Arrays

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

Wir werden in dieser Vorlesung das Deklararieren, Initialisieren, Lesen und Schreiben von Arrays kennenlernen. In diesem Zusammenhang führen wir außerdem das Konzept der `for`-Schleife ein, da dieser Schleifentypus häufig bei der Programmierung mit Arrays angewendet wird. Außerdem erläutern wir den Begriff Referenzvariablen. Dieser Abschnitt schließt mit einer kurzen Einführung in das Lesen und Schreiben auf Ein- und Ausgabeströme von Daten.

1 Arrays

Kapitel 1.4 aus R. Sedgewick, K. Wayne: Einführung in die Programmierung mit Java. 2011, Pearson Studium.

Arrays werden zur Speicherung und Verarbeitung größerer Datenmengen verwendet. Sie ermöglichen es mehrere Werte vom gleichen Typ kompakt zu addressieren und zu verwalten.

Definition Ein ***Array (dt. Feld)*** ist ein n-Tupel von Variablen des gleichen Typs.

Die genaue Syntax und Semantik von Arrays ist von Sprache zu Sprache verschieden. Wir betrachten hier zunächst die Umsetzung von Arrays in Java.
Mit `T[]` wird in Java der Typ von Arrays mit Komponenten vom Typ T bezeichnet.

Syntax für Array-Typen :

Typ → Typ `[]`

1.1 Deklaration und Initialisierung von Arrays

Die Deklaration einer Variablen `a` für ein Array mit Elementen vom Typ `T` erfolgt durch

```
T[] a;
```

Sie stellt den Speicherplatz für die Variable bereit, mit der das Array referenziert wird. Diese Art von Variable wird auch ***Referenzvariable*** genannt. Die Deklaration einer Variable allein erzeugt kein Arrayobjekt!

Syntax für das Erstellen von Arrays :

`Ausdruck` →
    `new` `ArrayElementTyp` `[` `Ausdruck` `]`

`ArrayElementTyp` → `PrimitiverTyp` | ≪ *Bezeichner* ≫

Ein neues Arrayobjekt mit n Komponenten vom Typ T wird von dem Ausdruck

```
new T[n]
```

erzeugt. `n` ist dabei ein Ausdruck, der zu einem ganzzahligen Wert auswertet. Der Ausdruck `new T[n]` liefert eine Referenz auf das neu erzeugte Array als Ergebnis.
Die Komponenten des Arrays sind zunächst mit dem Initialwert des Typs T initialisiert.

- Für alle numerischen Datentypen ist der Initialwert 0 bzw. 0.0.
- Für `boolean` ist er `false`.
- Für `String`s und andere nicht-primitive Typen ist er `null`.

Beispiel Folgende Anweisung erzeugt ein Array der Größe `N` mit Zahlen vom Typ `double`, die alle zunächst mit `0.0` initialisiert sind und dann auf `1.0` gesetzt werden.

```
int N = ...;
double[] a = new double[N];

// Veraendern der Array-Inhalte
int i = 0;
while (i < N) {
  a[i] = 1.0;
  i = i + 1;
}
```

Sei im Folgenden `exp` ein Ausdruck von einem ganzzahligen Typ (`byte`, `short`, `int`, `long`), der sich zu einem Wert k auswertet.
Ausdrücke zum Lesen und Schreiben eines Arrays `a`:

- `a.length` liefert die Anzahl der Arraykomponenten.
- `a[exp]` ist als

  R-Wert: der in der k-ten Arraykomponente gespeicherte Wert;

  L-Wert: die k-te Arraykomponente,
  d.h. z.B. wird durch die Anweisung `a[exp] = 7;` der k-ten Arraykompente der Wert 7 zugewiesen.

Bemerkungen:

- Das erste Array-Element eines Arrays `a` ist mit Index 0 indiziert (`a[0]`), das zweite an Index 1 (`a[1]`), usw.

- Das letzte Element ist an Position `a.length-1` zu finden.
- Bei Zugriff auf ein Array-Element muss sichergestellt sein, dass der Indexwert zwischen 0 und `a.length-1` ist. Andernfalls passiert ein `ArrayIndexOutOfBounds`-Fehler, der die reguläre Ausführung des Programms abbricht.

