

Arrays

Einführung in die Programmierung 1

Sommersemester 21

TU Wien

Überblick

- Eindimensionale Arrays
- Referenzen
- Mehrdimensionale Arrays
- Methoden und Arrays
- Rekursion und Arrays
- Hilfsklasse Arrays

Eindimensionale Arrays

Motivation

- Bisher
 - Eine Variable hat einen Wert
- Problem
 - Wir möchten z. B. 1000 Messwerte abspeichern und bearbeiten
 - 1000 Variablen?
- Lösung
 - Eine Struktur, die das Verwalten vieler Werte erleichtert

Array

- Ein Array ist eine Zusammenfassung gleichartiger Elemente
- Ein Array ist eine Datenstruktur
- Datenstrukturen
 - Legen größere Mengen an Daten strukturiert ab
 - Unterstützen den gezielten Zugriff auf die Daten

Arrays deklarieren

- Form: `Datentyp[] Name`
- Beispiel: `int[] number;`
- Achtung
 - Legt nur fest, dass `number` auf ein Array von ganzen Zahlen (ein `int`-Array) zeigt
 - Es wird noch **keine Größe** angegeben

Anlegen von Arrays

- Anlegen bei Deklaration

```
int[] arr = new int[10];
```

- Späteres Anlegen

```
int[] arr;
```

```
...
```

```
arr = new int[10];
```

new-Operator

- Bei Aufruf des new-Operators wird Speicher angefordert (abhängig von der Größe)
- Es wird ein Array-Objekt erzeugt
- Größe des Arrays
 - Ist ein ganzzahliger Ausdruck (kann auch berechnet werden)

Arrays – Initialisierung

- Alle Array-Elemente (Array-Einträge) werden automatisch initialisiert
 - 0 bei int (auch bei byte und short)
 - 0L bei long
 - 0.0f bei float
 - 0.0d bei double
 - '\u0000' bei char
 - false bei boolean
 - null bei String (und anderen Referenztypen)

Anlegen mit eigener Initialisierung

- Beispiel

```
int[] arr = new int[]{3, 4, 5, 6, 7};
```

```
// oder
```

```
int[] arr = {3, 4, 5, 6, 7};
```

- Array hat die Länge 5
- Array enthält die Elemente 3, 4, 5, 6, 7

Arrays – Index

- Jedem Element eines Arrays ist ein Index vom Typ `int` zugewiesen
- Indexzählung beginnt bei **0**
- Indexzählung geht bis **(Länge des Arrays) - 1**

Beispiel für Index

- Beispiel

```
int[] arr = new int[10];
```

- Schematisch (nach dem Anlegen)

Index	0	1	2	3	4	5	6	7	8	9
Inhalt	0	0	0	0	0	0	0	0	0	0

- Größe des Arrays

- Kann mit `arr.length` abgefragt werden (z. B. 10 im obigen Beispiel)

Zugriff auf einzelne Elemente

- Beispiel

```
int[] array = new int[5];
```

- Zugriff

- Indexwert in eckigen Klammern (z. B. array[3])
- Kann wie Variable verwendet werden

- Beispiel

```
int[] array = new int[5];  
array[0] = 2;  
array[1] = array[0] + 3;  
for (int i = 0; i < array.length; i++) {  
    System.out.println(array[i]);  
}
```

2
5
0
0
0

Array-Index

- Der Index kann auch ein beliebiger Ausdruck sein
 - Der Typ des Indexausdrucks muss aber int (bzw. short, byte oder char) sein
- Beispiel

```
int[] array = new int[5];
int n = array.length;
array[0] = 2;
array[1] = array[0] + 3;
array[2]++;
array[3] = array[n - 3];
array[4] = array[array[2]];
for (int i = 0; i < n; i++) {
    System.out.println(array[i]);
}
```

2
5
1
1
5

Indexfehler

- Indexwert muss gültig (im Bereich des Arrays) sein
- Sonst **Laufzeitfehler**, d. h. das Programm wird während der Ausführung **sofort unterbrochen**
- Programm wirft eine **ArrayIndexOutOfBoundsException**
- Hinweis
 - **Laufzeit \neq Übersetzungszeit**
 - Ausnahmen (Exceptions) werden zur Laufzeit geworfen

Indexfehler – Beispiel

- Beispiel

```
int[] array = new int[5];  
array[5] = 1;
```

- Ausgabe in IntelliJ bei Ausführung

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5  
    at Tester.main ...
```


Beispiel (Operationen auf Arrays)

