Java-Tutorial

– Programmieren lernen –

Karsten Brodmann

www.punkt-akademie.de

(Folge 4 - 22.12.2022)

Grundlagen der Java-Programmerstellung und Ausführung

Wie Sie mit den wichtigsten Werkzeugen des JDK manuell arbeiten, und was es dabei zu beachten gilt, soll im Folgenden an einem ersten Beispielprogramm erörtert und gezeigt werden.

Das erste Programm in einer neuen Programmiersprache ist stets das sogenannte *HelloWorld*-Programm. Dieses Programm ist seit etwas Anfang der 70er-Jahre des letzten Jahrhunderts Tradition. Es gibt einfach die Zeichenkette „Hello, World!“ auf der Standardausgabe aus. Die Standardausgabe ist, wenn nicht umgeleitet, mit dem Terminalfenster verbunden, welches bei einem deutschen Microsoft Windows etwas irritierend als Eingabeaufforderung bezeichnet wird.

Programmcode erfassen

Für unser erstes Programm verwenden Sie bitte noch keine IDE. Also nutzen Sie nicht IntelliJ, Eclipse oder ähnliche Software. Verwenden Sie einen einfachen Texteditor, um den folgenden Programmcode zu erfassen.

Die Zeilennummern sind nicht abzuschreiben. Sie dienen lediglich dazu, sich in Erläuterungen leichter auf einen bestimmten Abschnitt im Code beziehen zu können. Beachten Sie beim Abschreiben die Groß-/und Kleinschreibung.

```
1 import de.pakad.tools.StdOut;
2
3 /** Das erste Java-Programm.
4  *
5  * @author Karsten Brodmann
6  */
```

```
7  public class HelloWorld {
8      /**
9       * @param args nicht verwendet
10      */
11     public static void main(String[] args) {
12         StdOut.println("Hello, World!");
13     }
14 }
```

Quellcode 1: HelloWorld.java

In Java müssen wir immer eine sogenannte *Klasse* definieren. Das ist der Tribut an die Objektorientierung von Java. Für den Anfang werden wir diese, soweit es möglich ist, außer Acht lassen. Daher werde ich auch erst später erläutern, wie eine Klassendefinition aussieht, welche Optionen es gibt. Für den Anfang haben alle Klassen, die wir implementieren werden, den gezeigten Aufbau. Wir beginnen mit dem *Klassenkopf*. Der startet mit dem *Schlüsselwort* `public`, gefolgt von `class`, gefolgt vom Namen unserer Klasse. Das ist hier `HelloWorld`. Danach kommt ein Paar geschweifter Klammern (Zeile 1 und 5). Sie umschließen den *Klassenrumpf*.

Schlüsselworte bilden das Vokabular einer Programmsprache. Sie sind in den abgedruckten Programmcodes fett gedruckt und farbig hervorgehoben. Die Bezeichner der Schlüsselworte sind reserviert, weshalb man auch von *reservierten Worten* spricht. Ihre Bezeichner dürfen nur für die von Java vorgesehenen Zwecke verwendet werden. Das heißt, Sie dürfen beispielsweise keinen Klassennamen oder anderen Bezeichner vergeben, der mit dem Bezeichner eines Schlüsselwortes identisch ist.

Bezeichner bestehen in Java immer aus Buchstaben, dem Dollarzeichen, Unterstrichen oder Ziffern. Das erste Zeichen eines Bezeichners darf keine Ziffer sein. Groß- und Kleinschreibung ist signifikant. Von der Verwendung des Dollarzeichens ist abzuraten.

Eigentlich gibt es gar keine richtigen Java-Programme. Es gibt lediglich ausführbare Klassen und Klassen, die es nicht sind. Eine ausführbare Klasse muss eine *Methode* names `main` implementieren. Wenn Sie Java befehlen, eine Klasse auszuführen, sucht die JVM in der benannten Klasse nach der `main`-Methode. Sie bildet den Einsprungspunkt für die Programmausführung. Wird `main` gefunden, werden die darin notierten Anweisungen in der Reihenfolge ihrer Notation ausgeführt.

