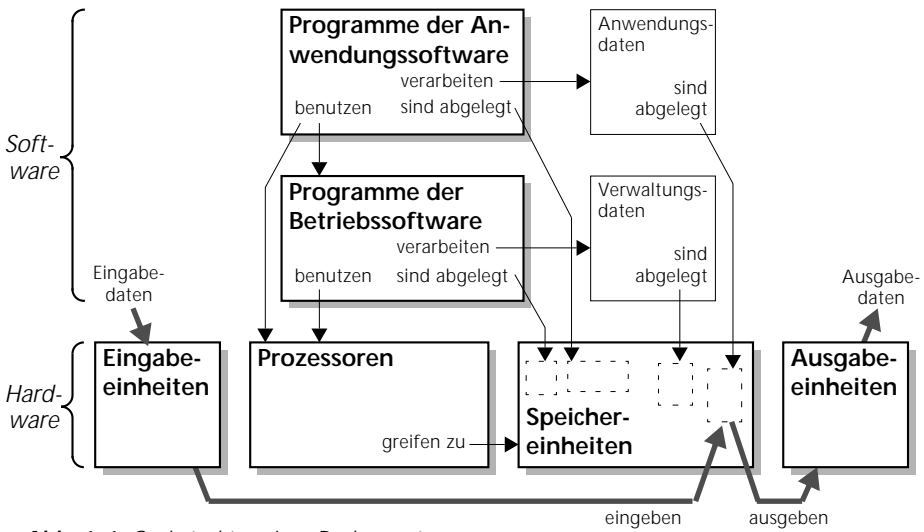


# **Lehrbuch der Programmierung mit Java**

Klaus Echte Michael Goedicke



**Abb. 1-1** Grobstruktur eines Rechensystems

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

---

Schritt 1: Lies Eingaben  $x$  und  $y$ , weiter mit Schritt 2  
Schritt 2: Falls  $x \leq y$ : weiter mit Schritt 3,  
falls  $x > y$ : weiter mit Schritt 4  
Schritt 3: Berechne  $a = y - x$ , weiter mit Schritt 5  
Schritt 4: Berechne  $a = x - y$ , weiter mit Schritt 5  
Schritt 5: Schreibe Ausgabe  $a$ , beende Ausführung

**Prog. 1-1** *Deterministischer Algorithmus*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Schritt 1: Lies Eingaben  $x$  und  $y$ , weiter mit Schritt 2 oder 3  
Schritt 2: Berechne  $a = x - y$ , weiter mit Schritt 4  
Schritt 3: Berechne  $a = y - x$ , weiter mit Schritt 4  
Schritt 4: Falls  $a \geq 0$ : weiter mit Schritt 5,  
falls  $a < 0$ : weiter mit Schritt 6  
Schritt 5: Setze  $b = a$ , weiter mit Schritt 7  
Schritt 6: Berechne  $b = -a$ , weiter mit Schritt 7  
Schritt 7: Schreibe Ausgabe  $b$ , beende Ausführung

### **Prog. 1–2** Indeterministischer Algorithmus

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Schritt 1: Lies Eingaben  $x$  und  $y$ , weiter mit Schritt 2  
Schritt 2: Berechne  $a = x + y$ , weiter mit Schritt 3  
Schritt 3: Berechne  $b = a / 2$ , weiter mit Schritt 4  
Schritt 4: Schreibe Ausgabe  $b$ , beende Ausführung

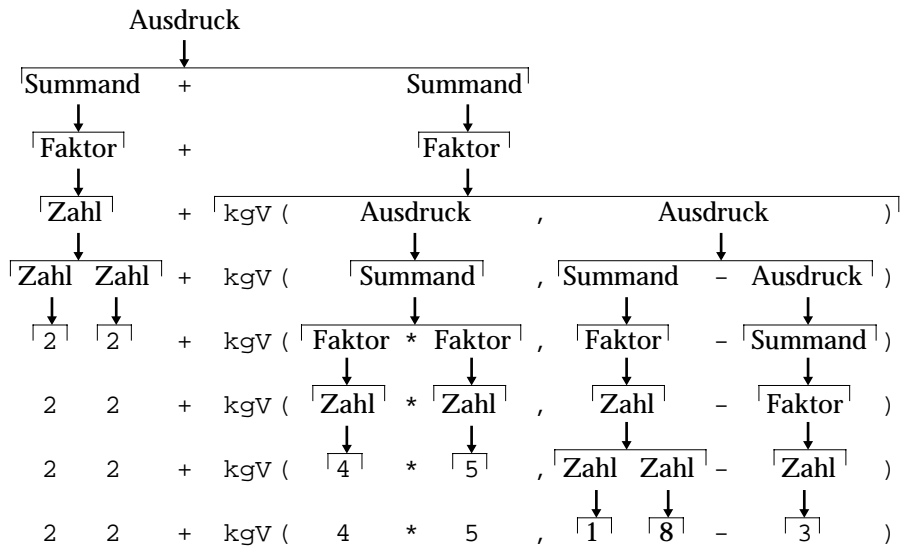
**Prog. 1–3** Berechnung des arithmetischen Mittels nach der Formel  $(x + y)/2$

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Schritt 1: Lies Eingaben  $x$  und  $y$ , weiter mit Schritt 2  
Schritt 2: Berechne  $a = 0,5 \cdot x$ , weiter mit Schritt 3  
Schritt 3: Berechne  $b = 0,5 \cdot y$ , weiter mit Schritt 4  
Schritt 4: Berechne  $c = a + b$ , weiter mit Schritt 5  
Schritt 5: Schreibe Ausgabe  $c$ , beende Ausführung

**Prog. 1–4** Berechnung des arithmetischen Mittels nach der Formel  $0,5 \cdot x + 0,5 \cdot y$

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



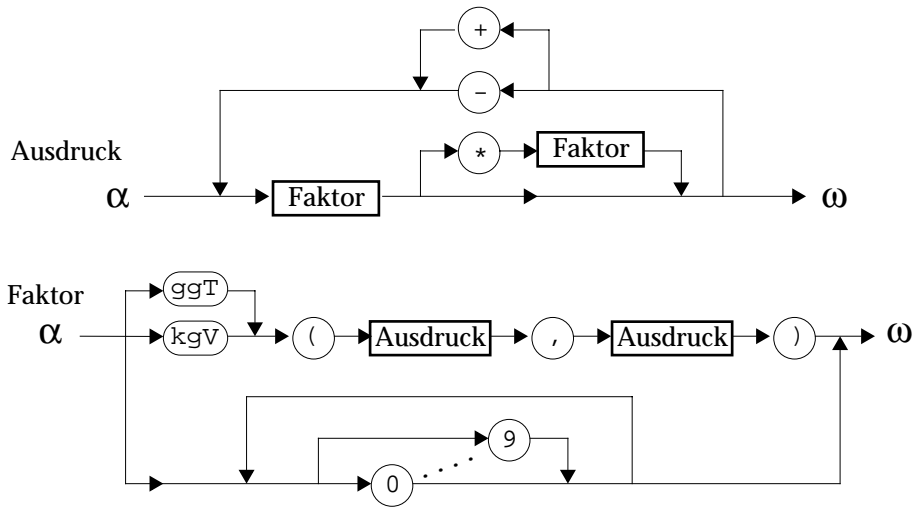
**Abb. 1-2** Ableitung eines Wortes aus dem Startsymbol „Ausdruck“

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

Produktionen einer Grammatik $a, b \in V, x, y, z, x_1, \dots, x_k \in (V \cup T)^*$	EBNF-Notation	Bemerkung
$a \rightarrow x$	$a = x$	
$a \rightarrow x_1, \dots, a \rightarrow x_k$	$a = (x_1 \mid \dots \mid x_k)$	Alternativen
$a \rightarrow xz, a \rightarrow xyz$	$a = x [ y ] z$	y ist optional
$a \rightarrow xa, a \rightarrow y$	$a = \{ x \} y$	x beliebig oft wiederholt
$a \rightarrow xb, b \rightarrow xb, b \rightarrow y$	$a = \{ x \}_1 y$	x mindestens 1 mal

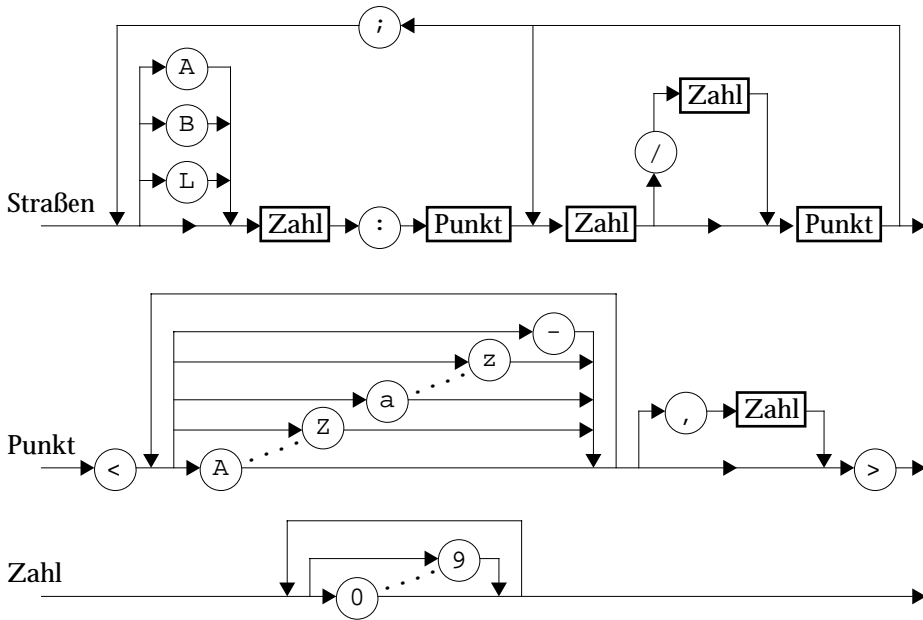
**Tab. 1-1** Erweiterte Backus-Naur-Form EBNF

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 1-3** Syntaxdiagramme für Ausdruck und Faktor zur Beschreibung einer Grammatik

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 1-4** Syntaxdiagramme für die Grammatik zur Straßen-Eingabe

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



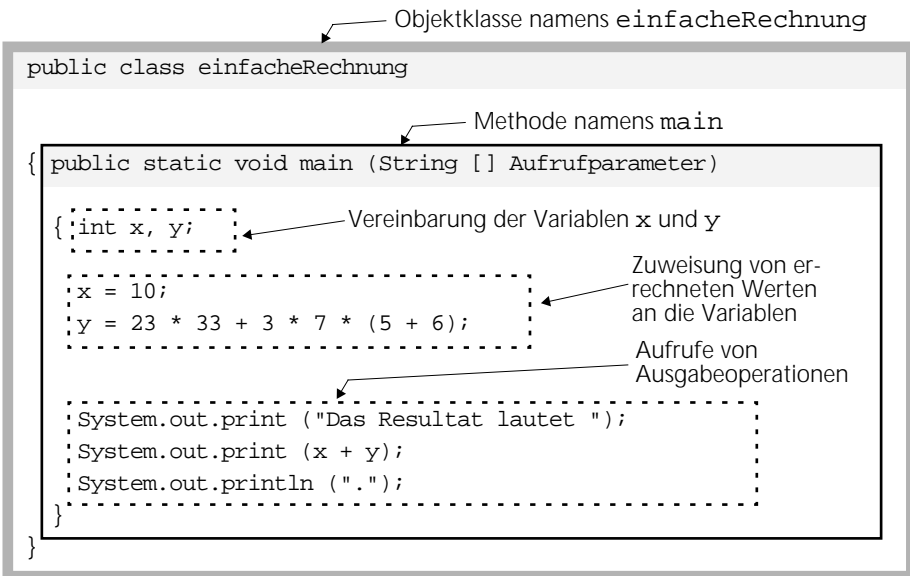
---

# 2

```
public class einfacheRechnung
{ public static void main (String [] Aufrufparameter)
  { int x, y;
    x = 10;
    y = 23 * 33 + 3 * 7 * (5 + 6);
    System.out.print ("Das Resultat lautet ");
    System.out.print (x + y);
    System.out.println (".");
  }
}
```

## **Prog. 2-1** Ein einfaches Programm

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Prog. 2-2** Erläuterung des Programms 2-1

```

public class Rechteck
{ public static void main (String [] Aufrufparameter)
  { int Laenge, Breite, Umfang, Flaeche;
    // Eingabe: ←————— Kommentar
    System.out.println ("Bitte Länge eingeben:");
    Laenge = Eingabe (); ←————— Zuweisung an eine Variable
    System.out.println ("Bitte Breite eingeben:");
    Breite = Eingabe (); ←————— Zuweisung an eine Variable
    // Berechnung: ←————— Kommentar
    Umfang = 2 * (Laenge + Breite);
    Flaeche = Laenge * Breite;
    // Ausgabe: ←————— Kommentar
    System.out.println ("R: " + Laenge + " " + Breite);
    System.out.println ("Umfang: " + Umfang);
    System.out.println ("Fläche: " + Flaeche);
  }
}

```

```

static int Eingabe ()
{ String s = "";
  try { s = new java.io.DataInputStream (System.in).
        readLine ();
    }
  catch (java.io.IOException e) {}
  return java.lang.Integer.parseInt (s);
}
}

```

**Prog. 2-3** Ein Programm zur Berechnung des Umfangs und der Fläche eines Rechtecks

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

Bitte Länge eingeben: ← Aufforderung zur Eingabe  
8 ← Eingabe  
Bitte Breite eingeben: ← Aufforderung zur Eingabe  
5 ← Eingabe  
R: 8 5 ← Ausgabe der eingegebenen Werte  
Umfang: 26 ← Ergebnisausgabe  
Fläche: 40 ← Ergebnisausgabe