```
public class ArrayTest {  
  
    public static void main(String[] args) {  
        int n = 5;  
        double[] data = new double[n];  
        double sum = 0.0;  
        for (int i = 0; i < n; i++) {  
            data[i] = Math.random() * 100;  
        }  
        for (int i = 0; i < n; i++) {  
            sum += data[i];  
        }  
        double average = sum/n;  
        for (int i = 0; i < n; i++) {  
            System.out.println("Element at position " + i + ": " + data[i]);  
        }  
        System.out.println("Sum: " + sum);  
        System.out.println("Average: " + average);  
    }  
}
```

Element at position 0: 71.13847532247067
Element at position 1: 40.372869246156554
Element at position 2: 68.0432110287555
Element at position 3: 9.694335299857093
Element at position 4: 45.36966934719746
Sum: 234.61856024443728
Average: 46.92371204888745

„Initialisieren“ des Arrays
mit zufälligen Werten

Beispiel (Inhalt eines Arrays umdrehen)

```
public class ArrayTest2 {  
  
    public static void main(String[] args) {  
        int n = 10;  
        int[] data = new int[n];  
        for (int i = 0; i < n; i++) {  
            data[i] = (int) (Math.random() * 100);  
        }  
        for (int i = 0; i < n; i++) {  
            System.out.print(data[i] + "\t");  
        }  
        System.out.println();  
        for (int i = 0; i < n/2; i++) {  
            int temp = data[i];  
            data[i] = data[n-i-1];  
            data[n-i-1] = temp;  
        }  
        for (int i = 0; i < n; i++) {  
            System.out.print(data[i] + "\t");  
        }  
    }  
}
```

26	37	28	54	72	50	64	42	31	8
8	31	42	64	50	72	54	28	37	26

Beispiel (Primzahlsuche, Sieb des Eratosthenes)

- Aufgabe
 - Bestimmen aller Primzahlen bis zu einer vorgegebenen Zahl
- Vorgehen
 - Alle Zahlen 2, 3, 4, ... bis zu einem Maximalwert n werden betrachtet
 - Die kleinste unmarkierte Zahl ist immer eine Primzahl
 - Wurde eine Primzahl gefunden, werden alle Vielfachen dieser Primzahl als zusammengesetzt markiert
 - Es wird die nächstgrößere nicht markierte Zahl bestimmt
 - Es werden wieder alle Vielfachen gestrichen und das Verfahren fortgeführt

Beispiel (Primzahlsuche, Sieb des Eratosthenes)

- Beispiel
 - Ausgangslage
 - 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 - Alle Vielfachen von 2 eliminieren (hier ausgegraut)
 - 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 - Alle Vielfachen von 3 eliminieren (hier ausgegraut)
 - 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 - Alle Vielfachen von 5 eliminieren (hier ausgegraut)
 - 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
- Code in TUWEL (Lösung mit boolean-Array)
 - Sieve.java

Iterator-Form der for-Schleife

- Ausgabe mit herkömmlicher for-Schleife

```
int[] arr = {3, 4, 5, 6, 7};  
for (int a = 0; a < arr.length; a++)  
    System.out.println(arr[a]);
```

- Mit der Iterator-Form (foreach-Schleife)

```
int[] arr = {3, 4, 5, 6, 7};  
for (int i : arr)  
    System.out.println(i);
```

- Schleifenrumpf wird für jedes Element des Arrays ausgeführt
- Elementvariable (im Beispiel i)
 - Erhält den Wert des jeweils nächsten Array-Elements
 - Kann verändert werden, der Array-Inhalt bleibt aber gleich
- Nur kompletter Durchlauf (aufsteigend)

char-Arrays und Strings

- Beide speichern eine Liste von Zeichen, z. B.

```
char[] c = {'H', 'e', 'l', 'l', 'o'};  
String s = "Hello";
```

- Einige Unterschiede
 - Ein String ist unveränderlich, ein char-Array nicht
 - Strings können mit + verkettet werden, char-Arrays nicht
 - String kann einfacher erzeugt werden (siehe obiges Beispiel)

Array von Strings

- Beispiel mit Belegung und Ausgabe

```
String[] ts = new String[]{ "Hello", "World", "Bye" };  
for (String t : ts) {  
    System.out.println(t);  
}
```

Hello
World
Bye

- Beispiel mit Standardinitialisierung

```
String[] ts = new String[3];  
for (String t : ts) {  
    System.out.println(t);  
}
```

null
null
null

Array von Strings – null-Wert

- null-Wert kann ausgegeben werden
- null-Wert kann aber nicht bei einem Aufruf einer Methode verwendet werden!
 - Ausnahme (**NullPointerException**) wird geworfen

```
String[] ts = new String[3];  
for (String t : ts) {  
    System.out.println(t.length());  
}
```

Erzeugt Ausnahme!
Methodenaufruf erfordert einen gültigen String!

Array von Strings – Überprüfungen

- Sicherstellen, dass Elemente Strings sind
- Wenn Array möglicherweise nicht mit Strings belegt ist
 - Auf null überprüfen

```
for (String t : ts) {  
    if (t != null) {  
        System.out.println(t.length());  
    }  
}
```

Referenzen

Zuweisung von Arrays

- Array-Variable ist eine **Referenz** auf das eigentliche Array
- Einer Array-Variable vom Typ x kann immer nur ein Array vom Typ x zugewiesen werden
 - Länge kann aber variieren
- **Es wird nur die Adresse kopiert, nicht der Inhalt**

Elementare Typen – Referenztypen

- Beispiel `int`-Variable (schematisch)
 - `int x = 1;`



- Beispiel `int`-Array (schematisch)
 - `int[] arr = {3, 4, 5};`



Zuweisung bei Array-Variablen (1)

- Wie lautet die Ausgabe?

