JAVA im Anfangsunterricht

Verschiedene Ausgaben auf der Konsole

Alfred Nussbaumer

Wegen des gewaltigen Sprachumfanges und der komplexen Klassenstruktur stellt JAVA im Anfangsunterricht hohe Anforderungen. Als eine Lösung bietet sich an, eigene Klassen zunächst auf der Konsole zu verwenden. Dabei sollen die Auszubildenden schrittweise an die Konzepte der Programmiersprache herangeführt werden.

In diesem Beitrag sollen einige Beispiele vorgestellt werden, die sich für „erste Schritte" eignen. Mit diesen Aufgaben sollen verschiedene Ergebnisse auf der Konsole ausgegeben werden. Sie stellen verschiedene Spracheigenschaften und einfache Algorithmen vor.

1. Grundlagen

JAVA läuft, zumindest was die erste Ausbildungsphase betrifft, auch auf älteren Systemen und kann in verschiedenen Betriebssystemumgebungen eingerichtet werden (vgl. **[1]**). Für die ersten Programme genügt ein einfacher Editor – der Quellcode kann auf der Konsole kompiliert und anschließend auf der Konsole ausgeführt werden:

```
nus@ice:~/java_bsp> javac Sinus.java
nus@ice:~/java_bsp> java Sinus
```

Achten Sie darauf, dass die Compiler-Version und die Version des auf Ihrem System vorhandenen JRE übereinstimmen – aktuelle Versionskonflikte betreffen etwa das JDK1.4.2 und JDK1.5.

Sobald Anwendungen für die grafische Benutzeroberfläche (vgl. **[3]**) entwickelt werden, ist eine entsprechende IDE (Eclipse, NetBeans, JBuilder, etc.) unverzichtbar.

2. Einfach formatierte Ausgaben

Von administrativen Aufgaben – etwa bei der Fernwartung eines Servers – sind Ausgaben auf der Textkonsole bekannt. Spielen gestalterische Argumente keine Rolle, kann man solche Ausgaben auch auf Webseiten finden. Im vorliegenden Beispiel `Tabelle.java` werden Zahlen und einfache Begrenzer mit Hilfe bestimmter Zeichen ausgegeben. Für die Anordnung sind so genannte Tabulatoren und Zeilenumbrüche wichtig – sie werden durch so genannte „Escape-Sequenzen" in den Ausgabetext eingefügt.

```
public class Tabelle {
  public static void linie() {
    System.out.println("+-------+-------+");
  }

  public static void main(String args[]) {
    linie();
    System.out.print("|  i"+"\t|  i*i"+"\t|\n");
    linie();
    for (int i=0;i<11;i++) {
      System.out.print("|  "+ i + "\t|  " + (i*i) + "\t|\n");
    }
    linie();
  }
}
```

An dieser Stelle wird die statische Methode `linie()` lediglich aus ökonomischen Zwecken eingesetzt: Die gleiche Folge von Zeichen wird dreimal ausgegeben. Zu beachten sind jedenfalls der horizontale Tabulator `\t` („tabulator") und die Zeilenschaltung `\n` („newline"). Sie sorgen dafür, dass verschieden große Zahlen untereinander ausgerichtet werden:

```
+-------+-------+
|  i    |  i*i  |
+-------+-------+
|  0    |  0    |
|  1    |  1    |
|  2    |  4    |
|  3    |  9    |
|  4    |  16   |
|  5    |  25   |
|  6    |  36   |
|  7    |  49   |
|  8    |  64   |
|  9    |  81   |
|  10   |  100  |
+-------+-------+
```

Komplizierter ist es, wenn die Zahlen rechtsbündig ausgerichtet werden sollen. In diesem Fall erhält man eine zufrieden stellende Ausgabe, wenn man die Zahlen vor der Ausgabe in Zeichenketten umwandelt, die der Zahlengröße entsprechend zusammen gesetzt werden. Die Klasse `Format` sowie die davon abgeleiteten Klassen `DateFormat`, `MessageFormat` und `NumberFormat` erlauben die Formatierung von Konsolenausgaben (Package `java.text`, vgl. **[4]**).

