

# **Praktische Informatik 1**

## **Konzepte benutzerdefinierter Datenstrukturen**

Thomas Röfer

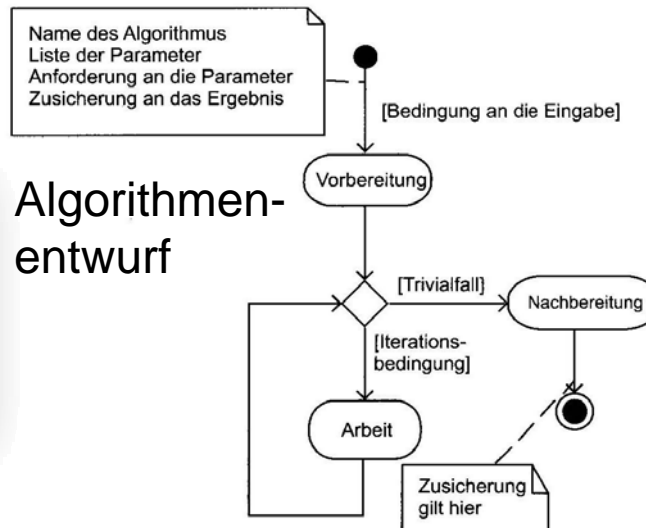
- Variablen
- Reihungen (Arrays)
- Zeichenketten (Strings)
- Verbunde (Klassen)
- Enthaltensein
- Entwicklungszyklus
- Musterlösung, Übungsblatt

# Rückblick „Abstrakte Algorithmen und Sprachkonzepte“

## Algorithmus

Spezifikation  
Durchführbarkeit  
Korrektheit

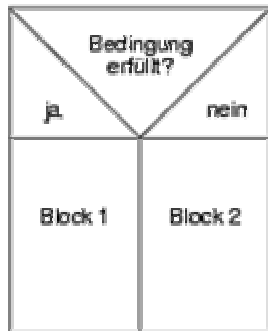
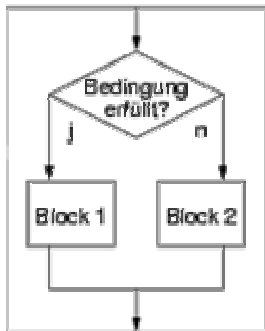
## Algorithmen-entwurf



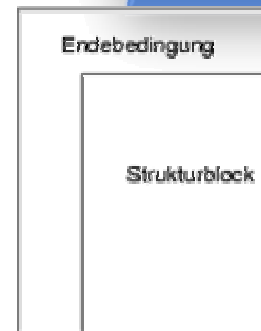
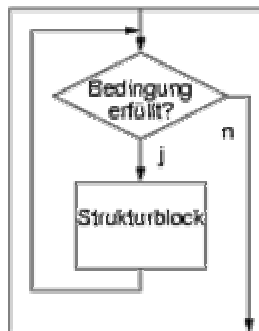
## Sprachkonstrukte