```
int[] x = new int[10], y;
```

```
y = x;
```

```
y[1] = 7;
```

```
System.out.println(y[0] + " " + x[0]);
```

```
System.out.println(y[1] + " " + x[1]);
```

- Wie entsteht diese Ausgabe?

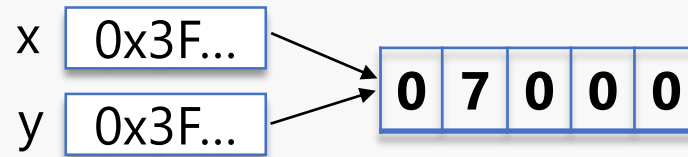


```
0 0  
7 7
```

Zuweisung bei Array-Variablen (2)

- Erklärung

- x und y sind Array-Variablen
- x und y zeigen auf den gleichen Speicherbereich
 - x und y sind Aliase
- Die Zuweisung bei y[1] verändert auch x[1]

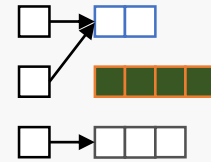
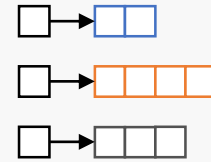


Freigabe des Array-Speichers

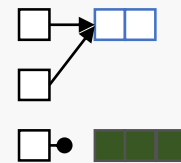
- Erfolgt durch das System
 - Speicher muss nicht explizit freigegeben werden
 - Nicht benutzter Speicher wird zur Laufzeit automatisch eingesammelt
 - Durch den Garbage-Collector
- **Voraussetzung für das Einsammeln**
 - **Array wird nicht mehr referenziert**
- Erreichbarkeit unterbinden
 - Einer Array-Variable ein anderes Array zuweisen
 - Einer Array-Variable den Wert `null` zuweisen
 - `x = null;`

Beispiel

```
private static void test() {  
    int[] a = new int[2];  
    int[] b = new int[4];  
    int[] c = new int[3];  
  
    ...  
  
    ...  
  
    b = a;  
  
    ...  
  
    c = null;  
  
    ...  
  
}
```



Keine Referenz auf das Array mehr – es kann eingesammelt werden



Keine Referenz auf das Array mehr – es kann eingesammelt werden



Am Ende der Methode werden alle lokalen Variablen freigegeben – Referenzen fallen weg

Kopieren von Arrays

- Echte Kopie erzeugen

- Beispiel

```
int[] original = {3, 4, 5, 6, 7};  
int[] copy = new int[original.length];  
for(int i = 0; i < original.length; i++) {  
    copy[i] = original[i];  
}  
copy[0] = 8;  
System.out.println(original[0]);
```

Elementweise
kopieren

Hier wird 3 ausgegeben

- Methode `System.arraycopy`

- Effiziente Methode
 - Kopieren von Abschnitten bzw. ganzen Arrays
 - API genau lesen!

Beispiel (System.arraycopy verwenden)

```
public class ArrayCopy {  
  
    public static void main(String[] args) {  
        int[] original = {3, 4, 5, 6, 7};  
        int[] copy = new int[original.length];  
        System.arraycopy(original, 0, copy, 0, original.length);  
        for (int i = 0; i < original.length; i++) {  
            System.out.print(copy[i] + " ");  
        }  
        System.out.println();  
        System.arraycopy(original, 0, original, 2, 3);  
        for (int i = 0; i < original.length; i++) {  
            System.out.print(original[i] + " ");  
        }  
    }  
}
```

Elementweise kopieren

Abschnitt (die ersten 3 Elemente) ab Position 2 in das Array selbst kopieren

3	4	5	6	7
3	4	3	4	5

Vergleiche bei Arrays

- Operator == bei Arrays meist nicht geeignet
 - Vergleicht nur, ob Arrays **die gleichen Referenzen** haben
 - Vergleicht **nicht den Inhalt** der Arrays
- Gilt auch für !=

Identität und Gleichheit

- Identität
 - Zwei Werte bzw. Objekte sind identisch, wenn die Anwendung von „==“ true liefert
- Gleichheit
 - Wenn auch referenzierte Objekte in den Vergleich miteinbezogen werden
- Unterscheidung
 - Gleiche Werte von elementaren Typen (z. B. int) sind immer identisch
 - Identische Objekte (z. B. Arrays) sind stets gleich
 - Gleiche Objekte (z. B. Arrays mit gleichem Inhalt) müssen aber nicht identisch sein

Inhalt zweier Arrays vergleichen

- Elemente paarweise vergleichen

```
int[] original = ...;
int[] copy = ...;