Syntax zum Lesen und Schreiben von Arrays :

```
Ausdruck →
     Ausdruck . length
   | Ausdruck [ Ausdruck ]

Zuweisung → Ausdruck [ Ausdruck ] = Ausdruck;
```

1.2 Einschub: for-Anweisung (Zählanweisung)

Die for-Anweisung dient vorrangig zur Bearbeitung von Arrays, deren einzelne Komponenten über Indizes angesprochen werden. Wir diskutieren sie daher in diesem Kontext.
Syntax in Java:

```
Anweisung →
     for (forInit; Ausdruck; forUpdate) Anweisung

forInit → Typ  ≪ Bezeichner ≫  = Ausdruck

forUpdate →
     ≪ Bezeichner ≫  =  Ausdruck
   | Ausdruck ++
   | Ausdruck --
```

- Die Initialisierungsanweisung forInit (deklariert und) initialisiert eine Zählvariable.
- Die Updateanweisung forUpdate ist typischerweise entweder
  - eine direkte Zuweisung (d.h. der Wert der Variablen wird auf den Wert eines Ausdrucks gesetzt und zurückgeliefert),
  - das Inkrementieren einer Variable durch den `++` Operator (d.h. der Wert der Variable wird um 1 erhöht und zurückgeliefert), oder
  - das Dekrementieren einer Variable durch den `--` Operator (d.h. der Wert der Variablen wird um 1 erniedrigt und zurückgeliefert)

Semantik:

- Es wird zunächst die Initialisierungsanweisung ausgeführt.
- ($*$) Als nächstes wird der boolsche Ausdruck ausgewertet. Falls er zu `false` auswertet, ist die Ausführung der Schleife beendet. Falls er zu `true` auswertet, wird der Schleifenrumpf (Anweisung) ausgeführt. Dannach wird die Updateanweisung ausgeführt.

- Die Auswertung wiederholt sich nun ab (∗).

Beispiele Arrays können dazu verwendet werden, um Vektoren zu implementieren. Das folgende Programm initialisiert zwei 3-elementige Vektoren mit Zufallszahlen, addiert sie und gibt das Ergebnis auf der Konsole aus.

```
public class VectorExample {
  public static void main(String[] args){
    int[] a = new int[3]; // 3-elementiger Vektor
    int[] b = new int[3];
    for (int i = 0; i < 3; i++){
      a[i] = (int) (Math.random() * 10); //erzeugt Zufallszahl zwischen 0
     und 9
      b[i] = (int) (Math.random() * 10);
    }

    int[] c = new int[3];
    for (int i = 0; i < 3; i++){
      c[i] = a[i] + b[i];
    }
    for (int i=0; i<3; i++) {
      System.out.print("a["+i+"] = "+a[i]);
      System.out.print(", b["+i+"] = "+b[i]);
      System.out.println(", c["+i+"] = "+c[i]);
    }
  }
}
```

Hinweis: Die Ausgabe mit `System.out.println()` beendet die Ausgabe mit einem Zeilenumbruch, `System.out.print()` macht diesen Zeilenumbruch nicht.
Als weiteres Beispiel haben wir noch ein Code-Snippet, mit dem das Maximum der Elemente eines Arrays ermittelt werden kann.

```
double[] a = new double[10];
... // Initialisierung von a mit Double-Werten

double max = a[0];
for (int i = 1; i < a.length; i++) {
  if(max < a[i]) {
    max = a[i];
  }
}
```

1.3 Fallbeispiel: Sieb des Eratosthenes

Um alle Primzahlen zu ermitteln, die kleiner oder gleich einem Eingabeparameter n sind, eignet sich der folgende Algorithmus, der als das "Sieb des Eratosthenes" bekannt ist.
Zunächst werden alle Zahlen 2, 3, 4, ... bis n aufgeschrieben. Die zunächst unmarkierten Zahlen sind alles potentielle Primzahlen. Die kleinste unmarkierte Zahl in diesem Verfahren ist immer eine Primzahl. Wenn eine Primzahl gefunden wird, werden alle Vielfachen dieser Primzahl als Nicht-Primzahlen markiert.