Datenspeicher/Zuweisungen  
Ausdrücke  
Sequenzierungsanweisungen

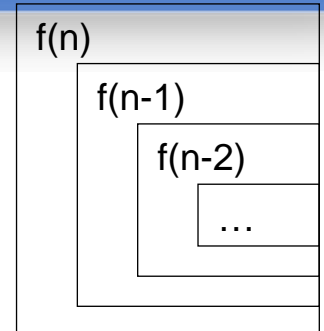
## Verzweigungen



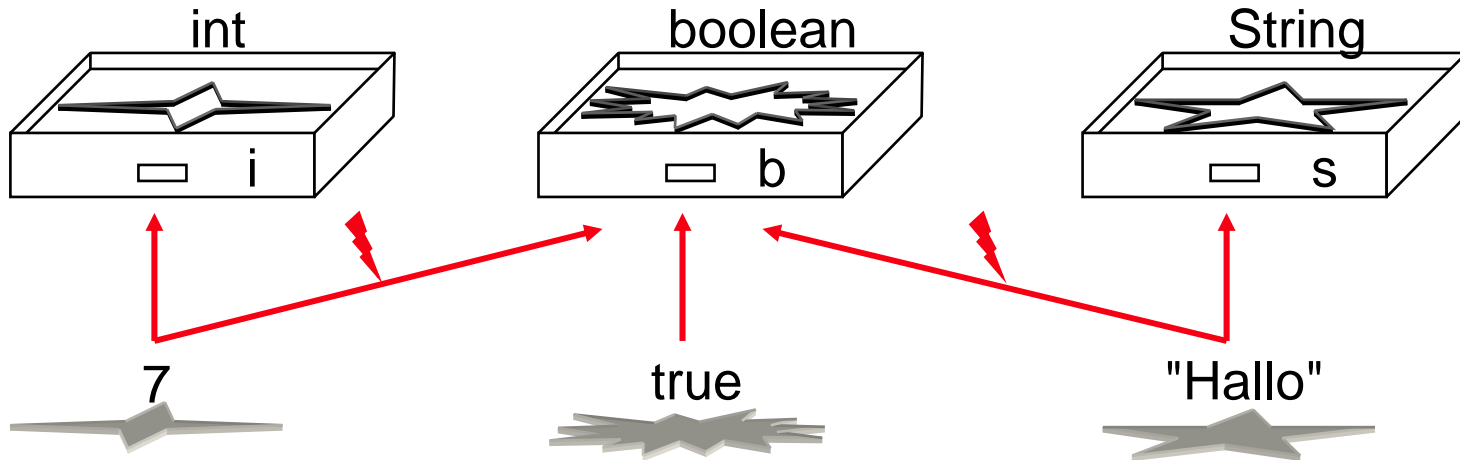
## Schleifen/Iteration



## Hilfskonstrukte



# Variablen



- Variablen haben

- einen Namen (i, b, s, ...)
- einen Typ (int, boolean, String, ...)
- einen Wert (7, true, "Hallo", ...)

- Richtig

- `int i = 7;`
- `boolean b = true;`
- `String s = "Hallo";`

- Falsch

- `boolean b = 7;`
- `boolean b = "Hallo";`

# Reihung (array)

- Definition

- Eine Reihung besteht aus einer festen Anzahl von Elementen gleichen Typs, auf die mit einem Index zugegriffen werden kann

- Beispiele

- |              |      |      |      |     |     |      |      |
|--------------|------|------|------|-----|-----|------|------|
| Zeitpunkt    | 1    | 2    | 3    | 4   | ... | 30   | 31   |
| Signalstärke | 10.5 | 10.5 | 12.2 | 9.8 | ... | 13.1 | 13.3 |

- Der Hauptspeicher eines Rechners ist eine Reihung

- Mathematische Interpretation

- Eine Reihung repräsentiert eine Funktion vom Indexbereich in den Wertebereich
- 1. Beispiel kann also als eine Funktion  $f: \mathbb{N} \rightarrow \mathbb{R}$  angesehen werden



# Reihungen in Java

0	Herbert Pappelbusch
1	Irene Schmidt
2	Claudia Miesmuffel
.	
.	
.	
.	
48	Tobias Kleinemann
49	Mareike Bumtrupp

```
String[] a = new String[50];  
a[0] = "Herbert Pappelbusch";  
a[1] = "Irene Schmidt";  
a[2] = "Claudia Miesmuffel";  
:  
a[48] = "Tobias Kleinemann";  
a[49] = "Mareike Bumtrupp";
```

```
String[] a = {  
    "Herbert Pappelbusch",  
    "Irene Schmidt",  
    "Claudia Miesmuffel",  
    :  
    "Tobias Kleinemann",  
    "Mareike Bumtrupp"  
};
```

# Zweidimensionale Reihungen

		2. Dimension →		
		0	1	2
1. Dimension ↓	0	Herbert	Pappelbusch	Herr
	1	Irene	Schmidt	Frau
	2	Claudia	Miesmuffel	Frau
	.			
	.			
	.			
	.			
	.			
	48	Tobias	Kleinemann	Herr
	49	Mareike	Bumtrupp	Frau

```
String[][] a = {
    {"Herbert", "Pappelbusch", "Herr"},
    {"Irene", "Schmidt", "Frau"},
    {"Claudia", "Miesmuffel", "Frau"},
    :
    {"Tobias", "Kleinemann", "Herr"},
    {"Mareike", "Bumtrupp", "Frau"}
};
```

```
String[][] a = new String[50][3];
a[0][0] = "Herbert";
a[0][1] = "Pappelbusch";
:
```