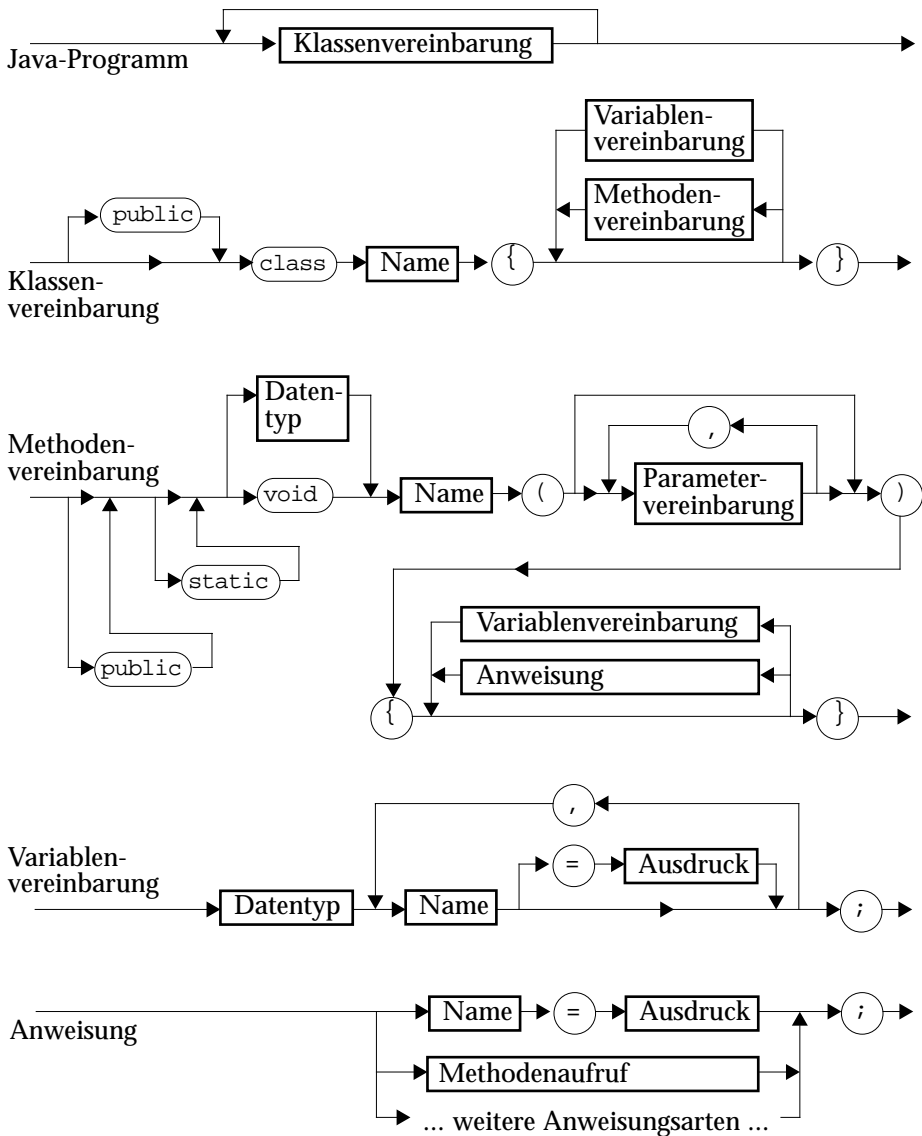
**Abb. 2-1** Ein- und Ausgabe von Prog. 2-3

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

Grundflaeche = 5 \* 7;    ← Tippfehler  
Volumen = Grundfaeche \* Hoehe;

**Prog. 2-4** Programmausschnitt ohne Variablenvereinbarung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

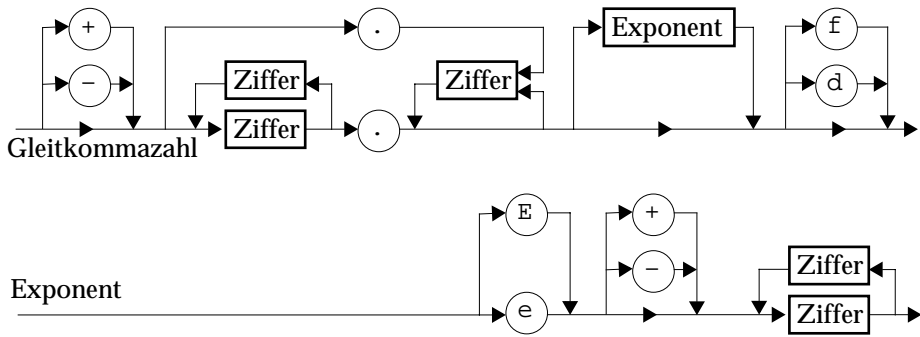


**Abb. 2-2** Syntaxdiagramme

```
int a = 5, b = 6, c;  
c = -(1 + a) * (-b + 2) - b / a + 4 * b % 13 / 2 * 3;
```

**Prog. 2-5** Ausschnitt aus einem Programm mit Variablen des Typs `int`

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2-3** Syntax einer Gleitkommazahl

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
float a = 20.5f, b = -3.e-2f, c;  
double d = 123.456, e;  
c = 3.5f * (-a + b) + b / 5.f;  
e = d + 3.4 * 7.5e4;
```

**Prog. 2-6** Programmstück mit Gleitkommazahlen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
float a = 300.1f, b = 400.1f,  
      c = 200.1f, d = 600.1f,  
      e = 10.0f, y;  
y = b * (a*b + e - c*d) * d + (a - c)/(e*e);
```

**Prog. 2-7** Programmstück mit sehr großer Rundungsabweichung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
boolean angemeldet, bezahlt, storniert, zahlungspflichtig;  
angemeldet = true;  
bezahlt    = false;  
storniert  = false;  
zahlungspflichtig = angemeldet && !(bezahlt || storniert);
```

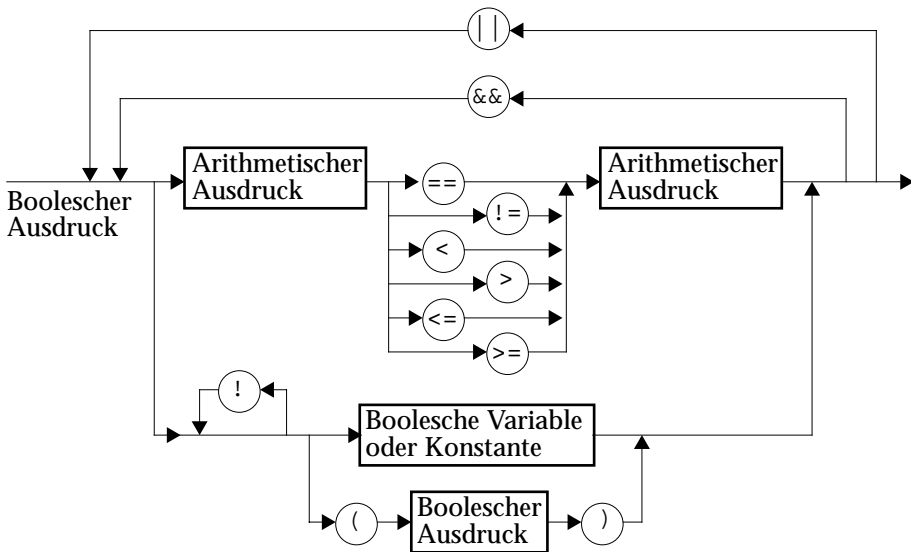
**Prog. 2-8** Programmstück mit booleschen Variablen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
float    gemessenerDruck = 5.0f, Maximaldruck = 18.8f;
int      ZuflussStufe = 2, AbflussStufe = 3;
boolean  Ueberdruck, unkritisch;
Ueberdruck = gemessenerDruck > Maximaldruck;
unkritisch =    Ueberdruck
                && gemessenerDruck < 1.2f * Maximaldruck
                && ZuflussStufe <= AbflussStufe;
```

**Prog. 2-9** Programmstück mit Booleschen Variablen und Vergleichen von Zahlen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2–4** Syntaxdiagramm eines Booleschen Ausdrucks

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

---

```
char    rund = '(', eckig = '[', geschweift = '{';
boolean x, y;
x = rund < geschweift;
y = eckig == geschweift;
```

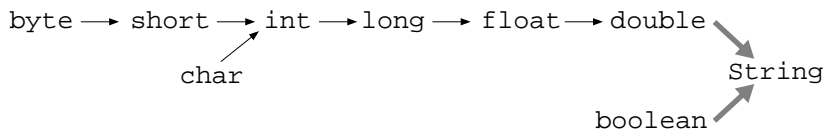
**Prog. 2-10** Programmstück mit Zeichen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
String a = "merkwürdig", b = "merkwürdig",  
      c = new String ("merkwürdig"),  
      d = new String ("merkwürdig");  
System.out.println (a + " " + (a==b) + " " + c + " " + (c==d));
```

**Prog. 2-11** Programmstück mit Strings

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2-5** Implizite Datentyp-Anpassungen in Java

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
float x = 82.2f, y = 8.5f;
int    n = (int)(x/y);
char   a = (char)(int)x, b = (char)(a + 1);
System.out.println ("gewandelt: " + n + " " + a + " " + b);
```

**Prog. 2-12** Programmstück mit expliziter Datentyp-Umwandlung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

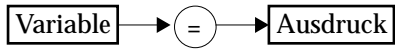
---

```
int a = 1, b = 2, c, d;  
a = 3 + 2 * (a + b);  
b = 7 - a/2;  
c = 3 * (d = a + b + 1);
```

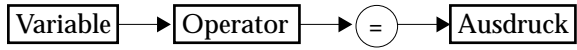
**Prog. 2-13** *Programmausschnitt mit Zuweisungen*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

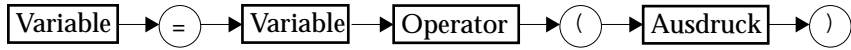
Zuweisung



Spezielle Kurznotation einer Zuweisung



steht abkürzend für:



**Abb. 2-6** Syntax der Zuweisung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
int a = 2, b = 3;  
a *= b++;  
b += 19 / --a;
```

**Prog. 2-14** *Programm-Ausschnitt mit Kurznotationen von Zuweisungen*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
int p = Eingabe (), q = Eingabe (), r = Eingabe ();           Zeile 1
int Ganze, Zaehler, Nenner;                                   2
float Wert;                                                  3
Ganze = p/q;   Zaehler = p%q;   Nenner = q;                 4
Wert = (float)p/q;                                          5
System.out.println (p + "/" + q + " = " + Ganze + " "      6
    + Zaehler + "/" + Nenner + " = " + Wert);              7
p = p + (r + 2)*q + r + 1;                                  8
Ganze = p/q;   Zaehler = p%q;   Nenner = q;                 9
Wert = (float)p/q;                                         10
System.out.println (p + "/" + q + " = " + Ganze + " "     11
    + Zaehler + "/" + Nenner + " = " + Wert);              12
```

**Prog. 2-15** Programmausschnitt berechnet gemischte Zahlen

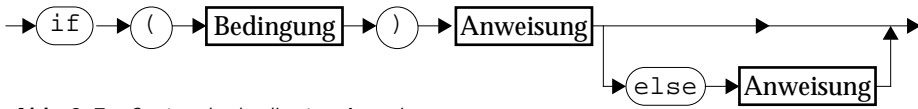
*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*

```
int p = Eingabe (), q = Eingabe (), r = Eingabe ();           Zeile 1
int Ganze, Zaehler, Nenner;                                   2
float Wert = (Ganze = p/q)                                    3
                + (float)(Zaehler = p*q)/(Nenner = q);       4
System.out.println (p + "/" + q + " = " + Ganze + " "       5
                + Zaehler + "/" + Nenner + " = " + Wert);    6
Wert = (Ganze += (Zaehler += Nenner + ++r)/Nenner + r)      7
        + (float)(Zaehler %= Nenner)/Nenner;                8
System.out.println (Ganze * Nenner + Zaehler + "/" + q      9
                + " = " + Ganze + " " + Zaehler + "/" + Nenner 10
                + " = " + Wert);                               11
```

### **Prog. 2-16** Programmausschnitt mit Zuweisungen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

## Bedingte Anweisung



**Abb. 2-7** Syntax der bedingten Anweisung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
if (g == 1)          k += 100.0f;  
else if (g == 2)    k += 10000.0f;
```

**Prog. 2-17** *Bedingte Anweisung zur Gewinnausschüttung*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
float Winkel = Eingabe ();  
if (Winkel > 90.0f && Winkel < 180.0f)  
{ System.out.println ("stumpfer Winkel");  
  Winkel = 180.0f - Winkel;  
}
```

**Prog. 2-18** *Bedingte Anweisung zur Abbildung eines stumpfen auf einen spitzen Winkel*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
int i = Eingabe (), j = Eingabe ();           Zeile 1
if (i == 5)                                   2
    if (j == 5)                               3
        System.out.println ("i und j sind 5"); 4
    else                                       5
        System.out.println ("nur i ist 5");    6
else                                           7
    if (j == 5)                               8
        System.out.println ("nur j ist 5");    9
```

**Prog. 2-19** Verschachtelte bedingte Anweisungen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
int i = Eingabe (), j = Eingabe ();           1
if (i == 5)                                   2
{ if (j == 5)                                  3
    System.out.println ("i und j sind 5");    4
}                                               zusätzliche Zeile 4a
else                                           5
    System.out.println ("i ist nicht 5");     6
if (j == 5)                                   8
    System.out.println ("j ist 5");          9
```

**Prog. 2-20** Verschachtelte bedingte Anweisungen

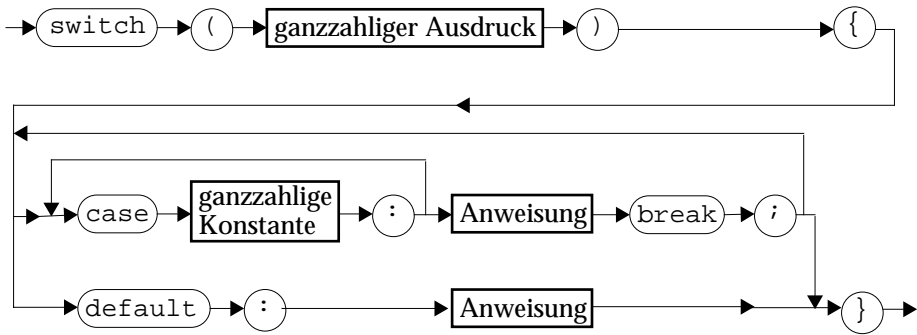
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
float Winkel = Eingabe ();
if      (Winkel <  90.0f)
    System.out.println ("spitzer Winkel");
else if (Winkel == 90.0f)
    System.out.println ("rechter Winkel");
else if (Winkel < 180.0f)
    System.out.println ("Stumpfer Winkel");
else if (Winkel == 180.0f)
    System.out.println ("gestreckter Winkel");
```

**Prog. 2-21** Verschachtelte bedingte Anweisungen für eine größere Fallunterscheidung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

## switch-Anweisung



**Abb. 2-8** Syntax der switch-Anweisung

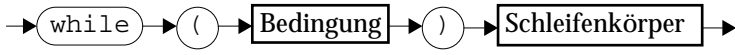
Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