3. Klassisch: Funktionswerte mit Zeichen ausgeben

Eine passable Übung zu Datentypen ergibt sich in der Art der „Grafikausgabe", wie sie zu Zeiten der Kettenraddrucker üblich war und bis zum Aufkommen von grafikfähigen Bildschirmen und Betriebssystemumgebungen eingesetzt wurde: Ein * wurde in der Ausgabezeile um so viele Leerzeichen nach rechts verschoben, wie es der Funktionswert festlegt.

```
public class Sinus {

  static void amplitude(int wert) {
    for (int i = 0; i < wert; i++)
      System.out.print(" ");
    System.out.println("*");
  }

  public static void main(String args[]) {
    double x;
    for (int i=0;i<20;i++) {
      x = i;
      amplitude((int)(30*Math.sin(x/3) + 30.5));
    }
  }
}
```

Hier wird die Verwendung einer Methode offensichtlich. Entscheidend ist auch, zur Berechnung des Sinuswertes eine Fließkommazahl zu verwenden (`double x`). Da die Anzahl der Leerzeichen eine ganze Zahl sein muss, wird der Typ des Funktionswertes in eine ganze Zahl umgewandelt. Dabei gehen die Nachkommastellen verloren.

Im letzten Beispiel lassen sich natürlich auch die „x-Achse", etwa mit senkrechten Strichen, und eine Skalierung eintragen. Eine mögliche Lösung dafür und eine flexible Angabe für den Funktionsterm sollen im nächsten Abschnitt vorgestellt werden.

4. Parameter von der Kommandozeile übernehmen

- Mit der Klasse Schwingung werden die Amplitude `a`, die Frequenz `f` und die Phasenverschiebung `p` als Parameter von der Kommandozeile übernommen. Werden weniger als 3 Parameter eingegeben, wird der Hinweis für den korrekten Aufruf ausgegeben, und das Programm bricht mit `System.exit(0)` ab:

```
public class Schwingung {

  private static int a = 1;
  private static int f = 1;
  private static int p = 0;

  static void eingabe(String[] args) {
    if (args.length == 3) {
      a = (int) Double.parseDouble(args[0]);
      f = (int) Double.parseDouble(args[1]);
      p = (int) Double.parseDouble(args[2]);
```

```
    }
    else {
      System.out.println("Aufruf: java Schwingung Amplitude (-30..30)");
      System.out.println("Frequenz (-10..10) Phase (-360..360)");
      System.exit(0);
    }
  }

  static void plot(int wert, int modus) {
    String ergebnis = "";
    String leerzeichen = " ";
    if (modus == 0) a = "-";
    for (int i=0; i<60; i++) {
      if (i==wert+30) ergebnis += "*";
      else if (i==30) ergebnis += "|";
      else ergebnis += leerzeichen;
    }
    System.out.println(ergebnis);
  }

  static void ausgabe() {
    double x;
    System.out.println
      ("\n\n\tf(x) = " + a + " * sin (" + f + " * x + " + p + ")");
    for (int i=0;i<32;i++) {
      x = i;
      plot((int)(a*Math.sin(f*x/20+p/57.3)),i);
    }
  }

  public static void main(String args[]) {
    eingabe(args);
    ausgabe();
  }
}
```

Beachten Sie auch, wie in der Methode `plot(int wert, int modus)` die Zeichenkette `ergebnis` für die Ausgabe zusammengesetzt wird: Für die erste Zeichenkette werden „-" für die vertikale Achse gezeichnet, ansonsten besteht die Zeichenkette vorwiegend aus Leerzeichen. Nur an der Stelle des Funktionswertes wird ein „*" und in der Mitte ein „|" für die horizontale Achse angehängt. Diese Vorgangsweise ist notwendig, weil in JAVA einzelne Zeichen einer Zeichenkette nicht direkt verändert werden können. Dies unterscheidet String-Variable deutlich von Arrays, die im nächsten Abschnitt behandelt werden.

Die Konkatenation mit Hilfe des „+"-Operators verwenden wir auch, um den Funktionsterm mit den eingegebenen Parametern als Zeichenkette auszugeben.

```
nus@ice:~/java_bsp> java Schwingung 29.3 5.7 45
```

liefert mit den ganzzahligen Anteilen der Parameter:

```
        f(x) = 29 * sin (5 * x + 45)
-----------------------------|-------------------*---------
                             |                       *
                             |                         *
                             |                         *
                             |                       *
                             |                   *
                             |               *
                             |           *
                             |       *
                             |   *
                         *   |
                     *       |
                 *           |
             *               |
         *                   |
     *                       |
   *                         |
  *                          |
  *                          |
   *                         |
      *                      |
          *                  |
              *              |
                  *          |
                      *      |
                             |      *
                             |           *
                             |               *
                             |                   *
                             |                       *
                             |                         *
                             |                         *
                             |                       *
                             |                   *
```

Vergleichen Sie die Methode `main(String args[])` in den letzten Beispielen: Während sie in den ersten beiden Beispielen einfach einige Anweisungen enthielt, bestand sie im letzten Beispiel aus zwei Methodenaufrufen. Typisch für JAVA ist eine andere Vorgangsweise: Da jede Klasse einen eigenen Datentyp bildet, sollte die Methode `main()` folgendermaßen aussehen:

```
  public static void main(String args[]) {
    Schwingung proggi = new Schwingung();
    proggi.eingabe(args);
    proggi.ausgabe();
  }
```

Die Kommandozeilenparameter werden immer als „Datenfeld", als Array von Zeichenketten an die Methode `main(String args[])` übergeben. Innerhalb der Klasse werden die einzelnen Zeichenketten anhand ihres Index, beginnend bei 0, ausgelesen. Beachten Sie bitte, wie ein Array als Parameter an eine Methode übergeben wird!

