Java-Tutorial

– Programmieren lernen –

Karsten Brodmann

www.punkt-akademie.de

(Folge 7 - 27.12.2022)

Implementierung des Collatz-Algorithmus

Sie erinnern sich an Folge 3, in welcher ich Ihnen den Collatz-Algorithmus vorgestellt habe. Der Collatz-Algorithmus geht zurück auf Lothar Collatz, einen deutschen Mathematiker. In den 30er-Jahren entdeckte Collatz eine einfach zu bildende Zahlenfolge, von der er vermutete, dass sie stets in einen Zyklus der Folge 4, 2, 1, 4, ... mündet. Das Bildungsgesetz lautet:

- Nimm eine positive ganze Zahl.
- Ist diese Zahl gerade, halbe sie, andernfalls multipliziere sie mit 3 und addiere 1.
- Wiederhole den vorigen Schritt mit der jetzt aktuellen Zahl.

Auf diese einfache Art und Weise baut sich eine Collatz-Folge auf. Die geäußerte Vermutung konnte bis heute nicht bewiesen werden. Sie gehört zu den ungelösten Problemen der Mathematik. Und auch wenn es keinen Beweis darstellt, ist es ebenso noch niemandem gelungen, eine Startzahl zu finden, für welche die Vermutung nicht zutrifft.

Um den Algorithmus in Java zu implementieren, gönnen wir uns zuerst den schon bekannten „syntaktischen Zucker“. Das heißt, wir spendieren uns eine Klasse, die wir hier `Collatz` nennen wollen, und eine `main`-Methode, weil die Klasse ausführbar sein soll. Dieses Grundgerüst speichern wir in einer Datei namens `Collatz.java`.

Wir wollen eine positive ganze Zahl einlesen und wir benötigen einen Zähler. Das sind zwei Werte, die sich im Programmablauf ändern werden. Wir müssen sie speichern. Dazu verwendet man *Variablen*. Wie der Name andeutet, sind sie variabel. Sie können ihre Werte ändern. Technisch betrachtet sind Variablen Bezeichner für ein Stück Hauptspeicher, der durch die Variablen referenziert wird, sodass wir darauf zugreifen können, um Werte dort hinein schreiben oder daraus lesen können.

In Java müssen Variablen mit einem Datentypen vereinbart werden. Der *Datentyp* legt den Wertebereich der in einer Variablen zu speichernden Daten fest, bestimmt die auf den Daten zulässigen Operationen und definiert, wie Literale zu notieren sind.

Unsere Variablen, wir nennen sie einfach `x` und `z`, sollen ganze Zahlen sein. Der in Java am häufigsten genutzte Datentyp für ganze Zahlen, ist der Datentyp `int`.

In den Zeilen 21 und 22 erklären wir diese Daten und initialisieren sie auch gleich. `x` bekommt den Wert einer interaktiven Eingabe, die wir mithilfe der Methode `readInt` aus den PATools realisieren. `z` bekommt den Wert 0 zugewiesen. – Noch wurde nichts gezählt. Das einfache Gleichheitszeichen bedeutet in Java: Weise den rechts stehenden Ausdruck der Variablen auf der linken Seite zu!

```
1  import de.pakad.tools.StdIn;
2  import de.pakad.tools.StdOut;
3
4  /**
5   * Berechnung der Collatz-Funktion und der Anzahl
6   * der Iterationen, die benötigt werden, um eine
7   * Eingabezahl auf 1 zu transformieren.
8   *
9   * @author Karsten Brodmann
10  */
11 public class Collatz {
12     /**
13      * @param args nicht verwendet
14      */
15     public static void main(String[] args) {
16         /* Definiere zwei Variablen x und z vom Typ "ganze Zahl"
17
18             x = Eingabe-/Startzahl
19             z = Zähler für die Anzahl der Transformationen
20          */
21         int x = StdIn.readInt("Bitte eine positive Ganzzahl: ");
22         int z = 0;
23
24         while (x > 1) {         // solange x größer als 1 ...
25             if (x%2 == 0)       // wenn x gerade,
26                 x = x/2;        // dann halbier x,
27             else                // sonst
28                 x = 3*x + 1;    // verdreifach x und addiere 1
29             z = z+1;            // erhöhe z um 1
30         }
31         StdOut.println(z);      // gib z aus
32     }
33 }
```

Quellcode 1: Collatz.java

Die eigentliche Vorschrift zur Bildung einer COLLATZ-Folge ist in den Zeilen 25 bis 28 implementiert. Mittels der `if`-Anweisung treffen wir eine Fallunterscheidung. Wir prüfen, ob `x` gerade ist. Das ist dann der Fall, wenn der aktuelle Wert von `x` ganzzahlig ohne Rest durch 2 teilbar ist. Dazu nutzen wir den Modulo-Operator, das `%`-Zeichen. Ist der Rest der ganzzahligen Division 0, dann ist `x` gerade. Den Vergleich auf Gleichheit mit dem Wert 0, formulieren wir mit dem Vergleichsoperator ==. `x%2 == 0` ist eine Bedingung. Eine Bedingung wird in runden Klammern notiert.