Für den Algorithmus bestimmt man immer die nächste nichtmarkierte Zahl. Da sie kein Vielfaches von Zahlen kleiner als sie selbst ist (sonst wäre sie markiert worden), kann sie nur durch eins und sich selbst teilbar sein. Folglich muss es sich um eine Primzahl handeln. Diese wird dementsprechend als Primzahl ausgegeben. Im nächsten Schritt streicht alle Vielfachen dieser Zahl und führt das Verfahren fort, bis man am Ende der Liste angekommen ist.

```
public class Eratosthenes {
  public static void main(String[] args) {
    // Eingabeparameter n
    int n = Integer.parseInt(args[0]);
    boolean[] isPrime = new boolean[n+1];
    // Initialisierung des Arrays mit true
    for (int i = 0; i < n+1; i++) {
        isPrime[i] = true;
    }
    // 2 ist die kleinste Primzahl
    for (int i = 2; i < n+1; i++) {
      if (isPrime[i]) {
        System.out.print(i + " ");
        for (int j = 2; i * j < n+1; j++) {
          // Alle Vielfachen von i koennen keine Primzahlen sein
          isPrime[i*j] = false;
        }
      }
    }
  }
}
```

In dieser Implementierung wird ein Array mit boolschen Werten angelegt, das die Information verwaltet, ob eine Zahl eine Primzahl ist. Für alle Zahlen wird diese Array zunächst mit `true` initialisiert (potentielle Primzahlen). Ausgehend von der kleinsten Primzahl 2 wird nun dieses Array durchlaufen. Die nächste nichtmarkierte Primzahl, d.h. der nächste Index i für den `isPrime[i]` den Wert `true` enthält, wird ausgegeben. Dannach werden die Vielfachen als Nicht-Primzahl markiert, indem der Wert des Arrays an den entprechenden Indizes auf `false` gesetzt wird.
Das Verfahren lässt sich folgendermaßen optimieren:

- Da mindestens ein Primfaktor einer Nicht-Primzahl immer kleiner gleich der Wurzel der Zahl sein muss, ist es ausreichend, nur die Vielfachen von Zahlen mit `false` zu markieren, die kleiner oder gleich der Wurzel von n sind.
- Es genügt beim Streichen der Vielfachen, mit dem Quadrat der Primzahl zu beginnen, da alle kleineren Vielfachen bereits markiert worden sind.

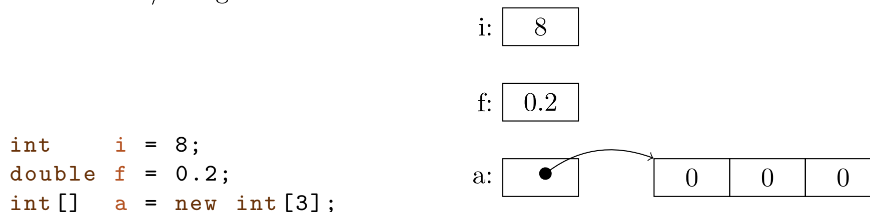
```
public class EratosthenesOptimized {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    boolean[] isPrime = new boolean[n+1];
    for (int i = 0; i < n+1; i++) {
        isPrime[i] = true;
    }
    // Mindestens ein Primfaktor einer Nicht-Primzahl muss
    // immer kleiner gleich der Wurzel der Zahl sein
```

```
    for (int i = 2; i*i < n+1; i++) {
      if (isPrime[i]) {
        // Es genuegt nur die Eintraege ab i zu betrachten
        for (int j = i; i*j < n+1; j++) {
          isPrime[i*j] = false;
        }
      }
    }
    for (int i = 2; i < n+1; i++) { // Ausgabe
        if (isPrime[i]) {
            System.out.print(i + " ");
        }
    }
  }
}
```

Diese Optimierung beschleunigt zwar die Berechnung, führt aber dazu, dass der Algorithmus weniger intuitiv und schwerer nachzuvollziehen ist.
Programmierer müssen immer gut abwägen, ob Optimierungen tatsächlich sinnvoll sind. Die Wiederverwendung von Variablen, um Deklarationen "zu sparen" ist beispielsweise keine sinnvolle Optimierung. Sie führt zu schwer verständlichem Programmtext ohne das Laufzeitverhalten tatsächlich zu verändern. Moderne Compiler können diese Art der Optimierung automatisch durchführen.