```
char Ziffer;  int Wert;
Ziffer = ...
switch (Ziffer)
{ case '0': case '1': case '2': case '3': case '4':
  case '5': case '6': case '7': case '8': case '9':
    Wert = Ziffer - '0';                break;
  case 'a': case 'b': case 'c': case 'd': case 'e':
  case 'f':
    Wert = Ziffer - 'a' + 10;          break;
  case 'A': case 'B': case 'C': case 'D': case 'E':
  case 'F':
    Wert = Ziffer - 'A' + 10;          break;
  default: System.out.println (Ziffer + " ist ungültig");
}
```

**Prog. 2-22** *Beispiel einer switch-Anweisung*

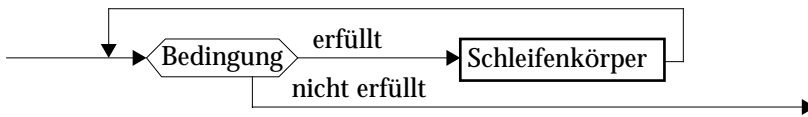
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

## while-Schleife



**Abb. 2-9** Syntax der *while*-Schleife

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2-10** Semantik der while-Schleife

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*

```
int i = 1, a = 3;
while (i++ < 1000000) a = (4*a + 5*a*a)%13579;
```

**Prog. 2–23** *while-Schleife berechnet das millionste Element einer Zahlenfolge*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
.  
  
int      n = Eingabe (),  Wurzel,  Teiler = 2;  
boolean  istPrimzahl = true;  
Wurzel = (int) java.lang.Math.sqrt ((float) n);  
while (Teiler <= Wurzel && istPrimzahl)  
    if (n % Teiler == 0)  istPrimzahl = false;  
    else                  Teiler++;  
System.out.println (n + " prim: " + istPrimzahl);
```

**Prog. 2-24** *while-Schleife prüft, ob eine Zahl prim ist*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

n	Wurzel	Teiler	istPrimzahl
21	4	2	true
21	4	3	false

(2 Iterationen)

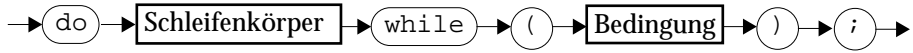
n	Wurzel	Teiler	istPrimzahl
37	6	2	true
37	6	3	true
37	6	4	true
37	6	5	true
37	6	6	true

(5 Iterationen)

**Abb. 2-11** Iterationen bei Ausführung von Prog. 2-24

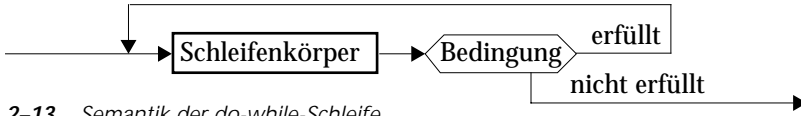
Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

## do-while-Schleife



**Abb. 2-12** Syntax der do-while-Schleife

*.ehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2-13** Semantik der *do-while*-Schleife

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
int Summe = 0, Anzahl = 0;
do { Summe = Summe + Eingabe ();
    Anzahl++;
}
while (Summe <= 100);
System.out.println ("Sum. " + Summe + ", Anz. " + Anzahl);
```

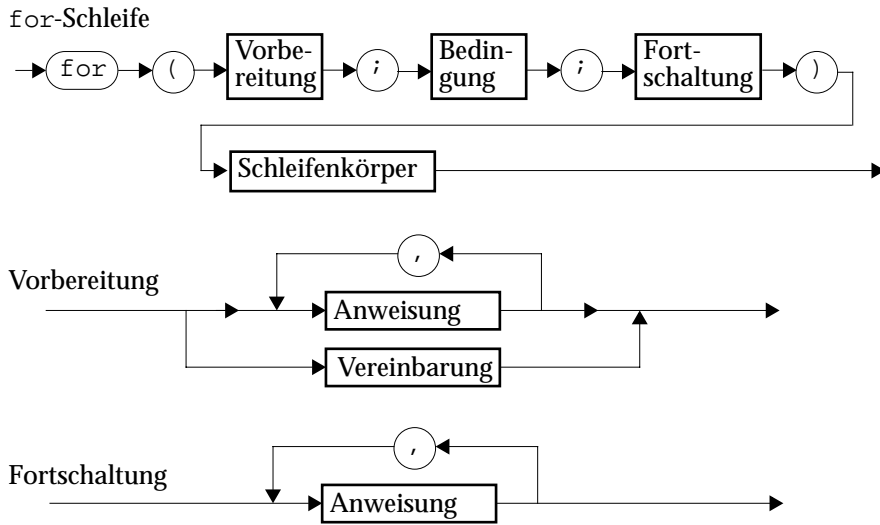
**Prog. 2-25** *Programmausschnitt mit einer do-while-Schleife*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
.  
  
float x = Eingabe (),  
      uG = 0, oG = x + 1, m, epsilon = 0.001f;  
do { m = (uG + oG)/2;  
    if (m*m > x) oG = m;  
    else uG = m;  
    }  
while (oG - uG > epsilon);  
System.out.println ( "Wurzel " + x  
                    + " beträgt ungefähr " + m);
```

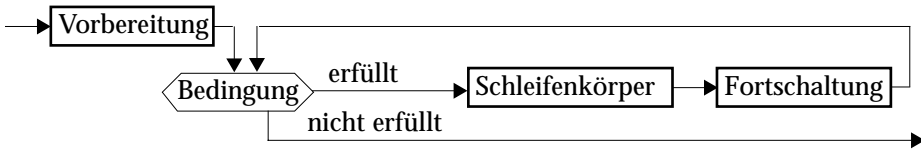
**Prog. 2-26** *Programmausschnitt zur Intervallschachtelung*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2-14** Syntax der for-Schleife

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 2-15** Semantik der for-Schleife

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
for (int i = 1, a = 3; i < 1000000; i++)  
    a = (4*a + 5*a*a)%13579;
```

**Prog. 2-27** *for-Schleife berechnet das millionste Element einer Zahlenfolge*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
int x, y;  
for (x = -5; x <= 12; x++)  
{ y = x*x - 8;  
  System.out.println ("x = " + x + ",   y = " + y);  
}
```

**Prog. 2-28** *for-Schleife druckt eine Wertetabelle*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
for (int i = 10; i <= 30; i = i + 10)
```

```
    for (int j = 1; j <= 4; j++)  
        System.out.print (i + " " + j + ", " );
```

innere  
Schleife

äußere  
Schleife

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

**Prog. 2-29** Zwei verschachtelte for-Schleifen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
for (int i = 1; i <= 3; i++)
```

```
    for (int j = 1; j <= i; j++)
```

```
        System.out.print (i + " " + j + ", ");
```

innere  
Schleife

äußere  
Schleife

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

**Prog. 2-30** Zwei verschachtelte for-Schleifen mit voneinander abhängigen Laufvariablen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

## Kalender 2003

## Januar

Mo 6 13 20 27  
Di 7 14 21 28  
Mi 1 8 15 22 29  
Do 2 9 16 23 30  
Fr 3 10 17 24 31  
Sa 4 11 18 25  
So 5 12 19 26

## Februar

Mo 3 10 17 24  
Di 4 11 18 25  
Mi 5 12 19 26  
Do 6 13 20 27  
Fr 7 14 21 28  
Sa 1 8 15 22  
So 2 9 16 23

## März

Mo 3 10 17 24 31  
Di 4 11 18 25  
Mi 3 10 17 24  
Do 5 12 19 26  
Fr 7 14 21 28  
Sa 1 8 15 22 29  
So 2 9 16 23 30

## April

Mo 7 14 21 28  
Di 1 8 15 22 29  
..... usw.

## Mai

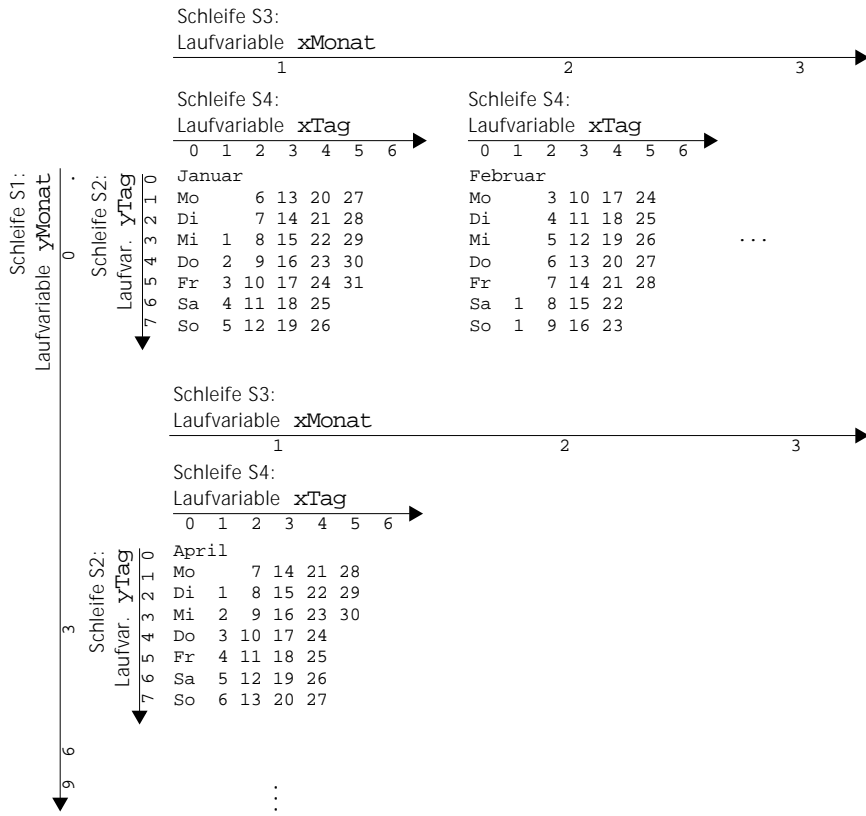
Mo 5 12 19 26  
Di 6 13 20 27

## Juni

Mo 2 9 16 23 30  
Di 3 10 17 24

**Abb. 2-16** Ausschnitt aus dem auszudruckenden Kalender

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 2-17** Laufvariablen der vier verschachtelten Schleifen S1, S2, S3 und S4

```

public class Kalender
{ public static void main (String [] unbenutzt)
  { int      Jahr   = Eingabe (),      // Jahreszahl.
        WochJ = Eingabe (),      // erster Wochentag des Jahres.
        vorM = 0,                // Anzahl der Tage der Vormonate.
        imM = 0,                // Anzahl der Tage im Monat.
        s = 0;                  // Anzahl der Schalttage (29. Februar).

    String NameM = "kein Monat"; // Name des Monats.
    System.out.println ("Kalender " + Jahr);
    if (Jahr % 4 == 0 && Jahr % 100 != 0 || Jahr % 400 == 0) s = 1;
    // S1: Schleife über die untereinander zu druckenden Monate:
    for (int yMonat = 0; yMonat < 12; yMonat += 3)
    { System.out.println ("");
      // S2: Schleife über untereinander zu druckende Tage in einem Monat:
      for (int yTag = 0; yTag <= 7; yTag++)
      { // S3: Schleife über die nebeneinander zu druckenden Monate:
        for (int xMonat = 1; xMonat <= 3; xMonat++)
        { switch (yMonat + xMonat)
          { case 1: NameM = "Januar "; vorM = 0; imM = 31; break;
            case 2: NameM = "Februar "; vorM = 31; imM = 28 + s; break;
            case 3: NameM = "März "; vorM = 59 + s; imM = 31; break;
            case 4: NameM = "April "; vorM = 90 + s; imM = 30; break;
            case 5: NameM = "Mai "; vorM = 120 + s; imM = 31; break;
            case 6: NameM = "Juni "; vorM = 151 + s; imM = 30; break;
            case 7: NameM = "Juli "; vorM = 181 + s; imM = 31; break;
            case 8: NameM = "August "; vorM = 212 + s; imM = 31; break;
            case 9: NameM = "September"; vorM = 243 + s; imM = 30; break;
            case 10: NameM = "Oktober "; vorM = 273 + s; imM = 31; break;
            case 11: NameM = "November "; vorM = 304 + s; imM = 30; break;
            case 12: NameM = "Dezember "; vorM = 334 + s; imM = 31; break;
          }
        switch (yTag)
        { case 0: System.out.print (" " + NameM + " "); break;
          case 1: System.out.print ("Mo"); break;
          case 2: System.out.print ("Di"); break;
          case 3: System.out.print ("Mi"); break;
          case 4: System.out.print ("Do"); break;
          case 5: System.out.print ("Fr"); break;
          case 6: System.out.print ("Sa"); break;
          case 7: System.out.print ("So"); break;
        }
        // S4: Schleife über die nebeneinander zu dr. Tage in einem Monat:
        for (int xTag = 0; xTag <= 6 && 0 < yTag; xTag++)
        { int Tag = 7*xTag + yTag - (WochJ + vorM - 1) % 7;
          if (1 <= Tag && Tag <= 9) // einstellige Zahl.
            System.out.print (" " + Tag);
          else if (10 <= Tag && Tag <= imM) // zweistellige Zahl.
            System.out.print (" " + Tag);
          else // keine Zahl
            System.out.print (" ");
        }
      }
      System.out.println (""); // Vorschub zur nächsten Druckzeile.
    }
  }
}