# Mehrdimensionale Reihungen

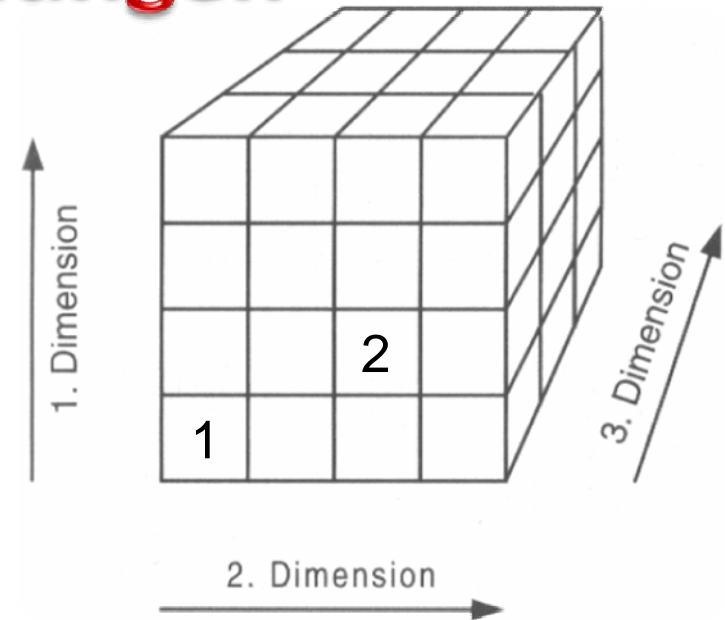
- Eine beliebige Anzahl von Dimensionen ist möglich

- Speicherplatzverbrauch bedenken!

- Flache Reihungen (C/C++)

- Mehrdimensionale Arrays sind eigentlich eindimensionale Arrays mit angepasster Index-Berechnung

- $a[i_1][i_2][i_3]$  entspricht  $a[0][0][((i_1 \cdot \text{length}_2) + i_2) \cdot \text{length}_3 + i_3]$



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

# Mehrdimensionale Reihungen

- In Java ist eine zweidimensionale Reihung eine Reihung von Reihungen
- Reihungen können in weiteren Dimensionen unterschiedliche Größen haben

Index	0	1	2	3
0	A	B	C	
1	D	E		
2	F	G	H	I

```
char[][] a = new char[3][];  
a[0] = new char[3];  
a[1] = new char[2];  
a[2] = new char[4];  
a[0][0] = 'A';  
:
```

```
char[][] a = {  
    {'A', 'B', 'C'},  
    {'D', 'E'},  
    {'F', 'G', 'H', 'I'},  
};  
System.out.println(a.length);    // 3  
System.out.println(a[2].length); // 4
```

# Eigenschaften von Reihungen

- Vorteil
  - Der Zugriff auf Elemente einer Reihung erfolgt in konstanter Zeit, ist also unabhängig von der Anzahl der Elemente in der Reihung
- Nachteil
  - Die Anzahl der Elemente einer Reihung ist konstant, d.h. eine Reihung kann nicht wachsen
- Tipps
  - Java stellt mit *java.util.Vector<E>* eine Klasse bereit, die dynamisch wachsende Arrays realisiert
  - Nur Funktionen durch Reihungen abbilden, die für die meisten Werte des Indexbereiches definiert sind, sonst ist der Speicherverbrauch zu hoch

# Zeichenkette (string)

- Spezielle Form der Reihung

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Zeichen	'H'	'a'	'l'	'l'	'o'	' '	'E'	'r'	's'	't'	's'	'e'	'm'	'e'	's'	't'	'e'	'r'

- Gebräuchliche Schreibweise: "Hallo Erstsemester"
- In Java
  - Möglich, aber umständlich und nicht weiter unterstützt:
    - `char[] c = {'H','a','l','l','o',' ','E','r','s','t','s','e','m','e','s','t','e','r'};`
  - Gebräuchlich:
    - `String s = "Hallo Erstsemester";`



# Zeichenketten in Java

- String ist ein Verbund (eine Klasse), die eine Zeichenreihung kapselt:
  - `char[] c = {'H','a','l','l','o',' ','E','r','s','t','s','e','m','e','s','t','e','r'};`
  - `String s = new String(c);`
- Die Klasse String bietet viele Methoden, z.B.
  - Länge: `int l = s.length(); // 18`
  - *n*-tes Zeichen: `char c = s.charAt(4); // 'o'`
  - Teil-String bilden
    - `String t = s.substring(1, 5); // "allo"`
    - `String u = s.substring(6); // "Erstsemester"`
  - Strings vergleichen
    - `boolean b = s.equals(t); // false`
    - `int r = s.compareTo(u); // 1 (-1: s < u, 0: s = u, 1: s > u)`
- Strings können addiert (verkettet) werden: `String a = "H" + t;`

# Verbund (record, struct, class)

- Definition

- Ein Verbund besteht aus Elementen möglicherweise unterschiedlichen Typs, auf die über ihren Namen zugegriffen werden kann

- Beispiel

- Stammdaten eines Beschäftigten (lt. Buch)
- Felder Name, Vorname,

Name	"Mustermann"
Vorname	"Martin"
GebTag	10
GebMonat	05
GebJahr	<u>1930</u>
Familienstand	"verheiratet"
...	...