2 Referenzvariablen

Variablen speichern Werte, d.h. elementare Daten von primitiven Datentypen oder Referenzen auf Objekte (z.B. Arrays, Strings). Letztere nennt man auch Referenzvariablen. Sie enthalten als Werte Referenzen auf den Speicherbereich, der die eigentlichen Daten enthält. Diese Referenzen können Variablen zugewiesen werden, genauso wie Werte primitiver Datentypen. Es können auch mehrere Variablen den gleichen Speicherbereich / die gleichen Daten referenzieren.



Ein spezieller Referenzwert ist `null`. Dieses Literal repäsentiert eine Referenz, der auf nichts verweist. Die Null-Referenz kann nicht dereferenziert werden, d.h. Lesen und Schreiben von Daten, die mit `null` referenziert werden, ist nicht möglich und führt in Java zu einem Fehler (`NullpointerException`).
Achtung: Die Operationen auf Referenzvariablen, wie z.B. Vergleiche oder Zuweisungen, arbeiten mit den Referenzen, nicht mit den referenzierten Objekten.

Beispiel: Vergleichen und Kopieren von Arrays

Frage 1: Was ist die Ausgabe des folgenden Programms?
Führen Sie folgendes Beispiel im Java Visualizer[1] aus!

```
public class Arrays {
  public static void main(String[] args) {
    int n = 5;
    int[] a = new int[n];
    for (int i = 0; i < n; i++) {
      a[i] = i;
    }
    int[] b = a; // a und b referenzieren das gleiche Objekt
    System.out.println(a == b);

    int[] c = new int[n];
    for (int i = 0; i < n; i++) {
      c[i] = a[i]; // c referenziert eine Kopie von a
    }
    System.out.println(a == c);

    b[2] = 100;
    System.out.println(a[2] + " vs " + c[2]);
  }
}
```

Beispiel: Vergleichen von Strings

Frage 2: Was ist die Ausgabe des folgenden Programms?
Visualisieren Sie folgendes Beispiel im Java Visualizer und verwenden Sie dabei die Option “ Show String/Integer/etc objects, not just values”!

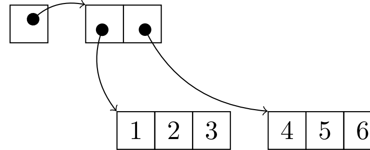
```
public class Strings {
  public static void main(String[] args) {
    String a = "Hello, world!";
    String b = "Hello, world!!".substring(0, 13);
    String c = "Hello, ";
    c = c + "world!";
    String d = "Hello, w"+"orld!";
    String e = a.substring(0, 13);
    System.out.println((a == b) + " " + a.equals(b));
    System.out.println((a == c) + " " + a.equals(c));
    System.out.println((a == d) + " " + a.equals(d));
    System.out.println((a == e) + " " + a.equals(e));
  }
}
```

[1] Online unter `http://cscircles.cemc.uwaterloo.ca/java_visualize/`

3 Mehrdimensionale Arrays

Mehrdimensionale Arrays sind “Arrays von Arrays”. Die Initialisierung erfolgt wie bei eindimensionalen Arrays durch Angabe der Anzahl der Elemente je Dimension.[2]

```
int[][] a = new int[2][3];
a[0][0] = 1;
a[0][1] = 2;
a[0][2] = 3;
a[1][0] = 4;
a[1][1] = 5;
a[1][2] = 6;
```



Beispiel: Matrizen als mehrdimensionale Arrays
Um die Multiplikation zweier $n \times n$-Matrizen a und b zum implementieren, müssen verschachtelte for-Anweisungen verwendet werden.

```
double[][] a = ...; // Initialisierung
double[][] b = ...; // Initialisierung
double[][] c = new double[n][n];
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    for (int k = 0; k < n; k++)
      c[i][j] = c[i][j] + a[i][k] * b[k][j];
```

