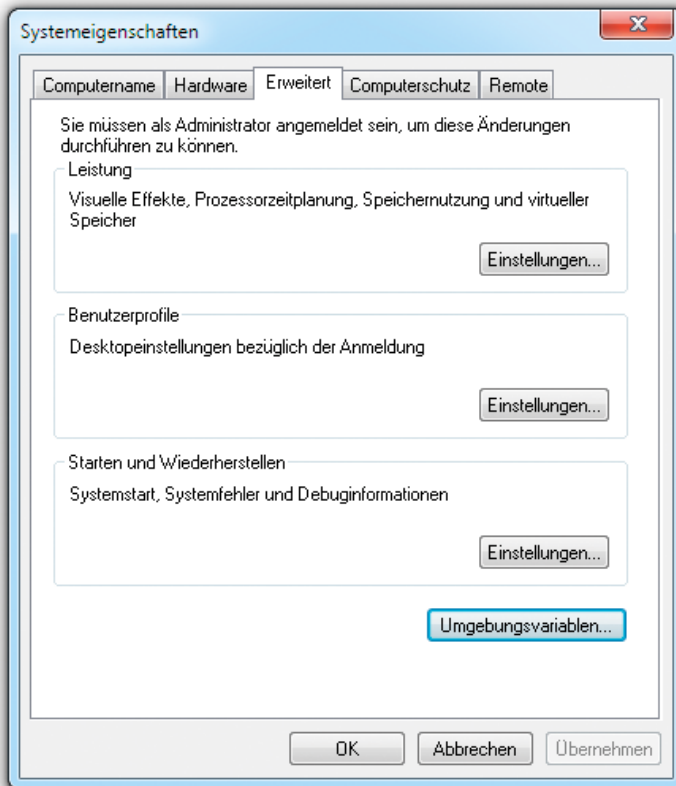



Abbildung 1.4: Erweiterte Systemeinstellungen

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.

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.

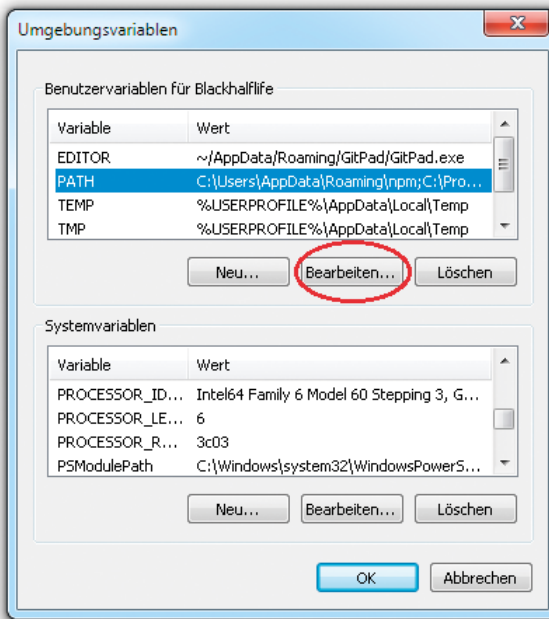


Abbildung 1.5: Umgebungsvariablen

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 `pPlugins` 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.

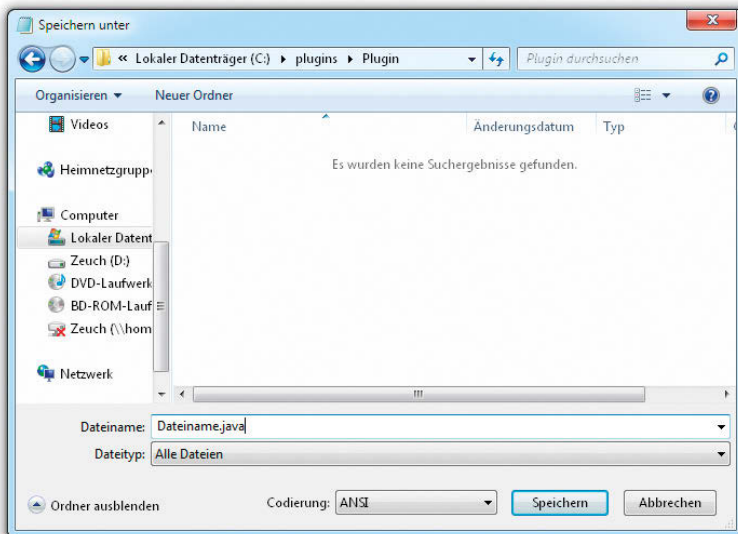


Abbildung 1.6: Speichern von Dateien mit dem Windows-Editor

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 **IDEs**. 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.