...
boolean eq = (original.length == copy.length);
for (int i = 0; eq && i < original.length; i++) {
    eq = (original[i] == copy[i]);
}
System.out.println(eq ? "equal" : "not equal");
```

- Obige Lösung funktioniert nur bei eindimensionalen Arrays
- Hinweis: In der Praxis gibt es dafür Bibliotheksmethoden (siehe Abschnitt über Hilfsklasse Arrays)!

Gleichheit bei Strings

- Gleichheit

- Zwei Zeichenketten sind gleich, wenn sie die gleichen Längen haben und die gleichen Zeichen in gleicher Reihenfolge enthalten
- Methode `equals` für Vergleich benutzen

- Beispiele

```
System.out.println("ab".equals("ab")); // true
System.out.println("ab".equals("ba")); // false
String s = "ab", t = "cd";
String u = "ab", v = t;
System.out.println(s.equals(u)); // true
System.out.println(s == u); // true
System.out.println(t.equals(v)); // true
System.out.println(t == v); // true
System.out.println((s+t).equals(u+v)); // true
System.out.println((s+t) == (u+v)); // false
```

Unterschied zu Arrays!
Compiler optimiert, d. h.
das String-Literal „ab“
gibt es nur einmal!

Zur Laufzeit keine
Optimierung!

Mehrdimensionale Arrays

Mehrdimensionale Arrays (1)

- Arrays können auch mehrere Dimensionen haben

- Beispiel: 2-dimensionales Array – Matrix

```
int[][] matrix = new int[3][3];
```

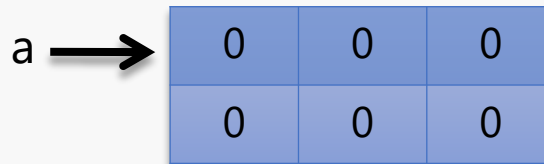
```
int[][] matrix2 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

- Beispiel für Zugriff

```
x = matrix2[1][2];
```

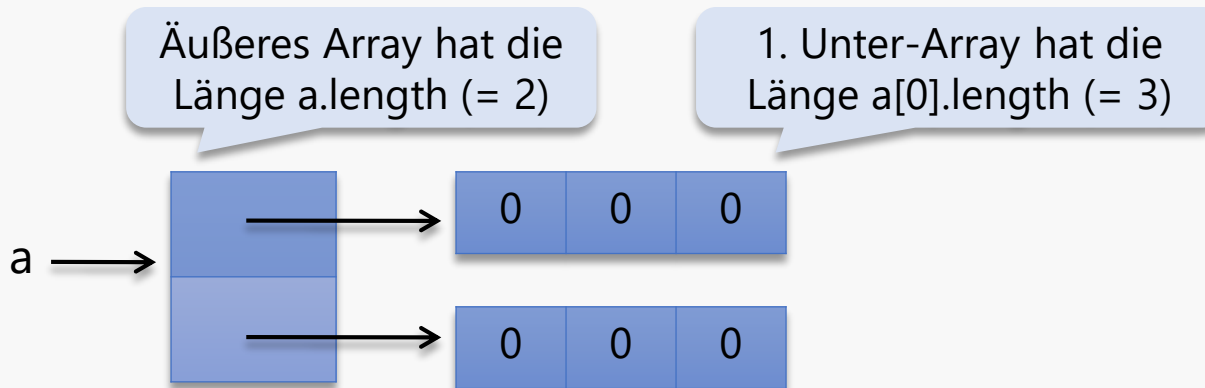
- Konzeptionelle Sicht

- z. B. `int[][] a = new int[2][3];`



Mehrdimensionale Arrays (2)

- Realität in Java
 - z. B. `int[][] a = new int[2][3];`



- Eindimensionale Arrays enthalten Elemente vom Basistyp
- Mehrdimensionale Arrays enthalten weitere Arrays

Beispiel (2D-Array erzeugen und ausgeben)

7 8 3
4 2 7
7 8 3
4 2 7

```
public class Array2D {  
    public static void main(String[] args) {  
        int[][] a = new int[2][3];  
        for (int i = 0; i < a.length; i++) {  
            for (int j = 0; j < a[i].length; j++) {  
                a[i][j] = (int) (Math.random() * 10);  
            }  
        }  
        for (int i = 0; i < a.length; i++) {  
            for (int j = 0; j < a[i].length; j++) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        for (int[] i : a) {  
            for (int j : i) {  
                System.out.print(j + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

Ausgabe mit herkömmlichen
for-Schleifen

Ausgabe mit
foreach-Schleifen

Weitere Aspekte (1)

- Einzelne Dimensionen können offen bleiben (nur am Ende) und später initialisiert werden

```
int[][] arr1 = new int[2][];  
for (int i = 0; i < arr1.length; i++)  
    arr1[i] = new int [5];
```

Weitere Aspekte (2)

- Einzelne Dimensionen können auch unterschiedliche Längen haben

