TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

Felder / Arrays

Arrays

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

- Oft verwendet zur Speicherung und Verarbeitung großer Datenmengen
- Die genaue Syntax und Semantik von Arrays ist von Sprache zu Sprache verschieden.

Deklaration und Initialisierung I

Mit `T[]` wird in Java der Typ von Arrays mit Komponenten vom Typ T bezeichnet.

Die Deklaration einer Variablen

```
T[] a;
```

stellt nur den Speicherplatz für die Variable bereit, mit der das Array referenziert wird.
Sie erzeugt kein Arrayobjekt!

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

```
new T[n]
```

erzeugt. Der Ausdruck liefert eine Referenz auf des Array als Ergebnis.

Deklaration und Initialisierung II

Die Komponenten sind 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 Feldinhalte
int i = 0;
while (i < N) {
  a[i] = 1.0;
  i = i + 1;
}
```

Programmieren mit Arrays

Sei im Folgenden `exp` ein Ausdruck von einem ganzzahligen Typ (byte, short, int, long), der sich zu 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. weist `a[exp] = 7;` der k-ten Arraykompente den Wert 7 zu.

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`.
- Bei Zugriff auf ein Array-Element muss sichergestellt sein, dass der Indexwert zwischen 0 und `a.length-1` ist. ($\Rightarrow$ `ArrayIndexOutOfBounds`-Fehler)

Einschub: for-Anweisung (Zählanweisung) I

Syntax in Java:

```
for (<forInit>;
     <boolescherAusdruck>;
     <forUpdate> )
        <Anweisung>
```

- Die Initialisierungsanweisung <forInit> (deklariert und) initialisiert eine Zählvariable.
- Die Updateanweisung ist 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)

Einschub: for-Anweisung (Zählanweisung) II

Semantik:
Es wird zunächst die erste Ausdrucksanweisung ausgeführt (sog. Initialisierung).

($*$) 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 ($*$).

Die for-Anweisung dient vorrangig zur Bearbeitung von Arrays, deren einzelne Komponenten über Indizes angesprochen werden.

Beispiel: Vektoren als Arrays I

Initialisiere zwei 3-elementige Vektoren mit Zufallszahlen und addiere sie.

```
public class VectorTest {
  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);
      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]);
    }
  }
}
```

Beispiel: Vektoren als Arrays II

Berechne das Maximum der Elemente eines Arrays.

```
double[] a = new double[35];
... // Initialisierung von a mit beliebigen Werten

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

Primzahlen

- Was ist eine Primzahl?
- Skizzieren Sie ein Programm, das alle Primzahlen ausgibt, die kleiner als ein Eingabeparameter n ist!

Beispiel: Sieb des Eratosthenes I

Berechnet die Primzahlen kleiner oder gleich dem Eingabeparameter n.

```
public class Eratosthenes {
  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;
    }
    // 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++) {
          isPrime[i*j] = false;
        }
      }
    }
  }
}
```

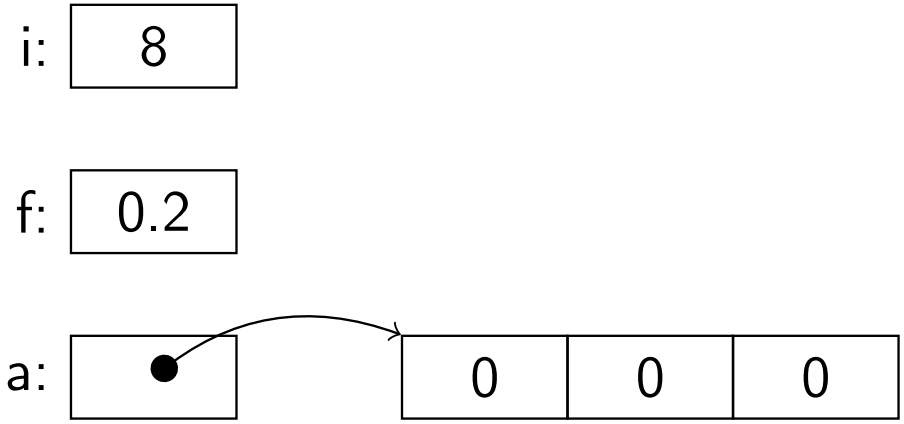
Beispiel: Sieb des Eratosthenes II

```
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,
        // da die kleineren Vielfachen bereits markiert wurden!
        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.println(i + " ");
        }
    }
  }
}
```

Referenzvariablen

- Variablen speichern Werte, d.h. elementare Daten von primitiven Datentypen oder Referenzen auf Objekte (z.B. Arrays, Strings).

```
int     i = 8;
double  f = 0.2;
int[]   a = new int[3];
```



