TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa

AG Softech
FB Informatik
TU Kaiserslautern

Lernziele

- Verwendung von Arrays
  - Deklaration, Initialisierung, Lesen und Schreiben von Elementen
- Semantik von Referenzvariablen kennen

Arrays

Ein **Array (dt. Feld)** ist ein Behälter für eine feste Anzahl von Werten eines bestimmten Typs.

Die Länge eines Arrays wird bei dessen Erstellung festgelegt und kann danach nicht mehr geändert werden.

Auf die Komponenten (auch: Elemente) eines Arrays wird über Indizes zugegriffen.

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 numerischen Datentypen: 0 bzw. 0.0.
- Für `boolean`: `false`
- Für `String`s und andere nicht-primitive Typen: `null`

Programmieren mit Arrays

Sei `exp` ein Ausdruck vom Typ `int`, 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,
  z.B. weist `a[exp] = 7;` der k-ten Arraykompente den Wert 7 zu.

`Ausdruck` → `Ausdruck` `.` `length`

`Ausdruck` → `Ausdruck` `[` `Ausdruck` `]`

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

Indizierung

- 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

Beispiel: Summe aller Einträge im Array

```
int[] ar = new int[5];
ar[0] = 6;
ar[1] = 2;
ar[2] = 8;
ar[3] = 3;
ar[4] = 2;

int sum = 0;
int i = 0;
while (i < ar.length) {
  sum = sum + ar[i];
  i = i + 1;
}

System.out.println(sum);
```

Beispiel: Initialisierung mit 1

```
int n = ...;
double[] a = new double[n];
// Array a ist jetzt mit 0 initialisiert

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

Einschub: for-Anweisung (Zählanweisung)

Anweisung → for (forInit; Ausdruck; forUpdate) Anweisung

forInit → Typ ≪ *Bezeichner* ≫ = Ausdruck

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

- forInit (deklariert und) initialisiert eine Zählvariable.
- Die Updateanweisung forUpdate ist entweder
  - eine Zuweisung
  - das Inkrementieren einer Variable durch den ++ Operator
  - das Dekrementieren einer Variable durch den -- Operator

Semantik der for-Anweisung

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

forInit → Typ ≪ Bezeichner ≫ = Ausdruck

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

- 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. Danach wird die Updateanweisung ausgeführt.
- Die Auswertung wiederholt sich nun ab ($*$).

Beispiel: Summe aller Einträge im Array

```
int[] ar = new int[5];
ar[0] = 6;
ar[1] = 2;
ar[2] = 8;
ar[3] = 3;
ar[4] = 2;

int sum = 0;
int i = 0;
while (i < ar.length) {
  sum = sum + ar[i];
  i = i + 1;
}

System.out.println(sum);
```

Beispiel: Summe aller Einträge im Array

```
int[] ar = new int[5];
ar[0] = 6;
ar[1] = 2;
ar[2] = 8;
ar[3] = 3;
ar[4] = 2;


int sum = 0;
for (int i = 0; i < ar.length; i++) {
  sum = sum + ar[i];
}


System.out.println(sum);
```

Beispiel: Vektoren als Arrays

```
// 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: Maximales Element eines Arrays

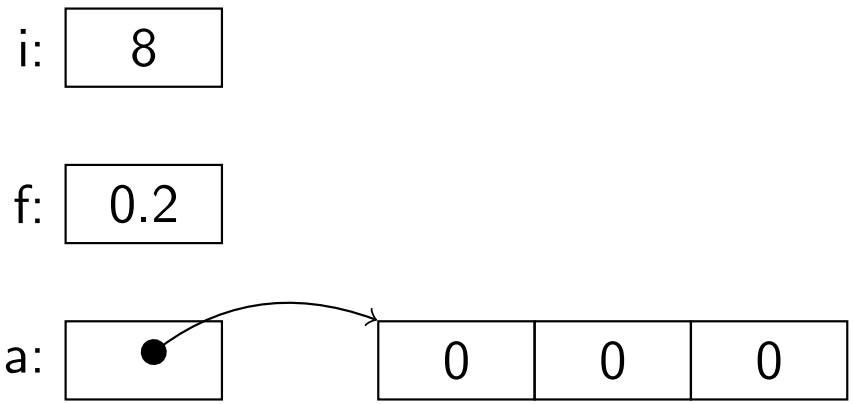
```
double[] a = new double[35];
... // Initialisierung von a mit beliebigen Werten

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

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.

Null-Referenz

- Literal `null` repräsentiert Referenz, die auf nichts verweist
- Kann nicht derefenziert werden
- Lesen bzw. Schreiben liefert `NullPointerException`

Frage

Was ist in den Variablen `a`-`d` gespeichert?