```

### Prog. 2-31 Drucken eines Kalenders mit verschachtelten for-Schleifen

```

// Gib jeden Freitag, den 13., eines Jahres aus.

int Wochentag = Eingabe (), // letzter Wochentag des Vorjahres.
    Schalttag = Eingabe (), // 1 im Schaltjahr, 0 sonst.
    Tag, Monat;

Monatsschleife:
for (Monat = 1; Monat <= 12; Monat++)
{ Tag = 0;

    Tagesschleife:
    while (++Tag <= 31)
    {
        if (    Monat == 2 && Tag > 28 + Schalttag
            || (    Monat == 4 || Monat == 6
                || Monat == 9 || Monat == 11) && Tag > 30)
            continue Monatsschleife;

        Wochentag = Wochentag % 7 + 1; // nächster Wochentag.

        if (Tag != 13) // Datum kann nicht Freitag, der 13., sein.
            continue Tagesschleife;

        if (Wochentag == 5) // Fünfter Wochentag ist Freitag.
            System.out.println ("Freitag, der 13." + Monat + ".");

        if (Monat == 12) // Abbruch nach dem 13.12.
            break Monatsschleife;
    }
}

System.out.println ("Dies sind die Glückstage des Jahres.");

```

### Prog. 2-32 Wirkung von break und continue

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



---

```
int [] m = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
int Mai = 5;  
int Tagesanzahl = m [Mai];
```

**Prog. 2-34** Erzeugung eines Arrays durch eine Liste von Konstanten

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
int [][] a = {{ 0, 1}, {11, 12}, {21, 22}},  
            b = new int [3][2];  
b [0][0] = 0;   b [0][1] = 1;  
b [1][0] = 11;  b [1][1] = 12;  
b [2][0] = 21;  b [2][1] = 22;
```

**Prog. 2-35** Erzeugung eines 2-dimens. Arrays durch verschachtelte Listen von Konstanten

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
int Anzahl = Eingabe ();
int Anfangswert = 0;
int [] [] Matrix = new int [Anzahl] [Anzahl + 12];
for (int i = 0; i < Matrix.length; i++)
    for (int j = 0; j < Matrix [0].length; j++)
        Matrix [i] [j] = Anfangswert;
```

**Prog. 2-36** Elementanzahl eines Arrays bildet Obergrenze für Laufvariablen

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*

```
int [][] Mat, Tabelle;  
Mat      = new int [12][15];  
Tabelle = new int [19][24];  
Mat      = Tabelle;  
int [][] Tabelle2 = {{52, 14, 15, 19}, {23, 61, 22, 29}};  
Tabelle = new int [2][Eingabe ()];
```

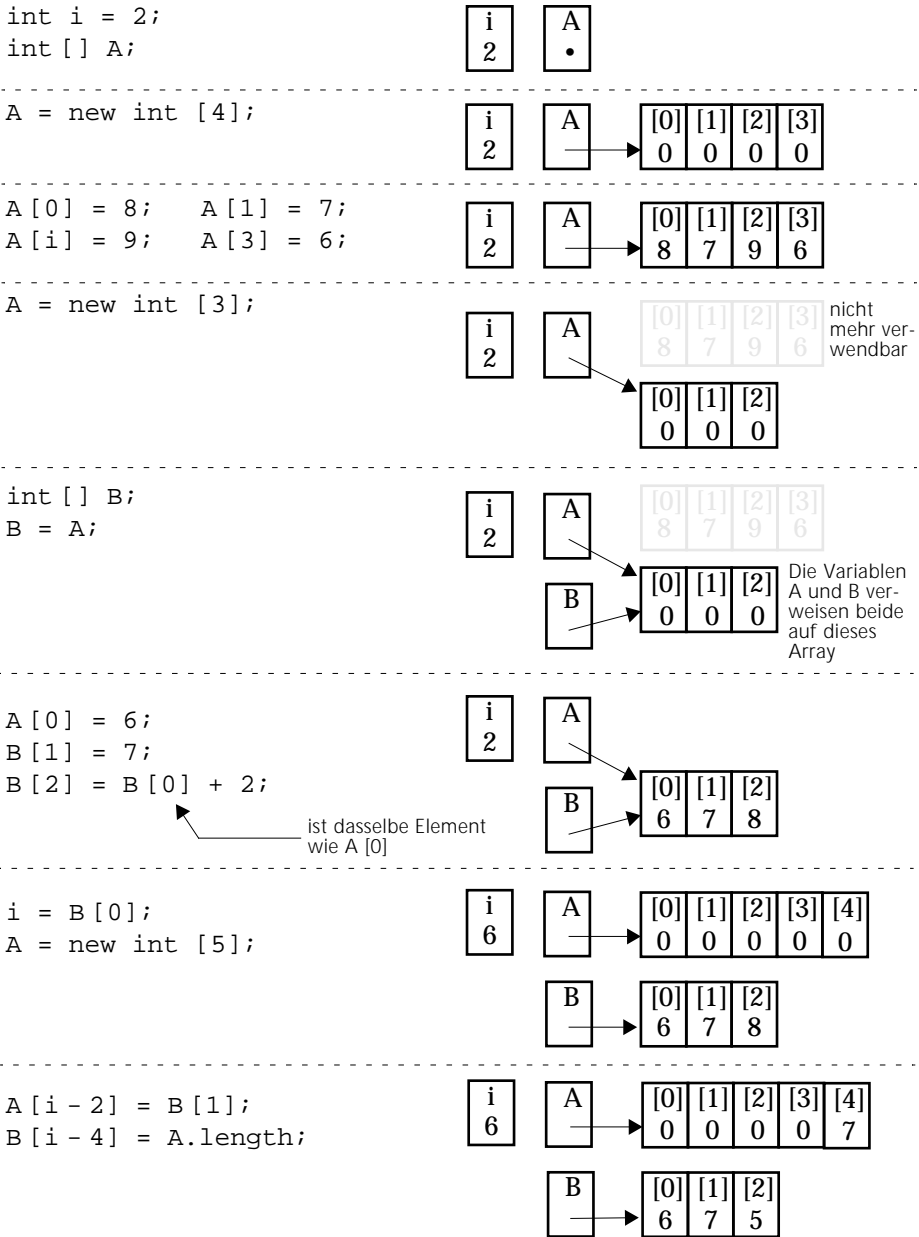
**Prog. 2-37** Zuweisung von Arrays an Array-Variablen

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2-18** links: `int`-Variable, Mitte: Array-Variable, die auf drei Array-Elemente zeigt

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 2-19** Verweise auf Arrays bei Ausführung eines Beispielprogramms

```
public class Sortierung
{ public static void main (String [] unbenutzt)
  { int i, j, z, n = Eingabe ();
    int [] a = new int [n];

    // Lies Elemente ein und gib sie dabei sofort wieder aus:
    System.out.println ("Die eingegebenen Elemente:");
    for (i = 0; i < n; i++)
    { a [i] = Eingabe ();
      System.out.print (a [i] + " ");
    }
    System.out.println ("\n"); // Zeilenvorschub.

    // Sortiere Elemente:
    for (i = 0; i < n - 1; i++)
      // Prüfe, ob a [i] Nachfolger hat, die kleiner
      // als a [i] sind:
      for (j = i + 1; j < n; j++)
        if (a [i] > a [j]) // Ist Nachfolger a [j]
                          // kleiner als a [i] ?
          { // Vertausche a [i] mit a [j]:
            z = a [i]; a [i] = a [j]; a [j] = z;
          }

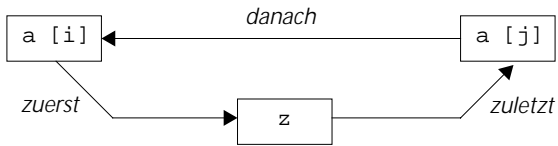
    // Gib sortierte Elemente aus:
    System.out.println ("Sortierte Elemente:");
    for (i = 0; i < n; i++)
      System.out.print (a [i] + " ");
    System.out.println ("\n"); // Zeilenvorschub.
  }
}
```

**Prog. 2-38** Einfacher interner Sortieralgorithmus (Die Implementierung der Methode `Eingabe ()` ist in Prog. 2-3 angegeben, siehe Seite 23)

i	j	a[0]	a[1]	a[2]	a[3]
0	1	81	95	32	60
0	2	81	95	32	60
		32	95	81	60
0	3	32	95	81	60
1	2	32	95	81	60
		32	81	95	60
1	3	32	81	95	60
		32	60	95	81
2	3	32	60	95	81
		32	60	81	95

**Abb. 2-20** Variablenbelegung bei Ausführung des Sortieralgorithmus

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



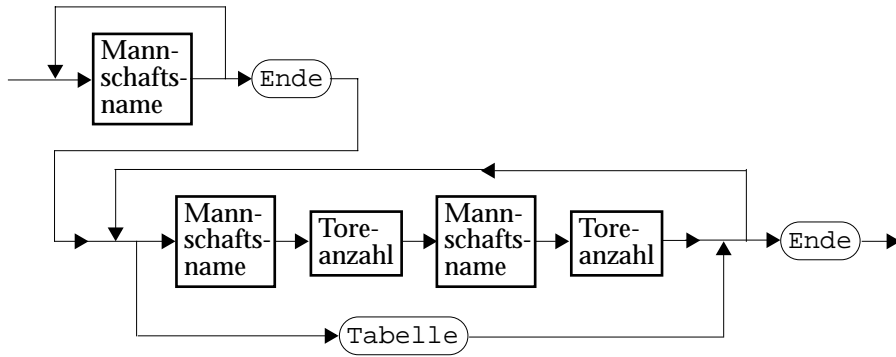
**Abb. 2-21** Vertauschung der Inhalte von  $a[i]$  und  $a[j]$

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

	Dortmund	Schalke	Duisburg	Leverkusen
Dortmund	Tore [0][0] <b>-1</b>	Tore [0][1] <b>3</b>	Tore [0][2] <b>7</b>	Tore [0][3] <b>1</b>
Schalke	Tore [1][0] <b>2</b>	Tore [1][1] <b>-1</b>	Tore [1][2] <b>0</b>	Tore [1][3] <b>1</b>
Duisburg	Tore [2][0] <b>4</b>	Tore [2][1] <b>5</b>	Tore [2][2] <b>-1</b>	Tore [2][3] <b>2</b>
Leverkusen	Tore [3][0] <b>0</b>	Tore [3][1] <b>1</b>	Tore [3][2] <b>3</b>	Tore [3][3] <b>-1</b>

**Abb. 2-22** Belegung des Arrays `Tore` mit Spielergebnissen

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 2-23** Syntax zur Eingabe von Mannschaften und Spielresultaten

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```

public class Platzierung
{
    public static void main (String [] unbenutzt)
    {
        int i, j, z, n = 0, m = 40; String zz;
        String M1, M2; // Mannschaften eines Spiels.
        int i1, i2; // Indizes von M1 bzw. M2.
        int t1, t2; // Anzahl der von M1 bzw. M2 erzielten Tore.

        // Schritt 1: Erzeuge Arrays und initialisiere Tore mit -1:
        String [] Mannschaft = new String [m], M = new String [m];
        int [][] Tore = new int [m][m];
        int [] Punkte = new int [m];
        for (i = 0; i < m; i++) for (j = 0; j < m; j++) Tore [i][j] = -1;

        // Schritt 2: Lies Mannschaftsnamen ein:
        M1 = EingabeString ();
        while ( ! gleich (M1, "Ende")) // nicht "Ende" eingegeben ?
        {
            Mannschaft [n++] = M1; M1 = EingabeString ();
        }

        // Schritt 3: Bearbeite Eingabe von Spielen:
        M1 = EingabeString ();
        while ( ! gleich (M1, "Ende")) // nicht "Ende" eingegeben ?
        {
            if (gleich (M1, "Tabelle")) // Tabellen-Erstellung verlangt ?
            {
                // Schritt 5.1: Berechne Punkte aller Mannschaften:
                for (i = 0; i < n; i++) // Anfänglich hat jede
                    Punkte [i] = 0; // Mannschaft 0 Punkte.
                for (i = 0; i < n - 1; i++) // Verschachtelte Schleifen
                    for (j = i + 1; j < n; j++) // über alle Spiele.
                    {
                        t1 = Tore [i][j]; t2 = Tore [j][i];
                        if (0 <= t1 && t1 < t2) Punkte [j] += 3;
                        else if (0 <= t2 && t2 < t1) Punkte [i] += 3;
                        else if (0 <= t1 && t1 == t2) { Punkte [i]++; Punkte [j]++; }
                    }

                // Schritt 5.2: Sortiere absteigend nach Punkten:
                for (i = 0; i < n; i++) // Kopiere Array Mannschaft
                    M [i] = Mannschaft [i]; // in Array Punkte.
                for (i = 0; i < n - 1; i++)
                    for (j = i + 1; j < n; j++) // Sortiere Arrays Punkte und M:
                    if (Punkte [i] < Punkte [j])
                    {
                        z = Punkte [i]; Punkte [i] = Punkte [j]; Punkte [j] = z;
                        zz = M [i]; M [i] = M [j]; M [j] = zz;
                    }

                // Schritt 5.3: Gib Arrays M und Punkte aus:
                for (i = 0; i < n; i++)
                    System.out.println (rechts (Punkte [i]) + " " + M [i]);
            }
            else // Schritt 4.1: Lies Eingabe eines Spielresultats:
            {
                t1 = Eingabe (); M2 = EingabeString (); t2 = Eingabe ();

                // Schritt 4.2: Bestimme Indizes i1 und i2 und trage Tore ein.:
                for (i = 0, i1 = -1; i < n && i1 == -1; i++) // Schleife sucht M1
                    if (Mannschaft [i].equals (M1)) i1 = i; // und bestimmt i1.
                if (i1 < 0) System.out.println ("Keine Mannschaft: " + M1);
                for (i = 0, i2 = -1; i < n && i2 == -1; i++) // Schleife sucht M2
                    if (Mannschaft [i].equals (M2)) i2 = i; // und bestimmt i2.
                if (i2 < 0) System.out.println ("Keine Mannschaft: " + M2);
                if (i1 >= 0 && i2 >= 0 && t1 >= 0 && t2 >= 0) // Gültige Eingabe ?
                {
                    Tore [i1][i2] = t1; Tore [i2][i1] = t2; // Trage Tore ein.
                }
                else // Ungültige Eing.:
                    System.out.println ("Ungültige Eingabe."); // Melde Fehler.
            }
        }
        M1 = EingabeString (); // Lies nächste Eingabe.
    }
}

```

### Prog. 2-39 Berechnung der Platzierung von Mannschaften (Hauptteil des Programms)