```
int[][] arr1 = new int[2][];  
for (int i = 0; i < arr1.length; i++)  
    arr1[i] = new int[i + 2];
```

Beispiel (Pascalsches Dreieck)

```
import java.util.Scanner;

public class PascalTriangle {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Please enter a positive number: ");
        int n = sc.nextInt();
        int p[][] = new int[n][];
        for (int i = 0; i < p.length; i++) {
            p[i] = new int[i + 1];
            for (int j = 0; j <= i; j++) {
                if ((j == 0) || (j == i)) {
                    p[i][j] = 1;
                } else {
                    p[i][j] = p[i - 1][j - 1] + p[i - 1][j];
                }
                System.out.printf("%2d ", p[i][j]);
            }
            System.out.println();
        }
    }
}
```

Die Methode printf ermöglicht eine Ausgabe mit speziellen Formatierungen (ähnlich zu C)

Beispiel (Pascalsches Dreieck, 7 Stufen)

Please enter a positive number: 7

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

Array p vor dem
Betreten der
äußeren for-Schleife

null
null
null
null
null
null
null

**Nach 3
Durchläufen**

→	1		
→	1	1	
→	1	2	1
	null		
	null		
	null		
	null		

Beispiel (verschieden lange „Zeilen“)

```
public class JaggedArrayTest {  
  
    public static void main(String[] args) {  
        int n = 5;  
        int[][] testArray = new int[n][];  
        int counter = 0;  
        for (int i = 0; i < testArray.length; i++) {  
            testArray[i] = new int[(int) (Math.random() * n)];  
            for (int j = 0; j < testArray[i].length; j++) {  
                testArray[i][j] = counter++;  
            }  
        }  
        for (int[] line : testArray) {  
            for (int number : line) {  
                System.out.print(number + "\t");  
            }  
            System.out.println();  
        }  
    }  
}
```

0	1		
2	3	4	
5	6	7	8
9	10		
11	12	13	

0		
1	2	3
4		
5	6	
7	8	9

Belegung (und Länge der Zeilen) ändert sich bei jeder Ausführung

Methoden und Arrays

Arrays als Parameter

- Übergabe
 - Es wird die **Referenz** kopiert
- Auswirkung
 - **Änderungen am Inhalt** sind **außerhalb** der Methode sichtbar
 - Änderungen am formalen Parameter selbst sind nicht außerhalb der Methode sichtbar

Beispiel (Array als Parameter)

```
public class CallTest {
```

```
    private static void test(int[] a) {
```

```
        int[] x = new int[3];
```

```
        a[1] = 3;
```

```
        a = x;
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        int[] t = {5, 6, 7};
```

```
        System.out.println(t[1]);
```

```
        test(t);
```

```
        System.out.println(t[1]);
```

```
    }
```

```
}
```

Parameter a zeigt auf den gleichen Speicherbereich wie t

Diese Änderung ist auch in main sichtbar

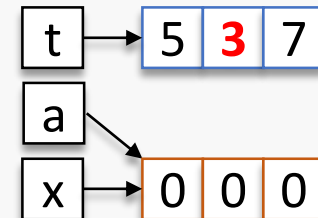
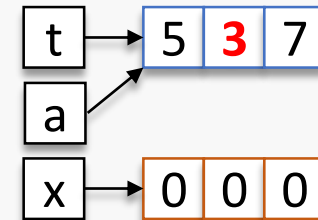
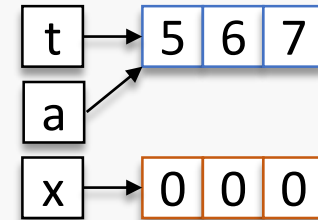
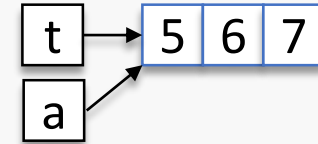
Diese Zuweisung hat keine Auswirkung nach außen

6

3

Beispiel (Array als Parameter, Visualisierung)

```
private static void test(int[] a) {  
  
    int[] x = new int[3];  
  
    a[1] = 3;  
  
    a = x;  
  
}
```



Arrays und return

- Eine Methode kann auch ein Array zurückliefern
- Beispiel