```
String[] a = null;
String[] b = new String[0];
String[] c = new String[1];
c[0] = null;
String[] d = new String[1];
d[0] = "";
```

NullPointerException

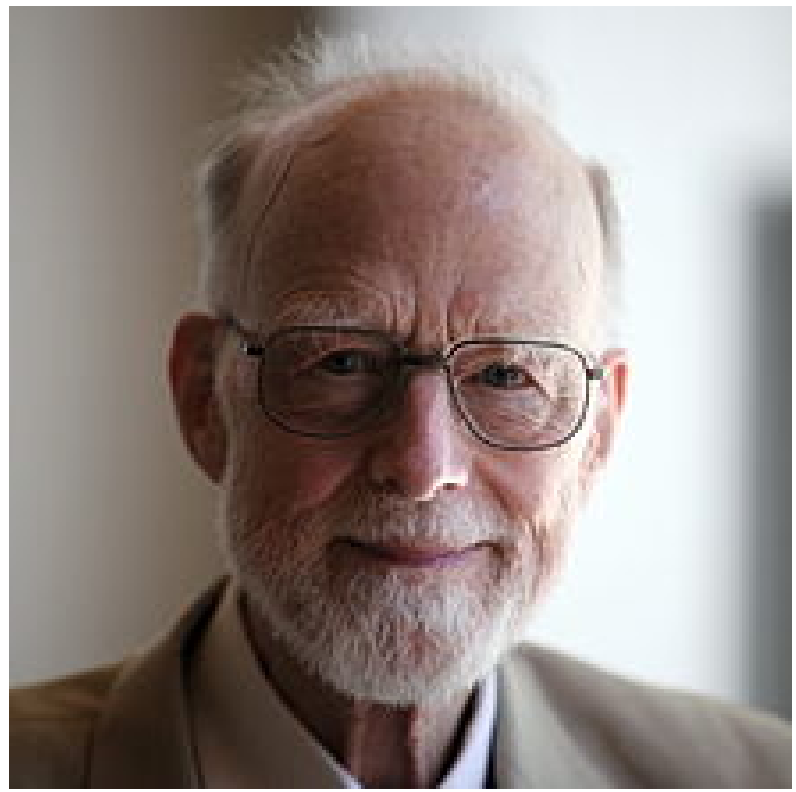
```
String[] a = null;
String[] b = new String[10];
...
```

Frage: Welche Anweisungen führen zu `NullPointerException`?

1. `a[0] = "Hallo";`
2. `b = a;`
3. `a = b;`
4. `a = new String[5];`
5. `b = new String[5];`
6. `System.out.println(a[0]);`
7. `System.out.println(b[0]);`
8. `System.out.println(a.length);`

"Null References: The Billion Dollar Mistake" [QCON 2009]

It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.



Sir Tony Hoare

Beispiel: Vergleichen und Kopieren von Arrays

```
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

```
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));
  }
}
```

Zusammenfassung: Arrays und Referenzvariablen

Arrays:

- Typen: `int[]`, `double[]`, `String[]`, etc.
- Erstellen: `new int[n]`
- Größe: `ar.length`
- Zugriff: `ar[i]`
- Iterieren: `for (int i = 0; i < ar.length; i++) { ... ar[i] ... }`

Referenzvariablen:

- Unterscheidung: Referenz und referenziertes Objekt
- Referenz-Werte vergleichen / Inhalte vergleichen
- `null`-Referenz
- Alias: Mehrere Referenzen auf das gleiche Objekt
- Kopien erstellen
- Größe von Arrays unveränderbar, aber Referenz kann geändert werden

Beispiel: Berechnung von Primzahlen

Sieb des Eratosthenes (Primzahlen zwischen 2 und n)

1. Zunächst werden alle Zahlen $2, 3, 4, \ldots$ bis n aufgeschrieben.
2. Die zunächst unmarkierten Zahlen sind alles potentielle Primzahlen.
3. Die kleinste unmarkierte Zahl in diesem Verfahren ist immer eine Primzahl.
4. Wenn eine Primzahl gefunden wird, werden alle Vielfachen dieser Primzahl als Nicht-Primzahlen markiert.
5. Für den Algorithmus bestimmt man jeweils die nächste nichtmarkierte Zahl.
6. 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.
7. Folglich muss es sich um eine Primzahl handeln.
8. Diese wird dementsprechend als Primzahl ausgegeben.
9. Im nächsten Schritt streicht man alle Vielfachen dieser Zahl und führt das Verfahren fort, bis man am Ende der Liste angekommen ist.

Optimierungen: Sieb des Eratosthenes

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.

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

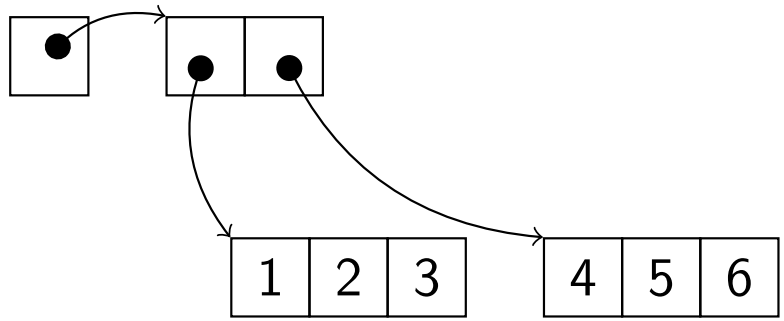
```
public class EratosthenesOptimized {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    boolean[] isPrime = new boolean[n+1];
    ... // Initialisierung

    // 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 + " ");
        }
    }
  }
}
```

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.
- Gefahr von `ArrayIndexOutOfBounds`-Fehler

```
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();
}
```

Simulationen

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(100.0 * gewinne/versuche + "% Erfolg");
    System.out.println("Durchschnittl. Anzahl Wetten: "
      + ((double) 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?

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 ($\Rightarrow$ 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 Mittelwert > data.txt
```

- 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

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