```
static int Eingabe ()           // Eingabe einer ganzen Zahl.
{ String s = "";
  try { s = new java.io.DataInputStream (System.in).readLine (); }
  catch (java.io.IOException e) {}
  return java.lang.Integer.parseInt (s);
}

static String EingabeString () // Eingabe eines Strings.
{ String s = "";
  try { s = new java.io.DataInputStream (System.in).readLine (); }
  catch (java.io.IOException e) {}
  return s.trim ();
}

static boolean gleich (String s1, String s2) // Abfrage, ob die Strings
{ return s1.equals (s2);                    // s1 und s2 den gleichen
}                                           // Inhalt besitzen.

static String rechts (int x)              // Ganze Zahl mit 10 Zeichen
{ String r = "          " + x;           // rechtsbündig schreiben.
  return r.substring (r.length () - 10, r.length ());
}
}
```

**Prog. 2-40** Berechnung der Platzierung von Mannschaften (Rest des Programms)

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000



**Abb. 3-1** Syntax zur Vereinbarung einer Klasse

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Rechteck
{ float Laenge, Breite;   int Strichstaerke;
  float Flaeche ()
  { return Laenge * Breite;
  }
}

public class Hauptprogramm
{ public static void main (String [] unbenutzt)
  { Rechteck R, S, T, U;   float f;
    R = new Rechteck ();
    S = new Rechteck ();
    T = new Rechteck ();
    U = new Rechteck ();
    ...
    R = S;
    ...
    f = R.Flache ();
    ...
  }
}
```

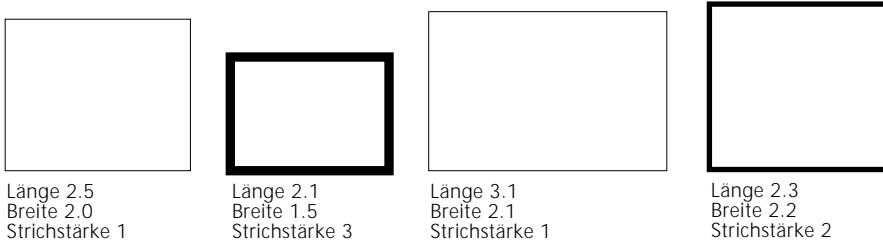
**Prog. 3-1** Darstellung von vier Rechtecken durch Objekte der Klasse Rechteck

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
public class Hauptprogramm
{ public static void main (String [] unbenutzt)
  { int n = 4;    float f;
    float [] Laenge = new float [n];
    float [] Breite = new float [n];
    int [] Strichstaerke = new int [n];
    ...
    Laenge [0] = Laenge [1];
    Breite [0] = Breite [1];
    Strichstaerke [0] = Strichstaerke [1];
    ...
    f = Laenge [0] * Breite [0];
    ...
  }
}
```

**Prog. 3-2** Darstellung von vier Rechtecken durch Arrays

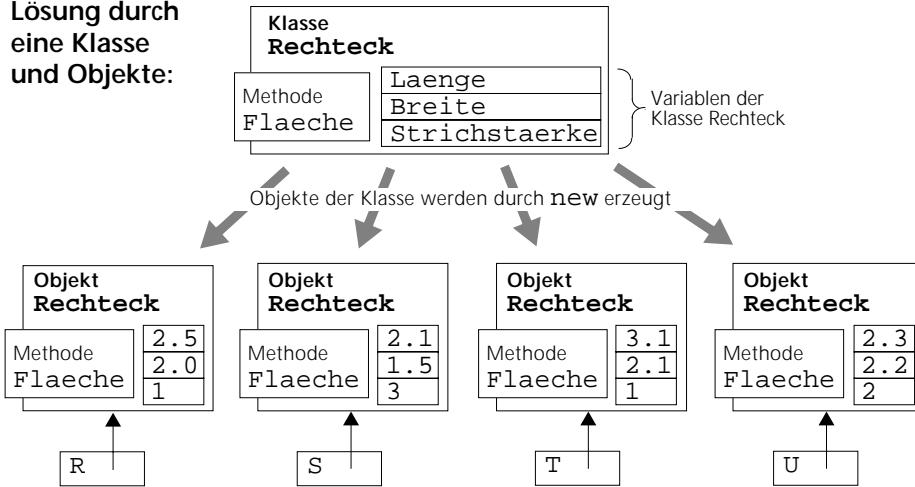
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



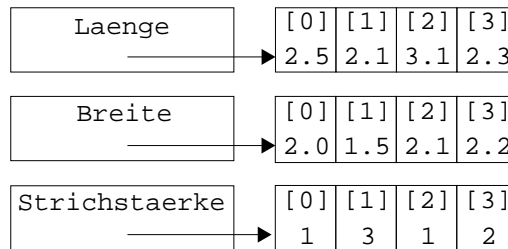
**Abb. 3-2** Vier Rechtecke, die in einem Programm darzustellen sind

*Lehrbuch der Programmierung mit Java, Echte Goedlcke, Heidelberg, © dpunkt 2000*

Lösung durch  
eine Klasse  
und Objekte:

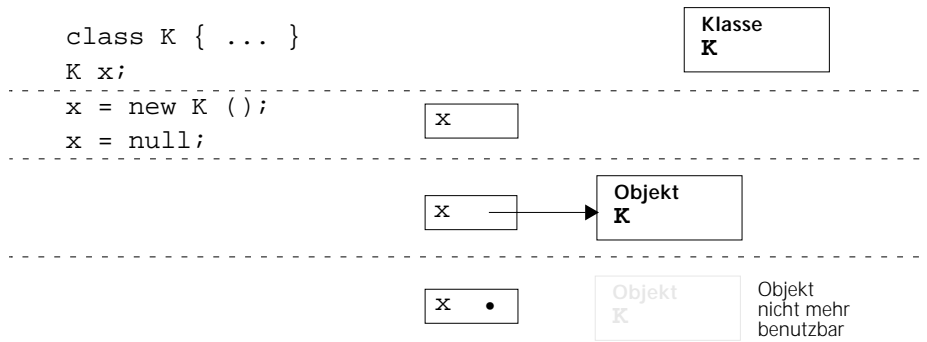


Lösung durch  
Arrays:



**Abb. 3-3** Klassen und Objekte im Vergleich zu Arrays

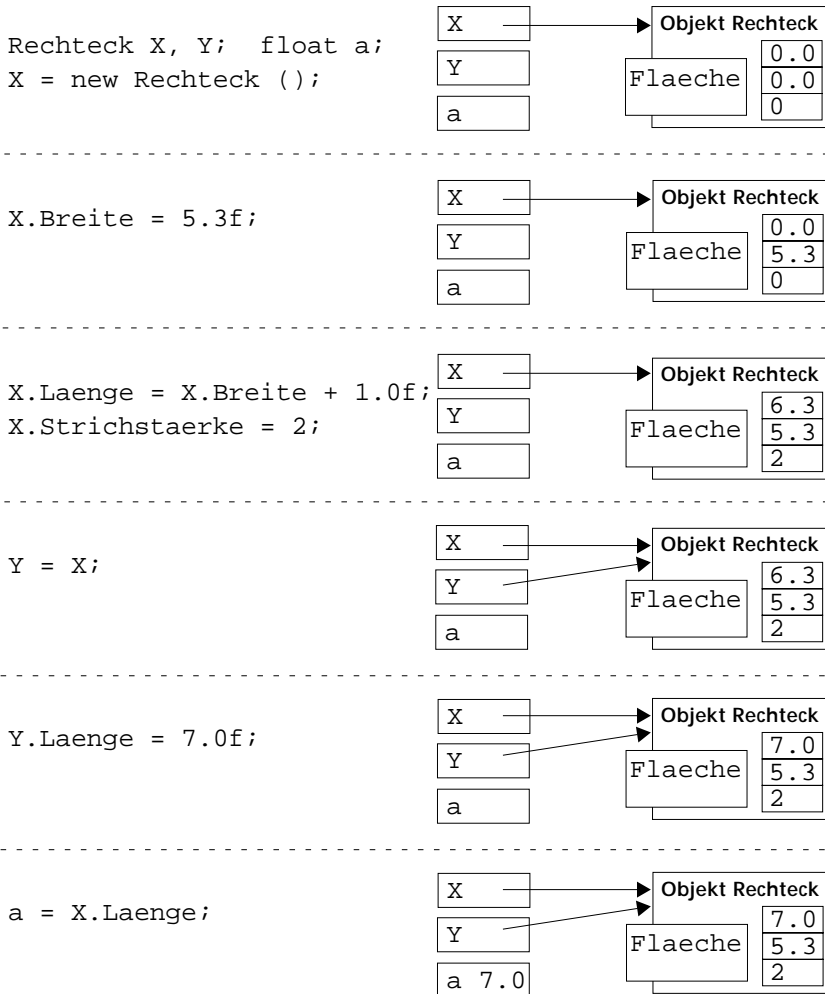
(in den Objekten sind die Namen der Variablen *Laenge*, *Breite* und *Strichstaerke* nicht eingetragen, um die Skizze nicht zu überladen)



**Abb. 3-4** Verweisvariable `x`

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

Die Klasse `Rechteck` sei wie in Prog. 3-1 vereinbart



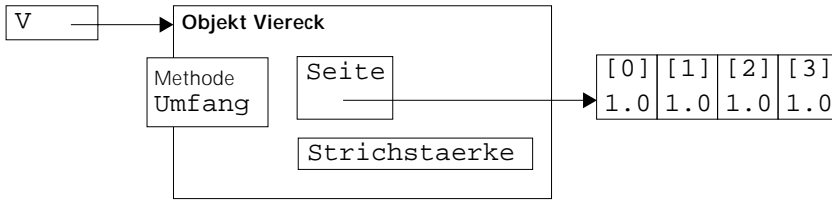
**Abb. 3-5** Objektstrukturen, die von den Anweisungen nacheinander erzeugt werden

```
class Viereck
{ float [] Seite;   int Strichstaerke;
  float Umfang ()
  { return Seite [0] + Seite [1] + Seite [2] + Seite
[3];
  }
}

public class Hauptprogramm
{ public static void main (String [] unbenutzt)
  { Viereck V = new Viereck ();
    V.Seite = new float [4];
    for (int i = 0; i < 4; i++) // Alle vier Seiten
werden
      V.Seite [i] = 1.0f;      // anfänglich auf die
```

**Prog. 3-3** *Array als Objektvariable*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 3-6** Array als Objektvariable

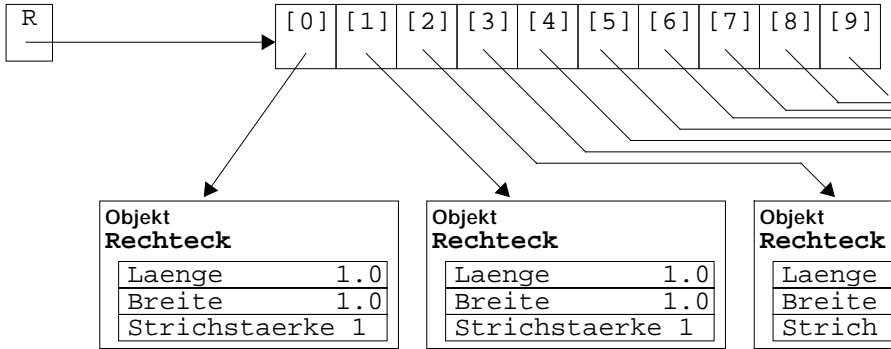
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Rechteck
{ float Laenge, Breite;   int Strichstaerke;
} // hier ohne Methode Flaeche vereinbart.

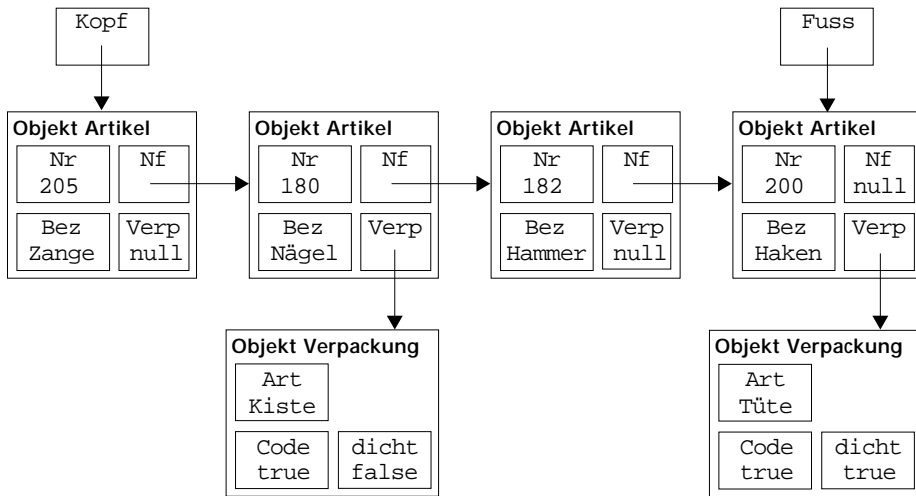
public class Hauptprogramm
{ public static void main (String [] unbenutzt)
  { int i;
    Rechteck [] R = new Rechteck [10];
    for (i = 0; i < R.length; i++)
    { R [i] = new Rechteck ();
      R [i].Laenge = 1.0f;
      R [i].Breite = 1.0f;
      R [i].Strichstaerke = 1;
    }
  }
}
```

**Prog. 3-4** Array von Verweisvariablen

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 3-7** Array von Verweisvariablen