Die Methode `main` hat immer den hier gezeigten Aufbau. Es gibt keine Variationen. Der *Methodenkopf* besteht aus den *Modifizierern* `public` und `static`. Danach folgt der Typ des Rückgabewertes. `main` besitzt keinen Rückgabewert. Daher wird `void` geschrieben. Dann kommt die *Signatur* der Methode. Sie besteht aus dem Methodennamen und einem runden Klammerpaar. In den Klammern können keine oder aber auch mehrere *Parameter* notiert werden, die einer Methode zur Verarbeitung und/oder Steuerung übergeben werden. Mehrere Parameterangaben werden mit Kommata getrennt. Bei `main` gibt es genau einen Parameter. Das ist `String[] args`. `String[]` gibt an, dass es sich bei

dem Parameter namens `args` um ein *Array* von Zeichenketten handelt. `String` bedeutet Zeichenkette. Nach dem Methodenkopf folgt dann der *Methodenrumpf*. Der ist, analog zum Klassenrumpf, in geschweiften Klammern zu notieren. – Die Bedeutung der genannten Komponenten erfolgt später. Im Moment ist das „syntaktischer Zucker“. Sie kennen jetzt aber schon die relevanten Begriffe.

Im Methodenrumpf von `main` stehen die auszuführenden Anweisungen. Hier, in Zeile 3, ist das nur eine einzige Anweisung. Sie gibt an, die Zeichenkette „Hello, World!“ auf der Standardausgabe auszugeben. *Zeichenkettenliterale* werden in Java paarweise in doppelten Anführungszeichen notiert. Jede Anweisung ist in Java mit einem Semikolon abzuschließen.

Speichern Sie die Datei unter dem Namen `HelloWorld.java` in einem beliebigen Verzeichnis Ihrer Festplatte ab. Achten Sie auch hierbei auf eine korrekte Schreibweise. Klassenname und Dateiname müssen miteinander korrespondieren.

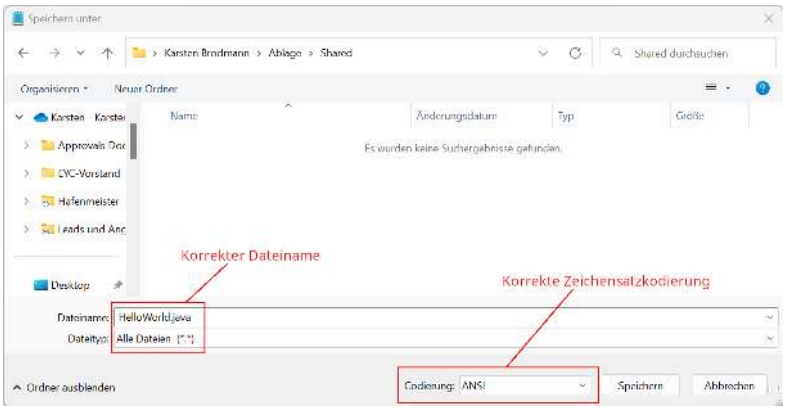

Abbildung 1: HelloWorld.java speichern

Achtung: Achten Sie beim Speichern auf die richtige Zeichensatzkodierung. Unter Microsoft Windows wählen „ANSI“. Manchmal heißt das, je nach Editor, auch „CP1252“ oder ähnlich. Verwenden Sie macOS oder Gnu/Linux als Betriebssystem, dann wählen Sie „UTF-8“. - Auf die Hintergründe gehe ich in Abschnitt „Stolperstein Zeichensatzkodierung“, Seite 4, ein.