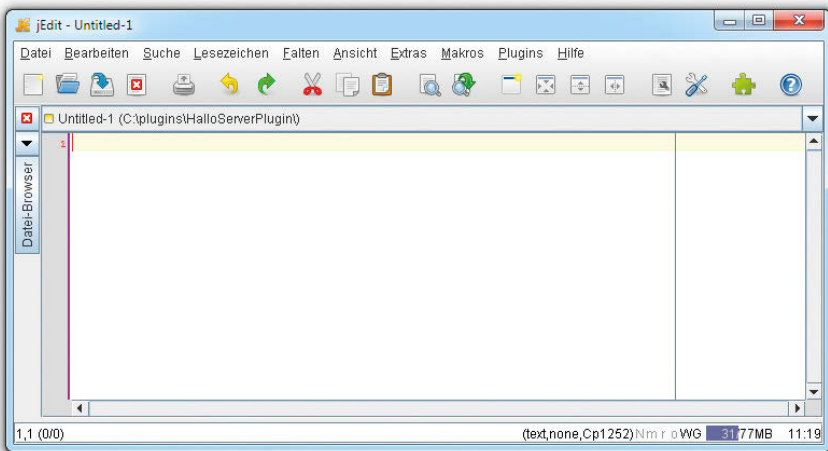


Abbildung 1.7: jEdit

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 Maschinsprache übersetzt.
Binärdatei	Eine Datei, die Maschinsprache enthält, die nur aus Nullen und Einsen besteht.

Index

Symbole

`^` 116
`!` 116
`!=` 108
`@Override` 172
`*/` 109
`*` 89
`/*` 109
`//` 109
`/=` 89
`&&` 114
`%=` 89
`++` 89
`+=` 89
`<` 108
`<=` 108
`-=` 89
`==` 108
`>` 108
`>=` 108
`||` 115

A

Abrunden 86
abstract 170, 171
abstrakte Klasse 170
abstrakte Methode 170
Abstraktion 150
add 215
addUnsafeEnchantment 264
Algorithmus 15
Alias 186
Angel 260
Array 93, 342
ArrayList 215
Attribut 151
Aufrunden 86
Ausrichtung 209

B

Befehl 36
Bett 191
Bezeichner 74
Binärdatei 16
Binnenmajuskel siehe CamelCase
Block 348
 Status 187
BlockFace 187
Bogen 267
BOLD 214
Boolean 113
break 136, 222
Bukkit siehe CraftBukkit
bukkit.yml 33
byte 75
Bytecode 17

C

CamelCase 74
case 135
case-Verzweigung 134
cd 26
Chat 55, 353
Chat-Befehl 55
Chat-Kommando siehe Chat-Befehl
Cheat siehe Befehl
class 151
clear 217
Client 25
commands 56
Compiler 16
config.yml 274
contains 217, 299
CraftBukkit 25
Crafting 259, 350

D

Daten-Ordner 277
 Deklaration 74
 Differenz 77
 Division 77
 Rest 77
 double 76
 do-while-Schleife 117

E

Eclipse 63
 Workspace 64
 else 130
 Enderbogen 267
 Enderperle 267
 Endlosschleife 109
 Entweder-Oder-Operator 116
 equals 136
 Ereignis siehe Event
 EVA-Prinzip 73
 Event 238
 PlayerJoinEvent 238
 Priorität 240
 setCancelled 266
 exists 288

F

Fackel 194
 faires Runden 86
 Fallunterscheidung 134
 Feld siehe Array
 first 299
 Fließkommazahl 75
 float 76
 Formatierung 213, 214
 for-Schleife 108
 Funktion 139, 345
 Signatur 170
 überschreiben 172
 Fußgesteuert 117

G

Gegenstand 349
 Geschweifte Klammer 45

get 217
 getData 187
 getDataFolder 277
 getShooter 269
 getTargetBlock 221
 getter-Methode 164
 getUUID 234
 getYaw 223
 Gleitkommazahl siehe Fließkommazahl

H

Handlungsvorschrift siehe Algorithmus
 HashMap 301

I

IDE siehe Integrierte Entwicklungsumgebung
 if-Verzweigung 127
 Implementierung 172
 implements 172
 import 44
 indexOf 217
 Info-Datei 49
 Initialisierung 93
 instanceof 128
 Instanzen 150
 int siehe Integer
 Integer 75
 Integrierte Entwicklungsumgebung 24, 63
 Interface 172
 Inventar 299, 349
 Slot 299
 IP-Adresse 39
 ITALIC 214
 ItemStack 259