**Abb. 3-8** Die Liste repräsentiert eingelagerte Artikel und ggf. ihre Verpackung

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

```

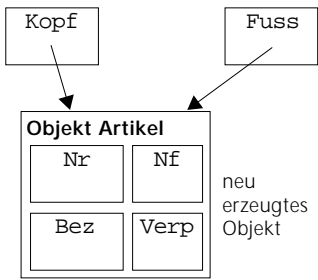
class ArtikelZeile 1
{ int Nr;   String Bez;   Verpackung Verp;   Artikel Nf;
2
}3

class Verpackung4
{ String Art;   boolean Code, dicht;5
}6

public class Lager7
{ public static void main (String [] unbenutzt)8
  { Artikel Kopf = null, Fuss = null, x;   String s;9
    int NrEin, CodeEin, dichtEin;   String BezEin, Ver-
pEin;10
    NrEin = Eingabe ();11
    while (NrEin >= 0)12
      { BezEin = EingabeString ();   VerpEin = Eingabe-
String ();13
        if (Kopf == null)14
          Kopf = Fuss = new Artikel ();15
        else 16
          { Fuss.Nf = new Artikel ();   Fuss = Fuss.Nf;17
            }18
          Fuss.Nr = NrEin;   Fuss.Bez = BezEin;   Fuss.Nf =
null;19
          if (VerpEin.equals (""))20
            Fuss.Verp = null;21
          else22
            { CodeEin = Eingabe ();   dichtEin = Eingabe ();
23
              Fuss.Verp = new Verpackung ();24
              Fuss.Verp.Art   = VerpEin;25
              Fuss.Verp.Code   = CodeEin   > 0;26
              Fuss.Verp.dicht   = dichtEin > 0;27
            }28
          NrEin = Eingabe ();29
        }30
      System.out.println ("Gelagerte Artikel:");31
      for (x = Kopf; x != null; x = x.Nf)32
        { if (x.Verp != null)33
          { s = x.Verp.Art;34

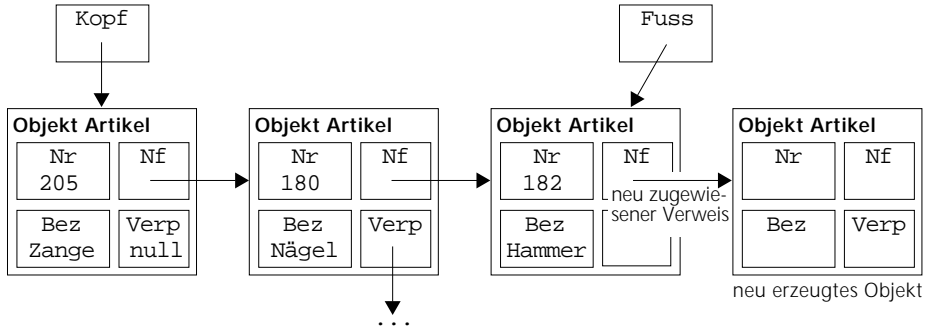
```

**Prog. 3-5** Aufbau und Ausgabe einer Liste zur Speicherung eingelagerter Artikel



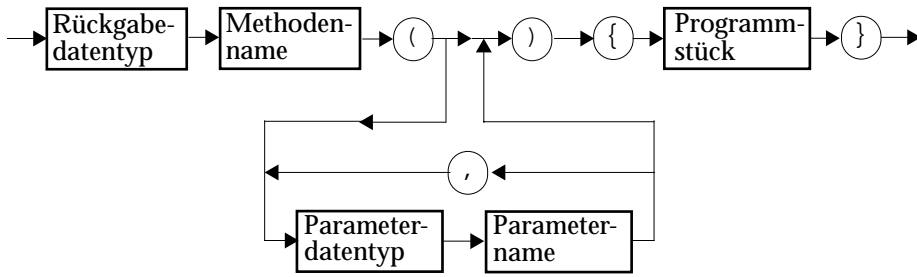
**Abb. 3-9** Liste, die aus nur einem Element besteht

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



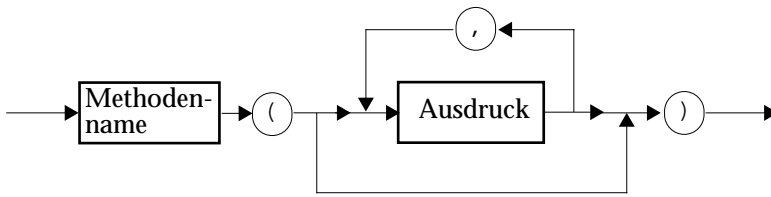
**Abb. 3-10** Die Liste repräsentiert eingelagerte Artikel und ggf. ihre Verpackung

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 3-11** Syntax zur Vereinbarung einer Methode in einer Klasse

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 3-12** Syntax eines Methodenaufrufs

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

Vereinbarte Methode mit drei formalen Parametern:

```
float Zins (float Kapital, float Zinssatz, int Jahre)
    // berechnet Zins und und Zinseszins.
{ float k = Kapital;
  for (int i = 1; i <= Jahre; i++)
      k = k + k * Zinssatz / 100;
  return k - Kapital;
}
```

Programmstück, das die Methode mit drei aktuellen Parametern aufruft:

```
float Betrag1 = 1000.0f, Betrag2 = 570.22f,
    Betrag3 = 120.0f, Gewinn;
Gewinn = Zins (Betrag1 + Betrag2, 3.5f, 4) + Betrag3;
```

└──┘  
Aufruf der Methode Zins

Beim Aufruf „automatisch“ durchgeführte Parameterübergabe:

Kapital = Betrag1 + Betrag2;
Zinssatz = 3.5f;
Jahre = 4;

**Abb. 3-13** Parameterübergabe beim Aufruf einer Methode

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

```
void Groesse (String Bez, int Wert, String Einheit)
    // Drucke eine Groesse mit Bezeichnung (40 Zeichen
    // linksbündig), Wert (10 Zeichen rechtsbündig) und
    // Einheit (linksbündig):
{ String leer = "                ";
  String b = (Bez + leer + leer).substring (0, 40);
  String w = leer + Wert;
  w = w.substring (w.length () - 10, w.length ());
```

**Prog. 3-6** Methode, die kein Ergebnis zurückliefert

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Vereinbarte Methode:

```
void f (int x)
{
    System.out.print ("  x = " + x);
    x = 7;  System.out.print ("  x = " + x);
}
```

Aufruf der Methode:

```
int x = 2, y;
f (x);
y = x;  System.out.println ("  y = " + y);
```

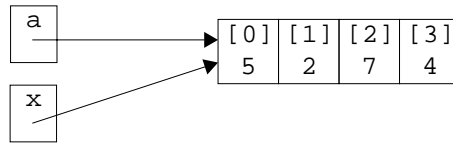
Ausgabe des Programms:

```
x = 2  x = 7  y = 2
```

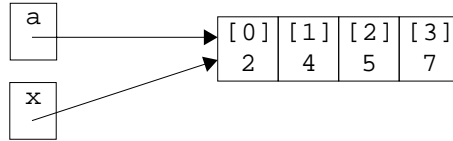
### **Prog. 3-7** Zuweisung an die Parameter-Variable im Unterprogramm

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

Nach der Parameter-  
übergabe:



Unmittelbar vor der  
Terminierung der  
Methode `sortiere`:



**Abb. 3-14** Array von Verweisvariablen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Vereinbarte Methode:

```
void sortiere (int [] x)
{ for (int i = 0; i < x.length - 1; i++)
  for (int j = i + 1; j < x.length; j++)
    if (x [i] > x [j])
      { int z = x [i]; x [i] = x [j]; x [j] = z;
      }
}
```

Aufruf der Methode:

```
int [] a = new int [4];
a [0] = 5; a [1] = 2; a [2] = 7; a [3] = 4;
sortiere (a);
... ←————— Hier kann mit dem sortierten Array a gearbeitet werden
```

**Prog. 3–8** Parameterübergabe, wenn der formale Parameter eine Array-Variable ist

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Vereinbarte Methode:

```

Rechteck Quadrat (Rechteck gross, Rechteck klein)
{ int dL = gross.Laenge - klein.Laenge,
    dB = gross.Breite - klein.Breite, SeiteQ;
  Rechteck Q;
  gross.Strichstaerke = 1;
  klein.Strichstaerke = 1;
  if (dL > 0 && dB > 0)
  // Kleines Rechteck im großen enthalten:
  { if      (gross.Laenge <= dB) SeiteQ = gross.Laenge;
    else if (gross.Breite <= dL) SeiteQ = gross.Breite;
    else if (dB <= dL)          SeiteQ = dL;
    else                        SeiteQ = dB;
    Q = new Rechteck ();
    Q.Laenge = SeiteQ; Q.Breite = SeiteQ;
    Q.Strichstaerke = 3;
    return Q;
  }
  else
    return null;
}

```

Aufruf der Methode:

```

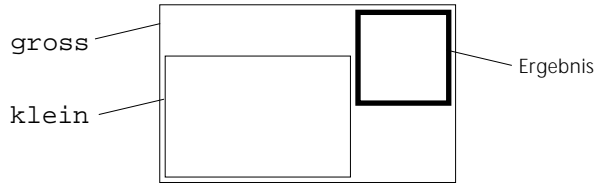
Rechteck R1, R2, R3;
R1 = new Rechteck (); ...
R2 = new Rechteck (); ...
R3 = Quadrat (R1, R2);

```

← Hier nicht dargestellt:  
← Zuweisung von Werten  
an die Objektvariablen

**Prog. 3–9** Parameterübergabe, wenn die formalen Parameter Verweis-Variablen sind

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 3-15** Rechtecke zu Prog. 3-9

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Rechteck
{ float Laenge, Breite;   int Strichstaerke;
  void setzeLaenge (float L)
{ Laenge = L;
}
  float liesLaenge ()
{ return Laenge;
}
  void setzeBreite (float B)
{ Breite = B;
}
  float liesBreite ()
{ return Breite;
}
  void setzeStrich (int s)
{ Strichstaerke = s;
}
  float liesStrich ()
{ return Strichstaerke;
}
  void umrande (float b)
{ Laenge += 2 * b;   Breite += 2 * b;
}
  float Flaeche ()
  { return Laenge * Breite;
  }
}
```

**Prog. 3-10** Kapselung der Objekt-Variablen durch Zugriffsmethoden

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Vereinbarung der Klasse `Rechteck`:

```
class Rechteck
{ float Laenge, Breite;   int Strichstaerke;

  Rechteck ()
  { Laenge = Breite = 1.0f;   Strichstaerke = 1;
  }

  Rechteck (float Kantenlaenge)
  { Laenge = Breite = Kantenlaenge;   Strichstaerke = 1;
  }

  Rechteck (float L, float B)
  { Laenge = L;   Breite = B;   Strichstaerke = 1;
  }

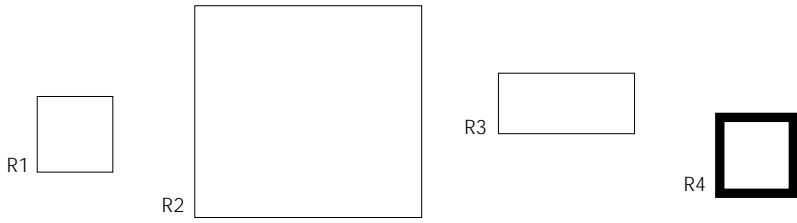
  Rechteck (int s)
  { Laenge = Breite = 1.0f;   Strichstaerke = s;
  }
}
```

Aufrufe zur Erzeugung von Objekten der Klasse `Rechteck`:

```
Rechteck R1, R2, R3, R4;
R1 = new Rechteck ();
R2 = new Rechteck (3.0f);
R3 = new Rechteck (1.8f, 0.8f);
R4 = new Rechteck (3);
```

### **Prog. 3-11** Konstruktoren der Klasse `Rechteck`

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 3-16** Von Prog. 3-11 erzeugte Rechtecke

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class Rechteck
{ float Laenge, Breite;   int Strichstaerke;

  Rechteck ()
  { Laenge = Breite = 1.0f;   Strichstaerke = 1;
  }

  Rechteck (float Laenge)
  { this.Laenge = this.Breite = Laenge;
    Strichstaerke = 1;
  }

  Rechteck (float Laenge, float Breite)
  { this.Laenge = Laenge;   this.Breite = Breite;
    Strichstaerke = 1;
  }

  Rechteck (int Strichstaerke)
  { Laenge = Breite = 1.0f;
    this.Strichstaerke = Strichstaerke;
  }
}
```

**Prog. 3-12** *Konstruktoren, die den Selbstverweis this benutzen*

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
class Artikelzeile 1
{ int Nr;   String Bez;   Artikel Nf;2
  Artikel (int Nr, String Bez, Lager L)3
  { this.Nr = Nr;   this.Bez = Bez;   Nf = null;4
    if (L.Kopf == null)5
      L.Kopf = L.Fuss = this;6
    else7
      { L.Fuss.Nf = this;   L.Fuss = this;8
        }9
      }10
  }11
```

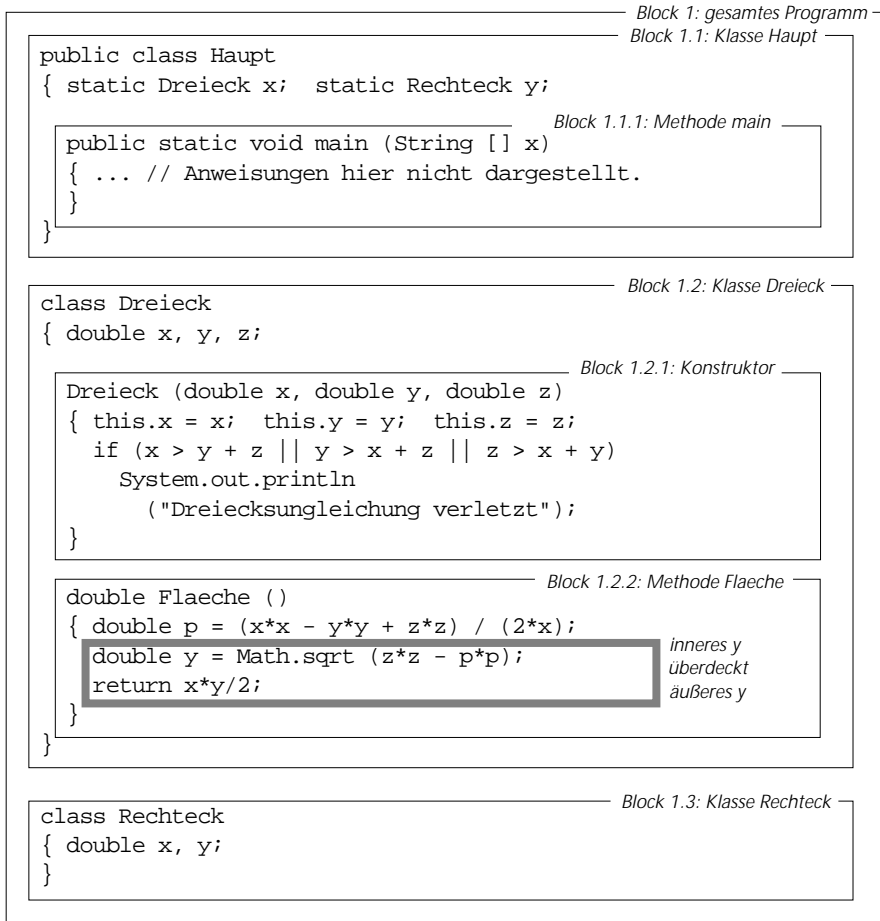
**Prog. 3-13** Konstruktor fügt neues Objekt an eine Liste an

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Rechteck
{ float Laenge, Breite;   int Strichstaerke;
  Rechteck (Rechteck Original)
  { if (Original != null)
    { Laenge = Original.Laenge; Breite = Original.Breite;
      Strichstaerke = Original.Strichstaerke;
    }
    else // kein Original vorhanden:
    { Laenge = Breite = 1.0f;   Strichstaerke = 1;
    }
  }
}
```