```
private static int[] m() {  
    ...  
    int[] a = new int[10];  
    ...  
    return a;  
}
```

- Hier wird am Ende der Methode eine Referenz zurückgegeben
- Muss außerhalb der Methode einer Array-Variablen zugewiesen werden
 - Mit dem Beenden der Methode wird die lokale Variable gelöscht
 - Nach der Zuweisung gibt es aber eine Referenz auf das Array von außen

Beispiel (ArrayInverter)

- Aufgabe
 - Methode `readArray`
 - Liest die Anzahl der Elemente im Array ein
 - Liest Elemente von der Kommandozeile ein und gibt das Array zurück
 - Methode `printArray`
 - Gibt die Elemente eines Arrays, das als Parameter übergeben wird, aus
 - Methode `invertArray`
 - Dreht den Inhalt eines Arrays, das als Parameter übergeben wird, um
 - Seiteneffekt!
 - Es wird kein neues Array erzeugt
- Code in TUWEL
 - `ArrayInverter.java`

Beispiel (MatrixMultiply)

- Aufgabe
 - Multiplikation zweier Matrizen (2-dimensionale Arrays)
 - Ergebnis als Matrix (2-dimensionales Array) zurückgeben
 - Rückgabe von null-Referenz
 - Null-Werte als Argumente oder als Array-Einträge
 - Matrizen lassen sich nicht multiplizieren (Dimensionen passen nicht)
- Code in TUWEL
 - `MatrixMultiply.java`

Beispiel (String-Arrays als Parameter)

- Beispiel
 - Vergleich von zwei String-Arrays in einer eigenen Methode
 - Test der Implementierung in der main-Methode
- Code in TUWEL
 - `StringArraysTester.java`

Kommandozeilenargumente

- main-Methode hat als Parameter ein Array von Strings
- Argumente übergeben
 - Kommandozeile: Nach dem Aufruf des Programms
 - Beispiel: `java ParameterTest Hello 1 2`
 - IntelliJ: Run -> Edit Configurations -> Programm arguments
- Ausgabe der Argumente

```
public static void main(String[] args) {  
    for (String s : args) {  
        System.out.println(s);  
    }  
    ...  
}
```

Hello
1
2

Kommandozeilenargumente verarbeiten

- Bestimmte Methoden für das Auslesen
- Beispiel `parseInt` in der Klasse `Integer`
 - Versucht einen String in einen Integer umzuwandeln

`args[0]` entspricht dem ersten Kommandozeilenargument.

Allgemein: Wird nichts (oder zu wenige Argumente) übergeben, dann gibt es eine Ausnahme.

- Beispiel (bei Aufruf wird ein Integer übergeben)

```
int grade = Integer.parseInt(args[0]);
String text = " no grade ";
if (grade == 1) {
    text = " excellent ";
} else if (grade == 2) {
    text = " good ";
} else if (grade == 3) {
    text = " satisfactory ";
} else if (grade == 4) {
    text = " sufficient ";
} else if (grade == 5) {
    text = " insufficient ";
}
System.out.println("Your grade:" + text);
```

Methoden mit variabler Parameteranzahl

- Manchmal wird eine beliebige Anzahl von Parametern eines Typs benötigt
- Herkömmliche Lösung mit einem Array

```
private static int sum(int[] values) { ... }
```
- varargs-Parameter

```
private static int sum(int... values){ ... }
```

 - Aus dem varargs-Parameter wird automatisch ein Array erzeugt
 - Der varargs-Parameter muss immer an der letzten Stelle der Parameterliste stehen
- Beispiel für Verwendung von varargs in TUWEL
 - VarargsTest.java

Rekursion und Arrays

Beispiel (rekursiv Array-Inhalte ausgeben)

```
public class RecursiveArrayOutputTest {
```

```
// arr.Length > 0 && 0 <= left < arr.Length && 0 <= right < arr.Length
```

```
private static void printArrayRecursive(int[] array, int left, int right) {
```

```
    if (left <= right) {
```

```
        System.out.print(array[left] + " ");
```

```
        printArrayRecursive(array, left + 1, right);
```

```
    }
```

```
}
```

Basisfall (keine Ausgabe) ist hier implizit

```
public static void main(String[] args) {
```

```
    int[] x = new int[]{1, 5, 4, 2, 6, 8, 0, 7};
```

```
    printArrayRecursive(x, 0, x.length - 1);
```

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

```
    printArrayRecursive(x, 2, 5);
```

```
}
```

```
}
```

1 5 4 2 6 8 0 7
4 2 6 8

Beispiel (rekursiv maximale Differenz finden)

Diese Methode berechnet die maximale absolute Differenz zweier aufeinander folgender Elemente von array im Indexbereich 0 bis i (inklusive).

```
public class RecursiveMaximumDifference {  
  
    // array != null, array.length >= 1, 0 <= i < array.length  
    private static int findMaxDiff(int[] array, int i) {  
        if (i > 0) {  
            return Math.max(Math.abs(array[i - 1] - array[i]), findMaxDiff(array, i - 1));  
        }  
        return 0;  
    }  
  