5. Arrays und Zufallszahlen

Dass bestimmte Datenstrukturen mit passenden Algorithmen gekoppelt sind, sollte schon im Einführungsunterricht thematisiert werden. Im Beispiel soll der gewichtete Mittelwert aus den Daten eines Arrays gebildet werden. Dabei sollen einige tausend Zufallszahlen zwischen 0 und 9 auf ihre Häufigkeit hin untersucht werden. Zu zeigen, wie ein Array mit bekannter Größe mit Zählschleifen durchgegangen werden kann, ist ein Übungsziel dieses Beispiels.

```
import java.util.Random;

public class Zufall {

  static int feld[] = new int[10];
  static int umfang = 100000;

  public Zufall() {
    Random zahl = new Random();
    for (int z = 0; z < umfang; z++)
      feld[zahl.nextInt(10)]++;
  }

  public static void zeile (char zeichen) {
    for (int i = 0; i<50; i++)
      System.out.print(zeichen);
    System.out.println();
  }

  public static void tabelle() {
    System.out.println("Häufigkeit von " +
      umfang + " Zufallszahlen von 0 bis 9:");
    zeile('=');
    for (int i = 0; i < 10; i++)
      System.out.println("h("+i+")="+feld[i]);
  }

  public static void mittelwert() {
    double mittelwert = 0;
    for (int i = 0; i < 10; i++)
      mittelwert += feld[i]*i;
    mittelwert /= umfang;
    zeile('-');
    System.out.println("Mittelwert: " +
      mittelwert + " (theoretisch: 4.50)");
    zeile('=');
  }

  public static void main(String args[]) {
    if (args.length == 1) {
      umfang = Integer.parseInt(args[0]);
      Zufall proggi = new Zufall();
      proggi.tabelle();
      proggi.mittelwert();
    }
    else {
      System.out.println("Aufruf: java Zufall
        [anzahl]");
    }
  }
}
```

Hier verwenden wir die Klasse `Random` aus dem Package `java.util`: Beim Initialisieren einer Instanz wählt der Konstruktor dieser Klasse die Systemzeit als Seed-Wert und liefert Zufallszahlen mit einer Reihe von Methoden; hier gibt die Methode `nextInt(10)` eine Zufallszahl zwischen 0 und 9 zurück.

Die Klassenvariable `feld` ist ein eindimensionales Array. Jeder Index steht für eine Zufallszahl; dieser wird um 1 erhöht, wenn die zugehörige Zufallszahl aufgetreten ist. Dies findet im Konstruktor der Klasse statt. Wie viele Zufallszahlen ermittelt werden, hängt vom Wert der Variablen umfang ab, die beim Aufruf der Klasse als Kommandozeilenparameter übergeben wird. Fehlt dieser Wert beim Programmaufruf, so wird eine entsprechende Mitteilung ausgegeben:

```
nus@ice:~/java_bsp> java zufall
```

Aufruf: `java zufall [anzahl]`

Wird die Anzahl der Zufallszahlen beim Aufruf angegeben, erhalten wir nahezu augenblicklich die Auswertung:

```
nus@ice:~/java_bsp> java Zufall 100000
Häufigkeit von 100000 Zufallszahlen von 0 bis 9:
================================================
h(0) = 9997
h(1) = 9982
h(2) = 10098
h(3) = 9907
h(4) = 9886
h(5) = 10088
h(6) = 10025
h(7) = 10076
h(8) = 9984
h(9) = 9957
------------------------------------------------
Mittelwert: 4.5005 (theoretisch: 4.50)
================================================
```

6. Den Quelltext ausgeben

Das letzte Beispiel gibt den Quelltext einer Textdatei aus.

```
import java.io.*;

public class Dateilesen {

  public static void main(String args[])
    throws Exception {

    String dateiname = "dateilesen.java";
    try {
      FileReader fr = new FileReader(dateiname);
      BufferedReader br =new BufferedReader(fr);
      String ausgabe = br.readLine();
      while (ausgabe != null) {
        System.out.println(ausgabe);
        ausgabe = br.readLine();
      }
    }
    catch (FileNotFoundException e) {
      System.out.println("Datei nicht
        vorhanden");
    }
  }
}
```

Zunächst wird ein `FileReader`-Objekt erzeugt, das die angegebene Datei zum Lesen öffnet. Das `BufferedReader`-Objekt übernimmt den Inhalt der Textdatei als Zwischenspeicher. Die `BufferedReader`-Methode `readLine()` liest aus ihm den Inhalt Zeile für Zeile aus.