J

Jar-Datei 49
 Java 17
 Java Development Kit 20
 Java Virtual Machine 17
 Java-Archive 49
 Java-Bytecode 17
 javac 19

JDK siehe Java Development Kit
JVM siehe Java Virtual Machine

K

Kaufmännisches Runden 86
Klasse 150, 344
 abstrakt 170
Kommentar 109
Konfigurationsdatei 273, 354
Konkatenationsoperator 78
Konstante 95
Konstruktor 151
Konvertierung 83
Koordinate 103
Kopfgesteuert 117
Kreis 198
Kugel 203

L

Laufbedingung 108
length 220
LinkedList 215
Liste 215, 342
 abrufen 217
 ändern 217
 ArrayList 215
 Länge 216
 lesen 217
 LinkedList 215
 Löschen 216
 Position 217
Listener 237, 353
Location siehe Position
Log-Datei 30
logischer Operator 346
Lokale Variable 157
long 75

M

Map
 TreeMap 304
Maschinensprache 16
Math.pow siehe Potenz
Meta-Daten 263

Methode 160
 abstrakt 170
 überschreiben 172
Methode von Horn 199
Methode von Metzger 199
Midpoint-Algorithmus 199
mkdirs 288
Mod 25
Modulo siehe Restwert

N

nextInt 310
Nicht-Operator 116
Notunterkunft 181

O

Oberklasse 166
Objekt 150, 344
Objektorientierte Programmierung 149
Oder-Operator 115
onDisable 288
OOP siehe Objektorientierte Programmierung

P

Parameter 57, 141
plugin.yml 49
PluginManager 238
Position 102, 347
Potenz 206
private 163
Produkt 77
Programmcode siehe Quellcode
Programmiersprache 15
 C 17
 Java 17
 Python 17
ProjectileHitEvent 268
protected 174
public 163

Q

Quellcode 44
Quotient 77

R

- Radius 198
- Rasterung 198
- Rechnen 351
- remove 216
- Restwert 77
- return 141
- Rezept 259, 350
 - ShapedRecipe 260
 - ShapelessRecipe 259
- Rückgabewert 140
- Runden
 - abrunden 86
 - aufrunden 86
 - fares 86
 - kaufmännisch 86
- runden 85

S

- saveResource 277
- Schild 207
 - Ausrichtung 209
 - hängend 207
 - stehend 208
 - Text 210
- Schleife 107, 339
 - do-while 117
 - endlose 109
 - for 108
 - fußgesteuert 117
 - Kopf 108
 - kopfgesteuert 117
 - Rumpf 109
 - Variable 108
 - while 112
- Schleifenkopf 108
- Schleifenrumpf 109
- Schleifenvariable 108
- Schneeballschlacht 297
- Schnittstelle siehe Interface
- Schriftfarbe 211
- Semikolon 44
- Serialisierung 284
- Serializable 284
- Server 25

- server.properties 31
- set 217
- setAmount 299
- setCancelled 266
- setDisplayName 263
- setHalf 187
- setLine 211
- setOpen 187
- setter-Methode 164
- shape 260
- short 75
- Signatur 170
- size 216
- Slot 299
- Sourcecode siehe Quellcode
- Speichern 273
- Spieler 346
- Spigot 25
- spigot.yml 35
- Startwert 108
- STRIKETHROUGH 214
- String 77
- Summe 77
- super 167
- switch 135
- switch-case-Verzweigung siehe case-Verzweigung
- Syntax 108

T

- teleport 270
- Textformatierung 213
- TreeMap 304
- Tür 187

U

- Überschreiben 172
- Umgebungsvariable 20
- UML-Diagramm 168
- UNDERLINED 214
- Und-Operator 114
- Unterklasse 167
- usage 56
- UUID 234

V

- Vanilla-Server 25
- Variable 73, 340
 - Deklaration 74
 - Initialisierung 93
 - lokal 157
 - Name 74
 - Typ 75
 - Wertzuweisungen 76
- Vererbung 166
- Vergleichsoperator 108
- Verzauberung 264
- Verzweigung 127, 340
 - case 134
 - else 130
 - if 127
- void 140

W

- Wahrheitstafel 114
- Wahrheitswert 113
- Welt 348

- Wertebereich 75
- Wertzuweisungen 76
- while-Schleife 112
- writeObject 288

X

- XOR-Operator 116

Z

- Zeichenkette siehe String
- Zufallsgenerator 310