    public static void main(String[] args) {  
        int[] arr = new int[]{3, 7, 1, 8, 2, 4, 0, 9};  
        System.out.println(findMaxDiff(arr, arr.length - 1));  
        System.out.println(findMaxDiff(arr, arr.length/2));  
        System.out.println(findMaxDiff(arr, arr.length/4));  
        System.out.println(findMaxDiff(new int[]{20}, 0));  
    }  
}
```

9
7
6
0

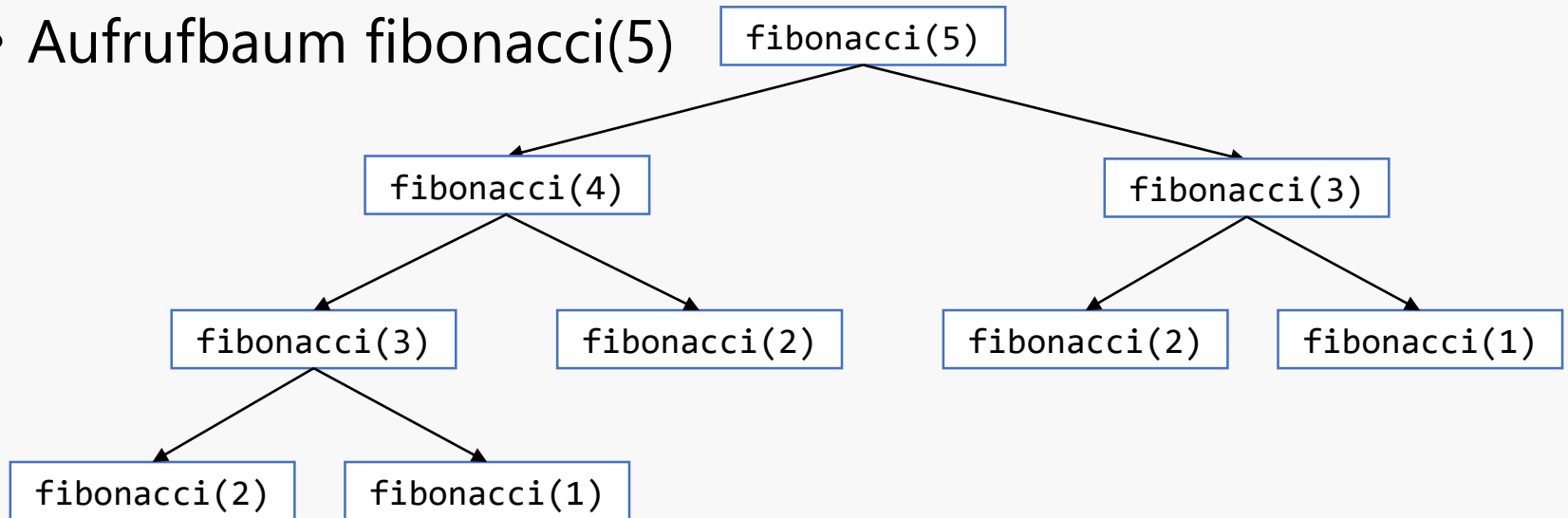
Memoisation

- Rückgabewerte von Funktionen werden zwischengespeichert
- Keine neue Berechnung, wenn Aufrufergebnis schon gespeichert

Fibonacci-Zahlen (Wiederholung)

```
private static long fibonacci(int n) {  
    if (n <= 2) {  
        return 1;  
    } else {  
        return fibonacci(n - 1) + fibonacci(n - 2);  
    }  
}
```

- Aufrufbaum fibonacci(5)



Beispiel (Memoisation)

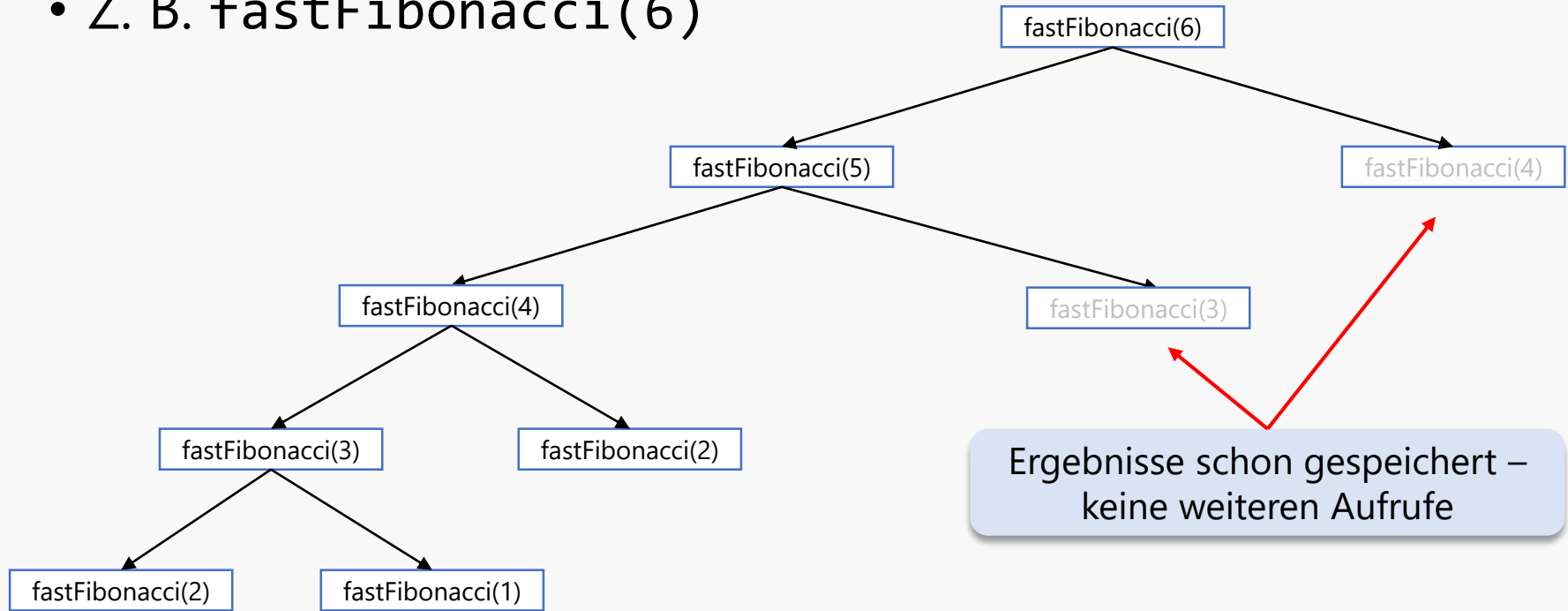
```
public class FibonacciWithMemoization {  
  