Da die Größe der Textdatei im Allgemeinen nicht bekannt ist, wählt man hier eine `while`-Schleife für die Ausgabe: Die `BufferedReader`-Methode `readLine()` liefert den Wert `null`, wenn der Dateizeiger das Ende der Datei erreicht hat: In diesem Fall bricht die `while`-Schleife ab.

```
nus@ice:~/java_bsp> java Dateilesen
import java.io.*;

public class Dateilesen {
...
}
```

7. Ausblick, Übungen

Weitere Beispiele können in mannigfaltiger Weise angeschlossen werden, je nach Zielsetzung des Kurses. Für Aufgabenstellungen auf der Textkonsole sind rekursive Funktionen, das Importieren von Daten oder erste Schritte bei der Programmierung von Netzwerkdiensten denkbar.

- Beurteilen Sie, ob es im Beispiel `Sinus.java` günstiger wäre, anstelle der Typenanpassung `(int)` zu runden (`Math.round()`)!
- Schreiben Sie eine Java-Klasse `Polynom.java`, mit der Polynomfunktionen ausgegeben werden! Die Koeffizienten sollen als Parameter beim Aufruf angegeben werden.
- Wie müsste der Programmcode erweitert werden, damit die Klasse `Dateilesen` beliebige Textdateien ausgibt, deren Pfade auf der Kommandozeile als Parameter übergeben werden?
- In der Kryptographie spielt bei bestimmten Verschlüsselungsalgorithmen die Analyse der Buchstabenhäufigkeit eine wichtige Rolle. Schreiben Sie ein Programm, das die Häufigkeit der Buchstaben eines beliebigen Textes ermittelt (vgl. auch **[2]**)!
- Die Häufigkeit von Gauss-verteilten Zufallszahlen ist zu ermitteln und grafisch darzustellen:

```
300
   -10:
   -9:
   -8:
   -7:
   -6:
   -5:  ****
   -4:  ***********
   -3:  ******************************
   -2:  ********************************************
   -1:  ************************************************************
   0:   ************************************************************
   1:   ********************************************
   2:   ******************************
   3:   *****************
   4:   *********
   5:   *
   6:
   7:
   8:
   9:
   10:
```

- Begründen Sie, warum die Zählschleife für Arrays mit bekannter Länge und warum die `while`-Schleife für das Einlesen von Texten unbekannter Länge verwendet werden! Formulieren Sie die Algorithmen jeweils mit alternativen Schleifen (Zählschleife, kopf- und fußgesteuerte Programmschleife)!

Rekursiv verwendete Funktionen lassen interessante und zum Teil schwierige Aufgabenstellungen zu. Trotz der einfachen Ausgabe auf der Konsole können dabei überraschende und unerwartete Ergebnisse studiert werden. Einige prominente Beispiele davon sollen ein einer weiteren Folge dieser JAVA-Beiträge behandelt werden.

8. Literatur, Weblinks

[1] PCNEWS Nr. 72, April 2001, S. 46, „JAVA“

[2] PCNEWS Nr. 79, Sept. 2002, S. 33, „JAVA – Textdateien durchsuchen“

[3] PCNEWS Nr. 92, Februar 2005, S. 18, „JAVA und die grafische Benutzeroberfläche“

[4] *http://java.sun.com/j2se/1.5/docs/index.html* (Dokumentation aller verfügbaren Packages)

[5] *http://java.sun.com/docs/books/tutorial/* (JAVA-Tutorial der Firma SUN)

[6] Guido Krüger, „Handbuch der JAVA-Programmierung“, Addison & Wesley, ISBN 3-8273-2201-4

[7] *http://www.javabuch.de/* („Handbuch der JAVA-Programmierung“, freier Download für schulische Zwecke)

[8] Herbert Schildt, „JAVA – Grundlagen der Programmierung“, mitp, IBSN 3-8266-1524-7

[9] Udo Müller, „JAVA – das Lehrbuch“, mitp, ISBN 3-8266-1333-3

[10] Christian Ullenboom, „Java ist auch eine Insel“, Galileo Computing, ISBN 3-89842-365-4

[11] *http://de.wikibooks.org/wiki/Java* (Lernmaterialien als WikiBook)

[12] *http://www.gymmelk.ac.at/nus/informatik/wpf/JAVA* (Unterrichtsbeispiele zum Programmieren mit JAVA, Quelltexte zum Downloaden)