**Prog. 3-14** Neues Objekt soll Kopie eines Originalobjekts sein

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



### Prog. 3-15 Geschachtelte Struktur eines Programms

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```

class R
{ float x, y;
}

public class Katastrophe
{
    R R (float x, float y)
    {
        R r = new R (); r.x = x; r.y = y;
        return r;
    }

    public static void main (String [] unbenutzt)
    { R[] r = new R[2];
      for (int R = 0; R < 2; R++)
      { r[R] = new R (); r [R].x = R; r[R].y = R;
      }
      System.out.println (r[0].x + ", " + r[0].y);
      System.out.println (r[1].x + ", " + r[1].y);
    }
}

```

← Hier ist R die Bezeichnung einer Klasse.

Hier wird eine Methode namens R vereinbart.

Die Methode liefert ein Objekt der Klasse R.

Verweis-Variablen r zeigt auf neues Objekt der Klasse R.

Array von Objekten der Klasse R.

Integer-Variablen R vereinbart.

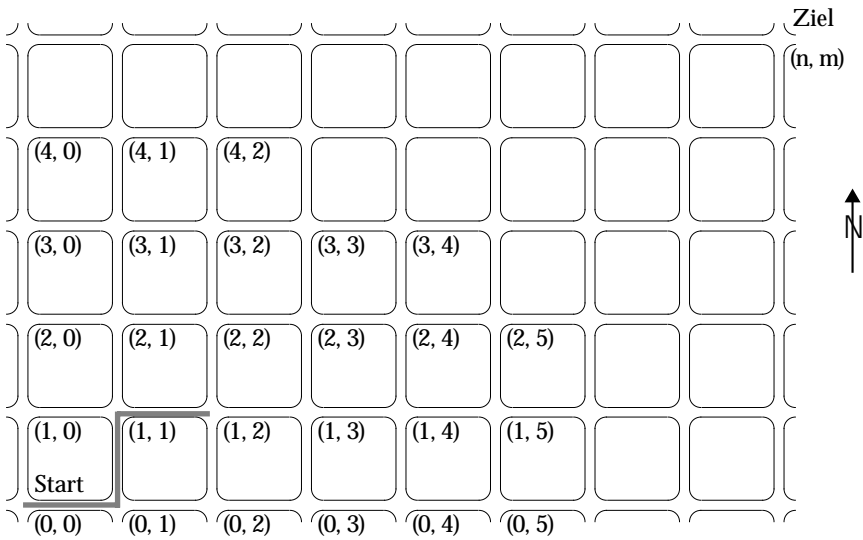
**Ausgabe:**

0, 0

1, 1

**Prog. 3-16** Korrekt, aber irreführend: sinnlose Mehrfachverwendung des Namens R

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000



**Abb. 4-1** Kansas-City-Problem: Weg 2 ist durch eine graue Linie markiert

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

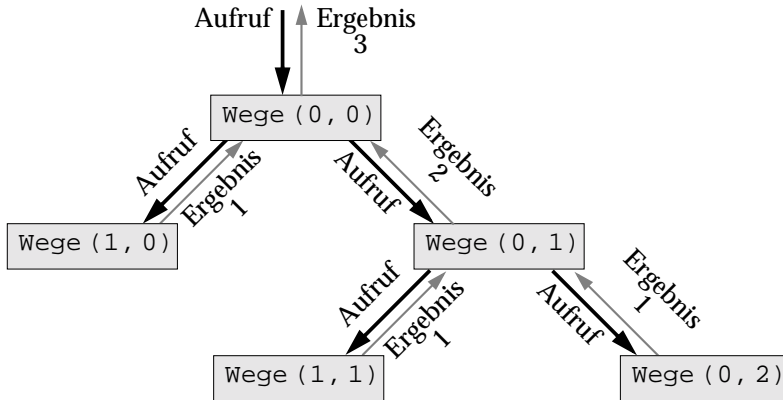
```
public class Hauptprogramm
{ static int n, m;

    public static void main (String [] unbenutzt)
    { n = Eingabe ();    m = Eingabe ();
      int w = Wege (0, 0);
      System.out.println (w + " kürzeste Wege");
    }

    static int Wege (int i, int j)
    { if (i == n || j == m)    return 1;
      else    return Wege (i + 1, j) + Wege (i, j + 1);
    }
}
```

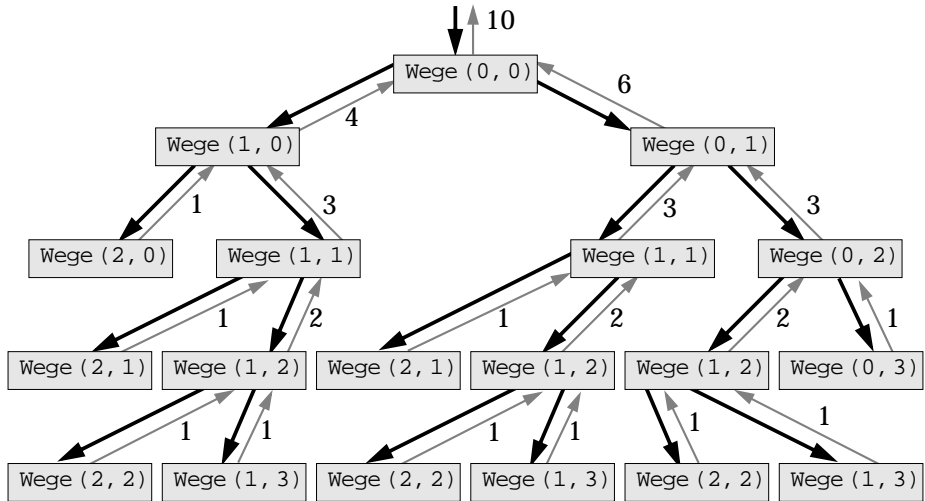
**Prog. 4-1** Rekursive Lösung des Kansas-City-Problems

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



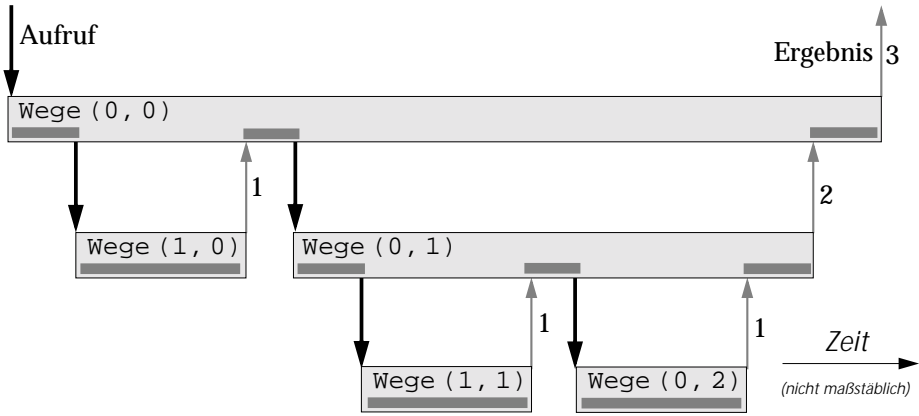
**Abb. 4-2** Ausführungen von Wege bei  $n = 1$  und  $m = 2$

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000



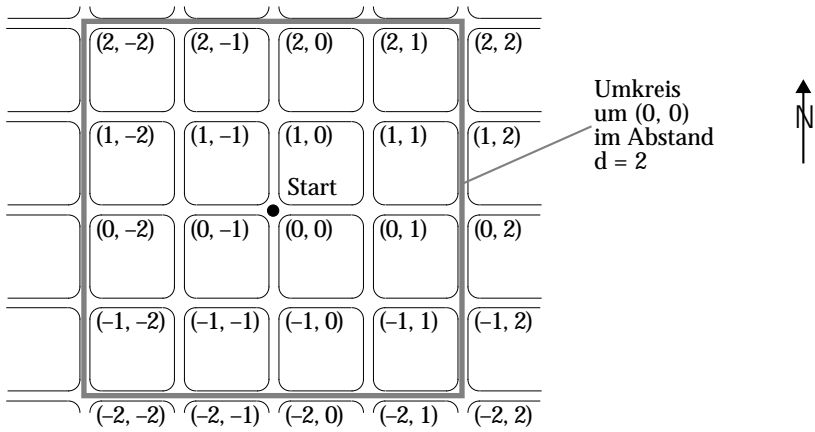
**Abb. 4-3** Ausführungen von Wege bei  $n = 2$  und  $m = 3$

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 4-4** Reihenfolge der Ausführungen von `Wege` bei  $n = 1$  und  $m = 2$   
 (Ausführungsabschnitte sind durch dunkelgraue Balken markiert)

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000



**Abb. 4-5** Chicago-Problem für  $d = 2$

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
static int Wege (int i, int j)
{ if (i == -d || i == d || j == -d || j == d)
    return 1;
  else
    return Wege (i + 1, j) + Wege (i - 1, j)
           + Wege (i, j + 1) + Wege (i, j - 1);
```

**Prog. 4-2** Keine Lösung des Chicago-Problems !

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

---

```
int ggT (int x, int y)
{ if (x == y)      return x;
  else if (x < y) return ggT (x, y - x);
  else           return ggT (x - y, y);
```

**Prog. 4-3** *Rekursive Berechnung des größten gemeinsamen Teilers von x und y*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

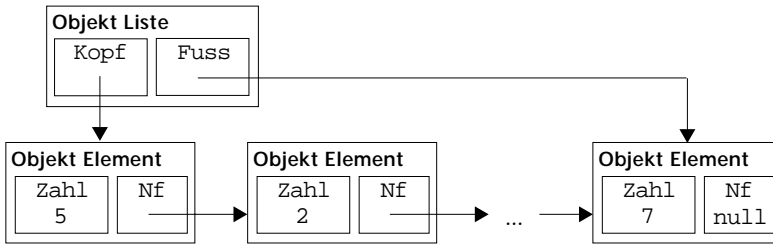
```
float Wurzel (float a, float b, float x)
{ float m = (a + b)/2, epsilon = 0.001f;
  if (b - a < epsilon) return m;
  else if (m * m < x) return Wurzel (m, b, x);
  else return Wurzel (a, m, x);
}
```

Beispiel eines Aufrufs der Methode Wurzel:

```
float q = 5.0f, w = Wurzel (0.0f, q + 1.0f, q);
System.out.println ("Wurzel " + q + " beträgt " + w);
```

**Prog. 4-4** Berechnung von  $\sqrt{x}$  durch rekursiv implementierte Intervallschachtelung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 4-6** Durch Verweis-Variablen rekursiv realisierte Liste

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class Element
{ int Zahl;    Element Nf;

  Element (int Zahl, Element Nf)
  { this.Zahl = Zahl;    this.Nf = Nf;
  }
}
```

**Prog. 4-5** Objekte der Klasse `Element` repräsentieren je ein Listenelement

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Liste
{ Element Kopf, Fuss;

  Liste ()
  { Kopf = Fuss = null;
  }

  Element suche (int Zahl)
  { return suche (Zahl, Kopf);
  }

  Element suche (int Zahl, Element e)
  { if      (e == null)      return null;
    else if (e.Zahl == Zahl) return e;
    else                return suche (Zahl, e.Nf);
  }

  void durchlaufe ()
  { System.out.print ("Liste ( "); durchlaufe (Kopf);
    System.out.println (");" );
  }

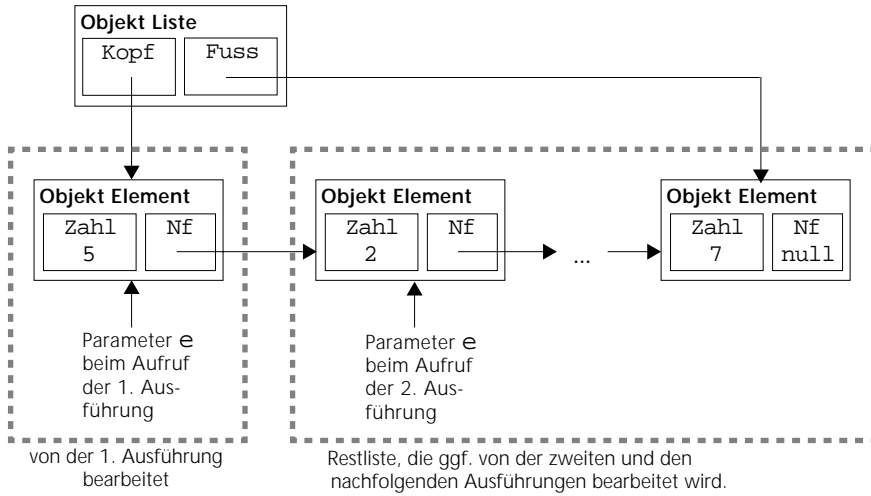
  void durchlaufe (Element e)
  { if (e != null) { System.out.print (e.Zahl + " ");
                    durchlaufe (e.Nf);
                  }
  }

  void durchlaufe_rueckwaerts ()
  { System.out.print ("Liste rückwärts ( ");
    durchlaufe_rueckwaerts (Kopf); System.out.println (");" );
  }

  void durchlaufe_rueckwaerts (Element e)
  { if (e != null) { durchlaufe_rueckwaerts (e.Nf);
                    System.out.print (e.Zahl + " ");
                  }
  }
}
```