# Verbunde (Klassen) in Java

Name	"Mustermann"
Vorname	"Martin"
GebTag	10
GebMonat	05
GebJahr	1930
Familienstand	"verheiratet"
...	...

```
class Stammdaten {  
    String name;  
    String vorname;  
    int gebTag;  
    int gebMonat;  
    int gebJahr;  
    String familienstand;  
}
```

```
Stammdaten s = new Stammdaten();  
s.name = "Mustermann";  
s.vorname = "Martin";  
s.gebTag = 10;  
:
```

# Konstruktoren für Klassen

```
class Stammdaten {  
    String name;  
    :  
    Stammdaten(String n, String v,  
        int t, int m, int j, String f) {  
        name = n;  
        vorname = v;  
        gebTag = t;  
        gebMonat = m;  
        gebJahr = j;  
        familienstand = f;  
    }  
}
```

- Ein Konstruktor initialisiert eine Instanz einer Klasse
- Ein Konstruktor hat keinen Rückgabetyp
- Es kann mehrere Konstruktoren pro Klasse geben,
  - die sich in den Parameterlisten unterscheiden
- Gibt man keinen Konstruktor an, generiert Java automatisch einen ohne Parameter
  - Z.B. Stammdaten()

```
Stammdaten s = new Stammdaten("Mustermann",  
    "Martin", 10, 5, 1930, "verheiratet");
```

# Verbund in Verbund

Name	Nachname	"Mustermann"
	Vorname	"Martin"
GebDatum	Tag	10
	Monat	05
	Jahr	1930
Familienstand	"verheiratet"	

```
Stammdaten s = new Stammdaten();  
s.name = new Name();  
s.gebDatum = new Datum();  
s.name.nachname = "Mustermann";  
s.name.vorname = "Martin";  
s.gebDatum.tag = 10;  
:
```

```
class Name {  
    String nachname;  
    String vorname;  
}
```

```
class Datum {  
    int tag;  
    int monat;  
    int jahr;  
}
```

```
class Stammdaten {  
    Name name;  
    Datum gebDatum;  
    String familienstand;  
}
```

# Nochmal mit Konstruktoren

```
class Name {  
    String nachname, vorname;  
    Name(String n, String v) {  
        nachname = n;  
        vorname = v;  
    }  
}
```

```
class Datum {  
    int tag, monat, jahr;  
    Datum(int t, int m, int j) {  
        tag = t;  
        monat = m;  
        jahr = j;  
    }  
}
```

```
class Stammdaten {  
    Name name;  
    Datum gebDatum;  
    String familienstand;  
  
    Stammdaten(Name n, Datum g, String f) {  
        name = n;  
        gebDatum = g;  
        familienstand = f;  
    }  
}
```

```
Stammdaten s = new Stammdaten(  
    new Name("Mustermann", "Martin"),  
    new Datum(10, 5, 1930),  
    "verheiratet");
```

# Mehrere Konstruktoren

```
class Stammdaten {  
    Name name;  
    Datum gebDatum;  
    String familienstand;  
    Stammdaten(Name n, Datum g, String f) {  
        name = n;  
        gebDatum = g;  
        familienstand = f;  
    }  
    Stammdaten(String n, String v, int t, int m, int j, String f) {  
        name = new Name(n, v);  
        gebDatum = new Datum(t, m, j);  
        familienstand = f;  
    }  
}
```



# Abgrenzung zwischen Reihung und Verbund

- Reihung
  - Zugriff auf Daten **gleichen Typs**, auf die mit einem **Index** zugegriffen werden kann
- Verbund
  - Zugriff auf Daten **unterschiedlichen Typs** oder **unterschiedlicher Bedeutung**, die inhaltlich zusammen gehören
  - Der Zugriff über einen (berechneten) Index wird nicht benötigt oder ist nicht sinnvoll

# Kombinationen aus Reihung und Verbund

- Verbund und Reihung können beliebig kombiniert werden
  - Reihungen von Verbunden
  - Reihungen in Verbunden

Name

Nach

Mustermann

Vor

Martin

Matrikel

123456

Punkte

0

...

1

11

# Reihung und Verbund in Java

Name	Nach	Mustermann
	Vor	Martin
Matrikel	123456	
Punkte	0	...
	1	
	11	

```
class Student {  
    Name name;  
    int matrikel;  
    int[] punkte = new int[12];  
  
    Student(Name n, int m) {  
        name = n;  
        matrikel = m;  
    }  
}
```

```
Student[] studenten = new Student[300];  
studenten[0] = new Student(new Name("Mustermann", "Martin"), 123456);  
studenten[0].punkte[0] = 95;  
:
```