    private static long[] f = new long[93];  
  
    // 1 <= n <= 92  
    private static long fastFibonacci(int n) {  
        if (n <= 2) {  
            return 1;  
        } else if (f[n] == 0) {  
            f[n] = fastFibonacci(n - 1) + fastFibonacci(n - 2);  
        }  
        return f[n];  
    }  
  
    public static void main(String[] args) {  
        System.out.println(fastFibonacci(50));  
        System.out.println(fastFibonacci(90));  
        System.out.println(fastFibonacci(80));  
    }  
}
```

Nur bis 92 sinnvoll, danach
arithmetischer Überlauf

12586269025
2880067194370816120
23416728348467685

Memoisation (Anzahl der Aufrufe)

- Z. B. `fastFibonacci(6)`



Memoisation (Anzahl der Aufrufe)

- Wie viele Aufrufe finden bei unterschiedlichen Werten für n (abhängig von vorherigen Aufrufen) statt?

```
...  
public static void main(String[] args) {  
    System.out.println(fastFibonacci(50));  
    System.out.println(fastFibonacci(90));  
    System.out.println(fastFibonacci(80));  
}
```

...

fastFibonacci(50) -> 97 Aufrufe
fastFibonacci(90) -> 81 Aufrufe
fastFibonacci(80) -> 1 Aufruf

Hilfsklasse Arrays

Hilfsklasse Arrays

- Hilfsklasse für Arrays (muss importiert werden)
- Nützliche überladene Methoden (hier für `int`)
 - `static int binarySearch(int[] a, int key)`
 - Binäre Suche in einem Array
 - `static boolean equals(int[] a, int[] a2)`
 - Vergleicht den Inhalt von zwei Arrays
 - `static void fill(int[] a, int val)`
 - Befüllt alle Stellen eines Array mit einem bestimmten Wert
 - `static void sort(int[] a)`
 - Sortiert den Inhalt eines Arrays aufsteigend
 - `static String toString(int[] a)`
 - Liefert den Inhalt eines Arrays als formatierten String zurück

Beispiel (Methoden aus Arrays austesten)

```
import java.util.Arrays;

public class ArraysTest {
    public static void main(String[] args) {
        int[] x = {3, 5, 2, 7, 9, 4, 6, 1, 8};
        boolean[] y = new boolean[3], z = new boolean[3];
        Arrays.sort(x);
        // java.util.Arrays.sort(x); impliziter Import
        System.out.println(Arrays.toString(x));
        System.out.println(Arrays.binarySearch(x, 9));
        Arrays.fill(y, true);
        Arrays.fill(z, true);
        System.out.println(Arrays.toString(y));
        System.out.println((y == z));
        System.out.println(Arrays.equals(y, z));
        y = z;
        System.out.println(y == z);
    }
}
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
8
[true, true, true]
false
true
true
```

Beispiel (rekursiv maximale Differenz finden)

Alternative Variante ohne Indexwert – das Array wird bei jedem rekursiven Aufruf verkürzt (ohne das erste Element) neu erzeugt (mit Methode aus Arrays).

```
public class RecursiveMaximumDifference {
```

```
    // array != null
```

```
    private static int findMaxDiff(int[] array) {
```

```
        if (array.length > 1) {
```

```
            return Math.max(Math.abs(array[0] - array[1]),
```

```
                            findMaxDiff(Arrays.copyOfRange(array, 1, array.length)));
```

```
        }
```

```
        return 0;
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        int[] arr = new int[]{3, 7, 1, 8, 2, 4, 0, 9};
```

```
        System.out.println(findMaxDiff(arr));
```

```
    }
```

```
}
```

9

Zusammenfassung

- Eindimensionale Arrays
 - Deklaration, Erzeugung, Initialisierung, Index
- Referenzen
 - Zuweisung, Vergleich
- Mehrdimensionale Arrays
- Methoden und Arrays
- Rekursion und Arrays
- Hilfsklasse Arrays