Programmcode übersetzen

Öffnen Sie nun ein Terminalfenster und begeben sich in das Verzeichnis, in welchem Sie `HelloWorld.java` abgelegt haben. Dort starten Sie den Java-Compiler.

```
$ javac HelloWorld.java
$ _
```

Wenn Ihr Programmcode syntaktisch fehlerfrei war, gibt der Java-Compiler keine Meldung aus. Er hat dann, ohne Murren, eine Datei namens `HelloWorld.class` erzeugt. Sie enthält den *Bytecode*, den die JVM ausführen kann.

Ist Ihr Programmcode syntaktisch fehlerhaft, dann wird der Java-Compiler das mitteilen.

```
$ javac HelloWorld.java
HelloWorld.java:3: error: ';' expected
        System.out.println("Hello, World!")
                                           ^
1 error
$ _
```

Hier habe ich, um Ihnen dies zu demonstrieren, das Semikolon hinter der Ausgabeanweisung weggelassen.

Im Fehlerfall erzeugt der Compiler keine (neue) `class`-Datei! - Wechseln Sie dann wieder zurück in Ihren Editor und korrigieren den Fehler. Irgendwann, nach endlich langer Zeit, wird der Compiler Ihren Programmcode akzeptieren.

Programm ausführen

Zur Ausführung eines Java-Programms, einer ausführbaren Klasse, wird die JVM aufgerufen und ihr der Name der auszuführenden Klasse übergeben.

```
$ java HelloWorld
Hello, World!
$ _
```

Die `class`-Datei wird anhand des Klassennamens identifiziert und zur Ausführung geladen.

Sie können die `class`-Datei auf jedem Betriebssystem ausführen, auf welchem eine entsprechende JVM installiert ist.

Stolperstein Zeichensatzkodierung

Betrachten wir nochmals die Zeichensatzkodierung, die wir beim Speichern der Quellcodedatei eines Programms verwenden. Dazu ändern wir die Zeichenkette in der Ausgabeanweisung auf `"Hällo, World!"`, verwenden also einen deutschen Umlaut.

Zum Verständnis der Problematik sollten sie das Folgende Wissen:

- Microsoft Windows verwendet bei deutschen Installationen auf der grafischen Benutzeroberfläche die Zeichensatzcodierung CP1252.

- In der Eingabeaufforderung wird bei einer deutschen Windows-Installation die Codepage 850 als Kodierung verwendet.
- Gnu/Linux und macOS nutzen sowohl auf deren grafischen Benutzeroberflächen als auch im Terminalfenster die Zeichensatzkodierung UTF-8.

CP1252 und die Codepage 850 verwenden zur Kodierung eines Zeichens ein Byte. UTF-8 nutzt ein bis vier Byte, um ein Zeichen zu kodieren. Deutsche Umlaute benötigen hierbei zwei Byte. Aber obwohl CP1252 und Codepage 850 jeweils ein Byte verwenden, sind die Zeichen in diesen Zeichensätzen anders angeordnet. In der Codepage 850 ist das Zeichen ä mit der dezimalen Zahl 132 (`10000100`) kodiert. Bei CP1252 hat das Zeichen die Nummer 228 (`11100100`). Je nach Zeichensatzkodierung unterscheidet sich also die Kodierung dieses Zeichens.

In einer Textdatei sind nicht wirklich Zeichen gespeichert. Sie enthält die binären Kodierungen, die uns als die bekannten Zeichen, hier Buchstaben, angezeigt werden. Zum Test lassen Sie sich unter Microsoft Windows den modifizierten Quellcode unseres Programm mittels des `type`-Kommandos ausgeben.

```
$ type HelloWorld.java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hõllo, World!");
    }
}
$ _
```

Sie erkennen die fehlerhafte Darstellung des Buchstabens ä. Bei der Ausführung der `class`-Datei tritt dieser Fehler nicht auf, wenn Sie die korrekte Zeichensatzkodierung beim Speichern gewählt haben.

```
$ java HelloWorld
Hällo, World!
$ _
```

Java kümmert sich automatisch um die korrekte Zeichendarstellung. Dabei schaut Java nach, in welcher Umgebung eine Klasse ausgeführt wird. Entsprechend dieser Umgebung werden die Zeichen korrekt dargestellt. Damit dies funktioniert, ist es jedoch wichtig, ein Programm korrekt zu übersetzen.