- **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

Visualisieren Sie folgendes Beispiel im Java Visualizer!

```
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

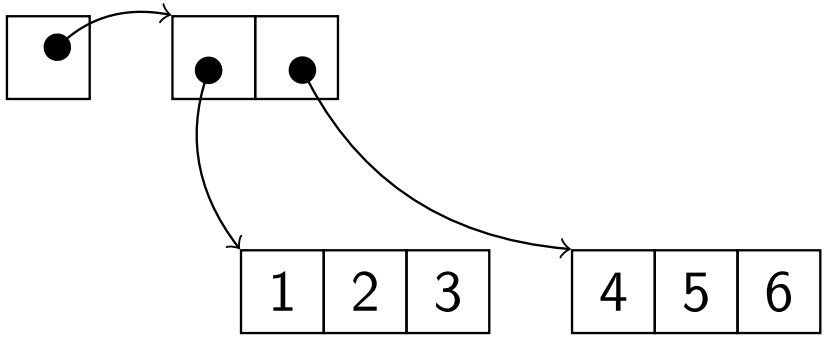
Visualisieren Sie folgendes Beispiel im Java Visualizer und verwenden Sie dabei die Option “ Show String/Integer/etc objects, not just values”!

```
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));
  }
}
```

Mehrdimensionale Arrays

Mehrdimensionale Arrays sind "Arrays von Arrays". Die Initialisierung erfolgt wie bei eindimensionalen Arrays durch Angabe der Anzahl der Elemente je Dimension.

```
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

Multiplikation zweier $n \times n$-Matrizen `a` und `b`

```
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];
```

Ungleichförmige Arrays

- Es ist 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.

```
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();
}
```

Fallstudien: Simulationen

Motivation

- 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.

Beispiel: Ruin des Spielers[1]

- Simulation von Gewinnwahrscheinlichkeiten beim Besuch von Spielbanken
- 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
- Faires Spiel (d.h. Gewinnwahrscheinlichkeit 50%)
- Das Spiel ist beendet, wenn der Spieler pleite ist oder wenn er den Zielbetrag erreicht hat

Frage

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

[1] Sedgewick & Wayne, S.87

```
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 * gewinne/versuche + "% Erfolg");
    System.out.println("Durchschnittl. Anzahl Wetten: "
      + wetten/versuche);
  }
}
```

Beobachtungen

- Die Erfolgswahrscheinlichkeit ist gegeben durch das Verhältnis von Einsatz zu Ziel.

  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 1 Million Versuche.

Frage

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

Beispiel: Sammelbilder[2]

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

- Karten werden als Array von n boolschen Werten repräsentiert, initialisiert mit `false`.
- Es wird nun eine Zufallszahl zwischen 0 und $n - 1$ erzeugt.
- Der Wert an der entsprechenden Arrayposition wird auf `true` gesetzt (falls er noch nicht `true` ist).
- Die einzelne Simulation endet, wenn alle Werte im Array `true` sind.
- Dabei wird gezählt, wie viele Versuche dazu notwendig waren.

[2] Sedgewick & Wayne, S. 121

```
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);
  }
}
```

Eingabe und Ausgabe

Eingabe und Ausgabe

- Interaktion und Kommunikation des Programms mit der Umgebung
- Bisher: Eingabe als Befehlszeilenparameter und Ausgabe auf Konsole
- Es gibt viele weitere Möglichkeiten Informationsschnittstellen zu gestalten
  - Grafik
  - Audio
  - Video
  - Drucker, etc...

Standardausgabe

- Abstrakter Zeichenstream unbegrenzter Größe, der mit dem Konsolenfenster verbunden ist
- Im Folgenden verwenden wir die Bibliothek `StdOut`

`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)

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

Standardeingabe

- 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*
- 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: `NumberFormatException`-Fehler

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
- Die Zeichenfolge **EOF** (End-of-File) gibt das Ende der Eingabe an (unter Linux: Zeilenumbruch gefolgt von Strg-D bzw. Strg-Z)

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);
  }
}
```

Anwendungsbeispiel:

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

Umleiten von Eingaben und Ausgaben

- Oft sind die Daten zu umfangreich für manuelle Eingabe oder sollen Ergebnisse zur späteren Verwendung gespeichert werden.

- Standardausgabe in Datei umleiten (hier: in Datei `data.txt`)

```
java Average > data.txt
```

- Aus einer Datei in die Standardeingabe umleiten (hier: aus Datei `data.txt`)

```
java Average < data.txt
```

Zusammenfassung

- Deklararieren, Initialisieren, Lesen und Schreiben von Arrays
- `for`-Schleifen
- Referenzvariablen
- Lesen und Schreiben auf Ein- und Ausgabeströme von Daten