3.1 Ungleichförmige Arrays

In Java ist es nicht notwendig, dass alle Zeilen bzw. Subarrays die gleiche Größe haben. Der Umgang mit ungleichförmigen Arrays erfordert aber besondere Sorgfalt, um `ArrayIndexOutOfBounds`-Fehler zu vermeiden.
Im folgenden Beispiel wird ein mehrdimensionales Array konstruiert, dessen erster Eintrag auf ein Array der Größe 2, der zweite auf ein Array der Größe 1, und der dritte Eintrag auf ein Array der Größe 3 verweist. Bei der Iteration über die Einträge wird für jedes dieser Arrays die Größe ermittelt (`j < a[i].length;`).

```
int n = 3;
int[][] a = new int[n][];
a[0] = new int[2];
a[1] = new int[1];
a[2] = new int[3]
for (int i = 0; i < n; i++) {
  for (int j = 0; j < a[i].length; j++) {
    System.out.print(a[i][j]+ " ");
  }
  System.out.println();
}
```

[2]Die formale Syntaxdefinition finden Sie in der Zusammenfassung am Ende.

3.2 Initialisieren von Arrays

Beim Erstellen eines Arrays können bereits initiale Werte für das Array angegeben werden.
Beispiele

```
int[] a = new int[] {1, 2, 3};
int[] b = {1, 2, 3};
int[][] c = {{1, 2, 3}, {4, 5}, {6}};
```

4 Fallstudien: Simulationen

Simulationen werden in vielen Bereichen (Wirtschaft, Technik, Naturwissenschaften) eingesetzt, um Modelle zu erstellen und zu validieren. Sie ergänzen die (math.) Analyse und ersetzen sie bisweilen sogar in komplexen Situationen.

4.1 Beispiel: Sammelbilder

Seite 121 ff aus R. Sedgewick, K. Wayne: Einführung in die Programmierung mit Java. 2011, Pearson Studium.

Sammelkarten sind sehr beliebt bei Kindern, die gerne ihr Taschengeld dafür ausgeben ein Sammelheft mit Fussballern, Tierbabys oder ihren Lieblingsserienhelden zu füllen.

Wie viele Sammelkarten muss man im Mittel kaufen, um eine Serie von n Bildern zu erhalten (ohne Tauschen, alle Karten gleich wahrscheinlich)?

Um diese Frage zu beantworten, werden wir den Kauf im Programm simulieren. Die Karten werden dabei als Array von n boolschen Werten repräsentiert, initialisiert mit `false`, da zu Beginn noch keine Karten gekauft wurden. Es wird nun eine Zufallszahl zwischen 0 und $n-1$ erzeugt, um den zufälligen Kauf einer Karte zu modellieren. Der Wert an der entsprechenden Arrayposition wird dann auf `true` gesetzt (falls er noch nicht `true` ist). Der Simulationsdurchlauf endet, wenn alle Werte im Array `true` sind. Dabei wird gezählt, wie viele Versuche dazu notwendig waren.

```
public class Sammelkarten {
  public static void main (String[] args) {
    int n            = Integer.parseInt(args[0]);
    boolean[] karte  = new boolean[n];
    int anzahl       = 0;
    int versuche     = 0;

    while (anzahl < n) {
      int naechste = (int) (Math.random() * n);
      versuche++;

      if (!karte[naechste]) {
        anzahl++;
        karte[naechste] = true;
```

```
      }
    }
    System.out.println("Anzahl an Versuchen: " + versuche);
  }
}
```

Eine anschauliche Erklärung zur mathematischen Analyse dieses Problems finden Sie im Spiegel Online Artikel "So funktioniert die Panini-Formel"[3] vom 23.04.2014.

4.2 Spielbankbesuche

Seite 87 ff aus R. Sedgewick, K. Wayne: Einführung in die Programmierung mit Java. 2011, Pearson Studium.

Wir betrachten hier zunächst eine Simulation von Gewinnwahrscheinlichkeiten beim Besuch von Spielbanken. Angenommen ein Spieler startet mit einem gegebenen Startkapital und will einen festgelegten Zielbetrag erreichen. Bei jeder Wette setzt er 1$ und kann entweder 1$ gewinnen oder verlieren. Wir nehmen an, dass das Kasino ein faires Spiel spielt (d.h. die Gewinnwahrscheinlichkeit liegt bei 50%). Das Spiel ist beendet, wenn der Spieler pleite ist oder wenn er den Zielbetrag erreicht hat.