Unter Microsoft Windows geht der Java-Compiler davon aus, dass die ihm zur Übersetzung gegebenen Quellcodes mit CP1252 kodiert sind. Bei Gnu/Linux und macOS ist das UTF-8. Genau genommen, schaut der Compiler nach, welche Zeichensatzcodierung in der Umgebung, in welcher er ausgeführt wird, gilt. Bei Microsoft Windows ist das, was ein wenig verwirren mag, die Zeichensatzkodierung der grafischen Benutzeroberfläche.

Nehmen wir an, wir hätten den Quellcode unseres Programms unter Microsoft Windows mit UTF-8 gespeichert. Dann kommt es zu fehlerhaften Ausgaben von Umlauten. Den Grund habe ich gerade erläutert.

```
$ java HelloWorld
HÃ¤llo, World!
$ _
```

Compilieren Sie einen Programmcode unter Gnu/Linux oder macOS, der nicht mit der zeichensatzcodierung UTF-8 kodiert ist, ergeben sich ganz analoge Fehler.

Wissen wir, mit welcher Kodierung ein Quelltext abgespeichert ist, können wir den Java-Compiler anweisen, die korrekte Zeichensatzkodierung bei der Übersetzung zu verwenden. Ein mit UTF-8 kodierter Programmtext wird wie folgt übersetzt.

```
$ javac -encoding utf8 HelloWorld.java
$ java HelloWorld
Hällo, World!
$ _
```

Bei abweichenden Zeichensatzcodierungen von Quelldatei und Ausführungsumgebung, können wir den Compiler anweisen, die richtige Kodierung zu nutzen.

Unter Gnu/Linux oder macOS, deren Standard-Zeichensatzcodierung UTF-8 ist, würde ein mit CP1252 kodierter Quellcode wie folgt übersetzt werden.

```
$ javac HelloWorld.java
...:3: error: unmappable character (0xE4) for encoding UTF-8
        System.out.println("H�llo, World!");
                           ^
1 error
$ javac -encoding cp1252 HelloWorld.java
$ java HelloWorld
Hällo, World!
$ _
```

Für den Druck habe ich die Ausgabe der Fehlermeldung leicht gekürzt. Entscheidend ist, dass der mit CP1252 kodierte Umlaut, das Zeichen `ä`, in UTF-8 nicht einmal definiert ist. Die Bitfolge ist in UTF-8 unbekannt. Das Zeichen ist mit zwei Byte kodiert. Der Compiler kann eine solche Datei daher gar nicht erst übersetzen. Die Angabe der korrekten Kodierung hilft jedoch, wie Sie sehen.

Ist ein Programm korrekt übersetzt, sorgt die JVM dafür alle Zeichen korrekt, und an die Ausführungsumgebung angepasst, anzuzeigen. Jedes korrekt übersetzte Programm ist deshalb plattformunabhängig ausführbar.

Wichtig: Auch wenn Sie vielleicht im weiteren Verlauf des Tutorials eine IDE, wie beispielsweise IntelliJ, einsetzen, achten Sie unbedingt darauf, diese entsprechend Ihrer Betriebssystemumgebung korrekt einzustellen. Bei den meisten Programmen ist das als Standard-Konfiguration bereits richtig. Verwenden Sie jedoch Microsoft Visual Studio Code, dann ist dessen Vorgabe (auch unter Microsoft Windows) UTF-8. In diesem Fall müssen Sie entweder beim Speichern Ihrer Dateien die passende Kodierung angeben oder den Java-Compiler entsprechend anweisen.