Ist die Bedingung wahr, `x` also gerade, dann wird die Anweisung in Zeile 26 ausgeführt, andernfalls die alternative Anweisung in Zeile 28. Die Alternative wird durch das Schlüsselwort `else` gekennzeichnet. Sie wird ausgeführt, wenn die `if`-Bedingung falsch ist. Weil in den Zeilen 26 und 28 jeweils nur eine Anweisung steht, brauchen wir keine geschweiften Klammern setzen. Wären es mehrere abhängige Anweisungen, müssten wir sie syntaktisch zu je einem Anweisungsblock zusammenfassen. Anweisungsblöcke stehen in geschweiften Klammern und bilden, obgleich sie mehrere Anweisungen beinhalten, syntaktisch eine Anweisung.

Aus Sicht der Schulmathematik sehen die Anweisungen in den Zeilen 26 und 28 seltsam aus. Sie drücken jedoch keine Gleichungen aus. Es handelt sich um Wertzuweisungen. Zeile 26 ist wie folgt zu lesen: Berechne den Term `x` ganzzahlig geteilt durch 2. Danach weise das Ergebnis der Variablen `x` auf der linken Seite des Zuweisungsoperators zu. Der alte Wert von `x` wird dabei überschrieben. Analog verhält es sich mit Zeile 28.

In Zeile 29 erhöhen wir auf die gleiche Art und Weise unsere Zählvariable `z` um 1.

Das Bildungsgesetz der COLLATZ-Folge verlangt, die Berechnung, die wir gerade vorgenommen haben, wiederholt auszuführen. Daher kleiden wir sie in eine Schleife. Im Schleifenrumpf notierte Anweisungen werden solange wiederholt ausgeführt, wie die Schleifenbedingung wahr ist.

Java kennt verschiedene Schleifenkonstrukte. Hier haben wir eine `while`-Schleife. Sie prüft, ob unser `x` einen Wert größer als 1 besitzt. Wenn ja, werden die eben besprochenen Anweisungen im Schleifenrumpf ausgeführt, der durch ein Paar geschweifter Klammern ausgedrückt wird. Sollte unser `x` bereits vor der Schleife einen Wert kleiner gleich 1 besitzen, wird der Schleifenrumpf nicht betreten. Das Programm würde dann direkt hinter der Schleife, in Zeile 31, fortgesetzt.

Haben wir jedoch einen Wert für `x` eingegeben, der die Schleifenbedingung `x > 1` erfüllt, tritt das Programm in die `while`-Schleife ein. Die darin notierten Anweisungen werden ausgeführt. Wenn diese Anweisungen nun abgearbeitet sind, prüft die `while`-Schleife, ob das `x` immer noch die Schleifenbedingung erfüllt. Wenn ja, werden die Anweisungen in der Schleife wiederholt ausgeführt. Ergibt die Bedingungsprüfung falsch, wird die Schleife beendet und das Programm wird hinter der Schleife fortgesetzt.

Hinter der `while`-Schleife steht nur noch eine Anweisung, die Ausgabe des Wertes von `z`, der Zählvariablen.

Beachten Sie, dass wir die Schleifenbedingung so formuliert haben, die in der Schleife notierten Anweisungen nicht unendlich oft auszuführen. Unsere Intention ist es, das Programm zu beenden, wenn wir den Wert 1 erreicht haben! – Weil die |Collatz|-Vermutung nie bewiesen wurde, ist dies ein „gewagtes Programm“. Grundsätzlich können wir nicht sagen, ob es terminiert. Sollte es eine Zahl geben, die durch die COLLATZ-Folge nicht zu 1 transformiert wird, könnte unser Programm potentiell endlos laufen. Bezüglich des Wertebereichs von `int` wissen wir aber zuverlässig, das es darin keine solche Zahl gibt.

Beispielhafte Programmausführungen:

```
$ java Collatz
Bitte eine positive Ganzzahl: 3
7
$ java Collatz
Bitte eine positive Ganzzahl: 1
0
$ java Collatz
Bitte eine positive Ganzzahl: 7
16
$ _
```

Kommentare

Sie sehen im Programmcode verschiedene Erläuterungen, die den Programmcode kommentieren. Man nennt sie dementsprechend *Kommentare*.

Kommentare dienen dem menschlichen Leser dazu, einen Programmcode leichter zu verstehen. Dem Java-Compiler sind sie egal. Kommentare haben keinen Einfluss auf die Generierung des Bytecodes und damit auch nicht auf das Laufzeitverhalten eines Programms.

Java kennt drei Arten von Kommentaren:

Einziliger Kommentar Der einzeilige Kommentar beginnt mit der Zeichenfolge `//` und reicht bis an das jeweilige Zeilenende.

Mehrzeiliger Kommentar Der mehrzeilige Kommentar wird in `/*` und `*/` eingeschlossen.

Dokumentationskommentar Dokumentationskommentare können ebenfalls mehrzeilig sein. Sie beginnen mit `/**` und enden mit `*/`. Sie werden vom Java-Dokumentationswerkzeug `javadoc` ausgewertet. Damit können sie automatisiert HTML-Dokumentationen Ihrer Programme erstellen. Weil es sich um eine HTML-Dokumentation handelt, die erstellt wird, dürfen Sie HTML-Tags in den Kommentaren verwenden. Spezielle Tags, wie zum Beispiel `@author`, werden direkt von `javadoc` ausgewertet, speziell behandelt und formatiert.