Wie hoch sind die Chancen, dass der Spieler den Zielbetrag erreicht?
Wie viele Wetten sind dazu notwendig?

Folgendes Programm simuliert den Spielbankbesuch. Es nimmt drei Programm-Parameter: den Einsatz/Startkapital, den Zielbetrag und die Anzahl an Versuchen. Es ermittelt die durchschnittliche Erfolgswahrscheinlichkeit und die Anzahl an Versuchen, die durchschnittlich nötig waren um zu gewinnen bzw. zu verlieren.

```
public class Wettspiel {
  public static void main (String[] args) {
    int einsatz  = Integer.parseInt(args[0]);
    int ziel     = Integer.parseInt(args[1]);
    int versuche = Integer.parseInt(args[2]);
    int wetten   = 0;
    int gewinne  = 0;

    for (int t = 0; t < versuche; t++) {
      int cash = einsatz;
      while (cash > 0 && cash < ziel) {
        wetten++;
        if (Math.random() < 0.5) {
          cash++;
        } else {
          cash--;
        }
        if (cash == ziel) {
          gewinne++;
        }
      }
    }
    System.out.println(100.0 * gewinne/versuche + "% Erfolg");
    System.out.println("Durchschnittl. Anzahl Wetten: "
      + ((double) wetten/versuche);
  }
}
```

[3] Artikel unter http://spon.de/aec54

Bei der Simulation lassen sich folgende Beobachtungen machen:

- Die Erfolgswahrscheinlichkeit ist gegeben durch das Verhältnis von Einsatz zu Zielwert.
  Beispiel: Bei $500 Einsatz wird das Ziel $2500 in 20% aller Fälle erreicht.
- Die durchschnittliche Anzahl der Wetten ist gegeben durch das Produkt von Einsatz und Zielgewinn.
  Beispiel: Um aus $500 Einsatz $2500 Gewinn zu machen, braucht man im Durchschnitt etwa 1 Million Versuche.

Frage 3: Terminiert die gegebene Implementierung für alle Eingaben?
Wie kann man die Implmentierung abändern, so dass die Terminierung immer nach endlich vielen Schritten gewährleistet ist?

5 Eingabe und Ausgabe

Programme interagieren und kommunizieren mit der Ausführungsumgebung. Bisher haben wir Eingaben als Programmparameter auf der Kommandozeile beim Programmstart übergeben und Ausgaben auf die Konsole gemacht. Es gibt viele weitere Möglichkeiten Informationsschnittstellen zu gestalten:

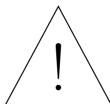
- Grafik
- Audio
- Video
- Drucker, etc...

In der Vorlesung SE1 werden wir uns allerdings auf einfache Interaktionsmöglichkeiten über die Kommandozeile beschränken.

5.1 Standardausgabe

Die Standardausgabe ist ein abstrakter Zeichenstream unbegrenzter Größe, der mit dem Konsolenfenster verbunden ist. Im Folgenden verwenden wir die Bibliothek `StdOut`, die verschiedene Prozeduren bereitstellt:

`void print(String s)`	Gibt den String `s` aus
`void println(String s)`	Gibt den String `s` gefolgt von einem Zeilenumbruch aus
`void println()`	Gibt einen Zeilenumbruch aus
`void printf(String f,...)`	Formatierte Ausgabe (⇒ siehe Übungen)



Die Signaturen in der Bibliothek `StdOut` weichen hiervon minimal ab, können aber genauso verwendet werden, wie hier beschrieben.

5.2 Standardeingabe

Analog zur Standardausgabe ist die Standardeingabe ein abstrakter Zeichenstream, der leer ist oder eine Folge von Werten enthält, die duch Leerzeichen, Tabulatoren, Zeilumbruchzeichen, etc. von einander getrennt sind. Beim Lesen werden die Werte *verbraucht*, d.h. eingegebene Werte können nur einmal aus der Standardeingabe eingelesen werden. Hier ein Ausschnitt aus der Bibliothek `StdIn`:

`boolean isEmpty()`	Testet, ob es weitere Werte gibt
`int readInt()`	Liest einen Wert vom Typ `int`
`double readDouble()`	Liest einen Wert vom Typ `double`
`boolean readBoolean()`	Liest einen Wert vom Typ `boolean`
`String readString()`	Liest einen Wert vom Typ `String`
`String readLine()`	Liest den Rest der Zeile
`String readAll()`	Liest den Rest des Textes

Bei fehlerhafter Eingabe wird ein `NumberFormatException`-Fehler geworfen (z.B. wenn versucht wird ein Integer einzulesen, aber die kein Integer eingegeben wurde).

5.3 Interaktive Benutzereingaben

Alle Zeicheneingaben nach der Befehlszeile (inkl. Befehlszeilenparametern) bilden den Eingabestream. Der Eingabestream wird *zeilenweise* verfügbar gemacht, d.h. erst nach Drücken der Eingabetaste (Enter) stehen die nächsten Werte des Streams zur Verfügung. Das **EOF**-Zeichen (End-of-File) gibt das Ende der Eingabe an. (Unter Linux kann dieses Zeichen erzeugt werden durch die Eingabe eines Zeilenumbruchs gefolgt von Strg-D[4], unter Windows durch Strg-Z gefolgt von einem Zeilenumbruch).

Beispiel: Verarbeitung von Eingaben beliebiger Länge

```
// Berechnet den Mittelwert aller Eingaben
public class Mittelwert {
  public static void main(String[] args) {
    double sum = 0.0;
    int count = 0;
    while (!StdIn.isEmpty()) {
      double value = StdIn.readDouble();
      sum = sum + value;
      count++;
    }
    StdOut.println("Mittelwert: " + sum/count);
  }
}
```

Das Programm kann folgendermaßen verwendet werden:

```
> java Mittelwert
10.0 5.0 6.0
3.0
7.0 32.0
<crtl-d>
Mittelwert: 10.5
```

[4]Das heißt man drückt die Steuerungstaste (Str oder Crtl) und gleichzeitig die Taste mit dem Buchstaben D

5.4 Umleiten von Eingaben und Ausgaben

Oft sind die Daten zu umfangreich für manuelle Eingabe, oder es sollen Ergebnisse zur späteren Verwendung gespeichert werden.
Man kann dazu die Standardausgabe in eine Datei umleiten (hier: in Datei `data.txt`):

```
java Mittelwert > data.txt
```

Ebenso lassen sich Daten aus einer Datei in die Standardeingabe umleiten (hier: aus Datei `data.txt`):

```
java Mittelwert < data.txt
```

Soll die Ausgabe eines Programms direkt an ein anderes Programm weitergeleitet werden, so kann dazu der Pipe-Operator verwendet werden:

```
java Range 0 100 2 | java Mittelwert
```

Zusammenfassung: Neue Sprachelemente

```
Typ →
    ...
  | Typ []

Ausdruck →
    ...
  | new ArrayElementTyp  [ Ausdruck ]
  | new ArrayElementTyp  ArrayLaengen ArrayTypKlammern
  | new ArrayElementTyp ArrayTypKlammern ArrayKonstante
  | null
  | Ausdruck . length
  | Ausdruck [ Ausdruck ]

ArrayElementTyp → PrimitiverTyp | ≪ Bezeichner ≫

Zuweisung →
    ...
  | Ausdruck [ Ausdruck ] = Ausdruck;

Anweisung →
    ...
  | for (forInit; Ausdruck; forUpdate) Anweisung

forInit → Typ  ≪ Bezeichner ≫  = Ausdruck

forUpdate →
    Zuweisung
  | Ausdruck ++
  | Ausdruck --


ArrayLaengen →
     [ Ausdruck ] ArrayLaengen
  |  [ Ausdruck ]

ArrayTypKlammern →
     []
  | ε


Deklaration →
    ...
  | Typ  ≪ Bezeichner ≫  =  ArrayKonstante;
```

```
ArrayKonstante →
      { ArrayEintragListe }
   | {}

ArrayEintragListe →
      Ausdruck , ArrayEintragListe
      ArrayKonstante , ArrayEintragListe
   | Ausdruck
   | ArrayKonstante
```