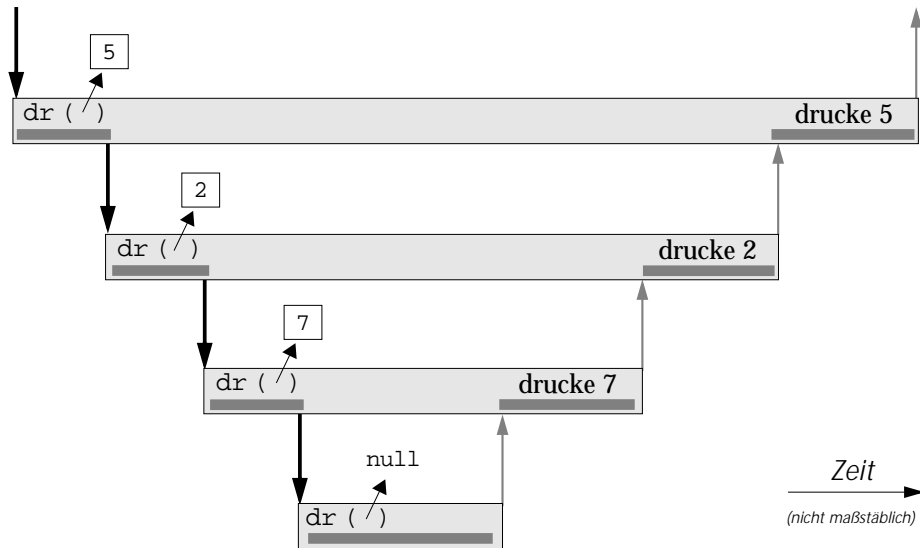
#### **Prog. 4-6** Klasse Liste, Teil 1

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 4-7** Rekursive Suche eines Elements

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 4-8** Ausführung der rekursiven Methode `durchlaufe_rueckwaerts`, hier mit `dr` abgekürzt

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
void erzeuge_Kopf (int Zahl)
{ if (Kopf == null) Kopf = Fuss = new Element (Zahl, null);
  else                Kopf = new Element (Zahl, Kopf);
}

void erzeuge_Fuss (int Zahl)
{ if (Kopf == null) Kopf = Fuss = new Element (Zahl, null);
  else                Fuss = Fuss.Nf = new Element (Zahl, null);
}

void erzeuge_Nf (int Zahl_in_Liste, int Zahl)
{ Element e = suche (Zahl_in_Liste);
  if (e != null)
  { e.Nf = new Element (Zahl, e.Nf);   if (e == Fuss) Fuss = e.Nf;
  }
}

void sortiere_ein (int Zahl)
{ Kopf = sortiere_ein (Zahl, Kopf);
}

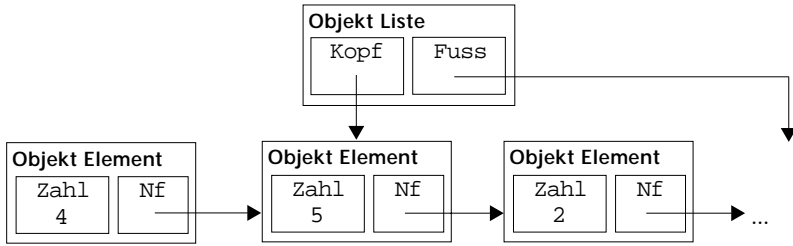
Element sortiere_ein (int Zahl, Element e)
{ if (e == null)
  return Fuss = new Element (Zahl, null);
  else if (Zahl < e.Zahl)
  return new Element (Zahl, e);
  else
  { e.Nf = sortiere_ein (Zahl, e.Nf); return e;
  }
}

void entferne (int Zahl)
{ Kopf = entferne (Zahl, Kopf);
}

Element entferne (int Zahl, Element e)
{ if (e != null)
  if (e.Zahl == Zahl)
  return e.Nf;
  else
  { e.Nf = entferne (Zahl, e.Nf); return e;
  }
  else return null;
}

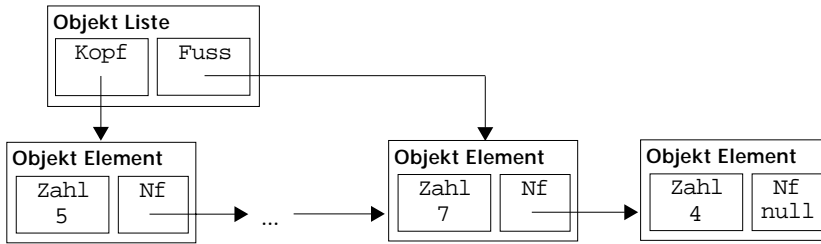
void entferne_alle ()
{ Kopf = Fuss = null;
}
}
```

**Prog. 4-7** Klasse Liste, Teil 2



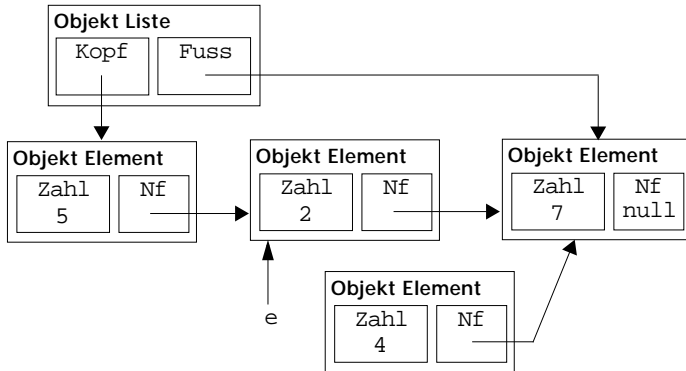
**Abb. 4-9** Zwischenzustand beim Einfügen eines neuen Elements am Anfang der Liste

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



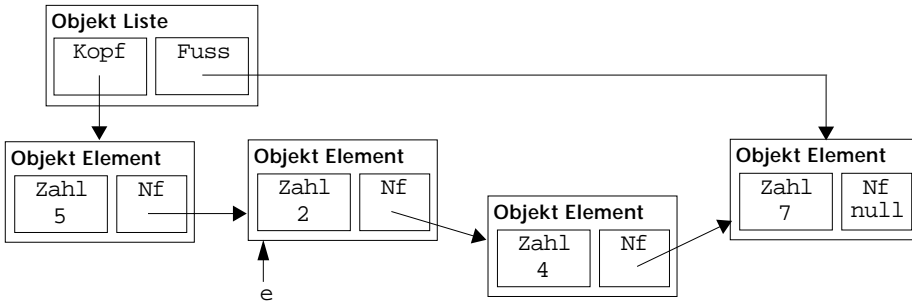
**Abb. 4-10** Zwischenzustand beim Einfügen eines neuen Elements am Ende der Liste

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



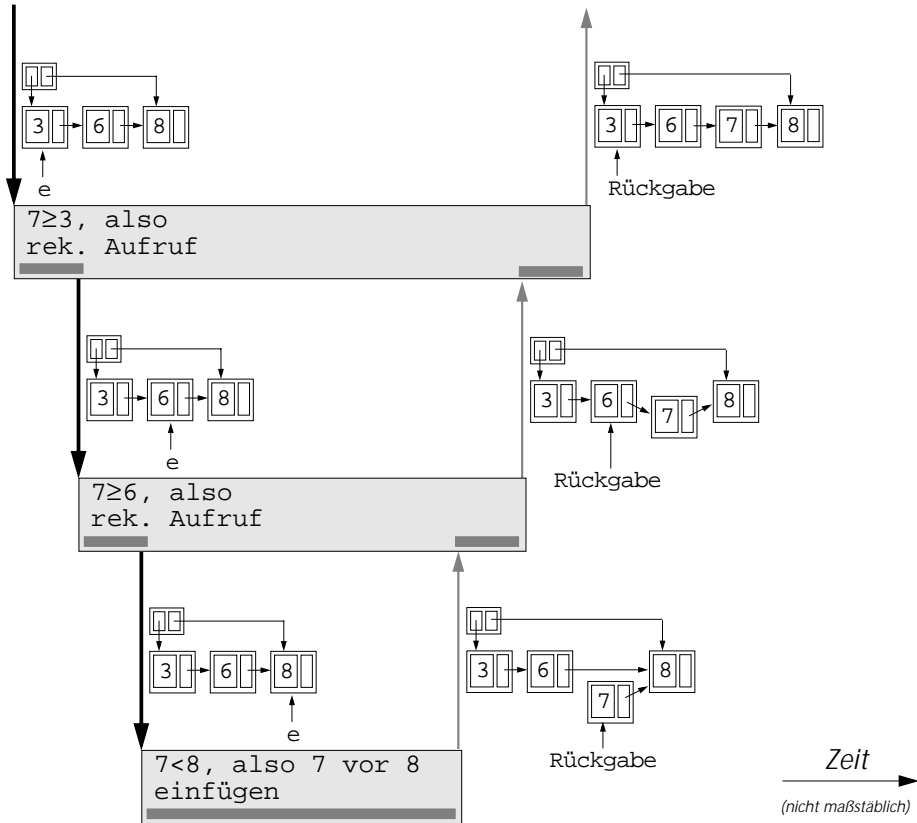
**Abb. 4-11** Zwischenzustand beim Einfügen eines neuen Elements nach dem Element *e*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 4-12** Endzustand beim Einfügen eines neuen Elements nach dem Element *e*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



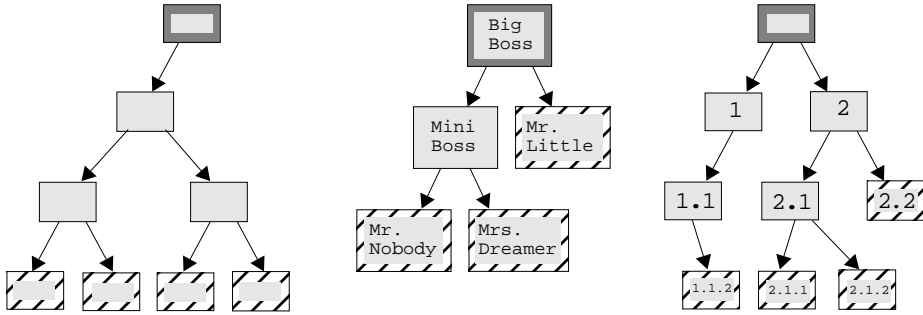
**Abb. 4-13** Einsortieren eines neuen Elements mit dem Zahlenwert 7

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
public class Haupt
{ public static void main (String [] unbenutzt)
  { Liste L = new Liste ();
    L.erzeuge_Fuss (25); L.erzeuge_Kopf (28); L.erzeuge_Fuss (14);
    L.durchlaufe ();      L.durchlaufe_rueckwaerts ();
    L.entferne_alle ();   L.durchlaufe ();
    L.sortiere_ein (15);  L.sortiere_ein (25); L.sortiere_ein (10);
    L.durchlaufe ();      L.erzeuge_Nf (15, 17); L.durchlaufe ();
    Element e = L.suche (15);
    System.out.println ("gefunden: " + e.Zahl);
    L.entferne (25);      L.durchlaufe ();
  }
}
```

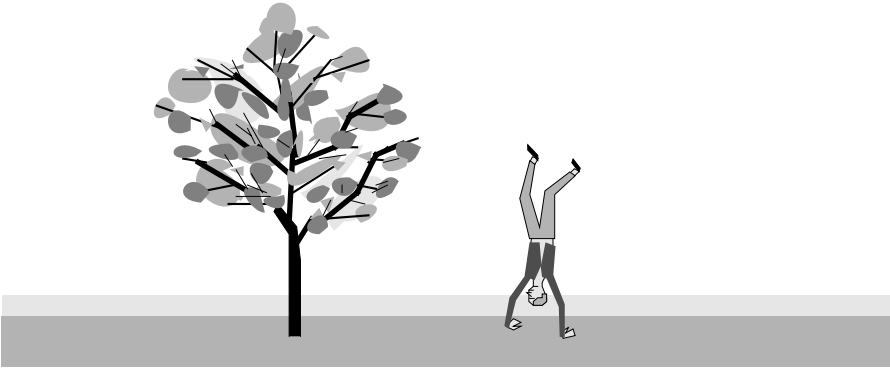
**Prog. 4–8** Hauptprogramm, das Methoden der Klasse Liste aufruft

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 4-14** Drei Binärbäume

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 4-15** Ein Baum und ein Informatiker

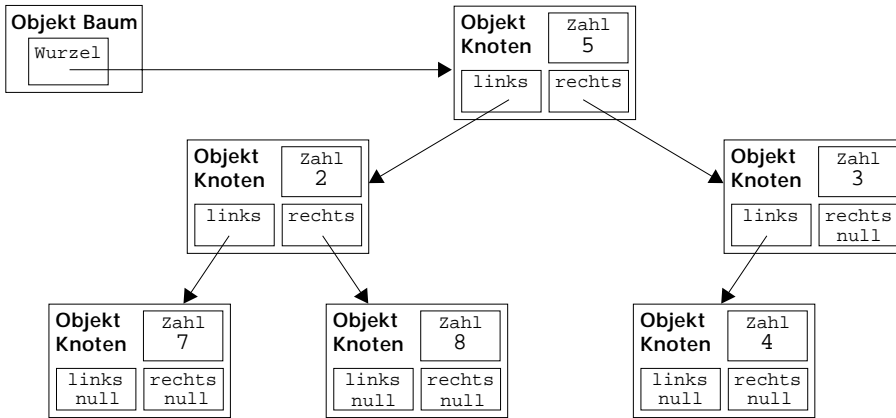
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Knoten
{ int Zahl;  Knoten links, rechts;

  Knoten (int Zahl)
  { this.Zahl = Zahl;  links = rechts = null;
  }
}
```

**Prog. 4-9** Objekte der Klasse `Knoten` repräsentieren je einen Knoten des Binärbaums

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 4-16** Realisierung eines Binärbaums durch Objekte der Klasse `Knoten`

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```

class Baum
{ Knoten wurzel;

  Baum ()
  { wurzel = null;
  }

  Knoten suche (int Zahl)
  { return suche (Zahl, wurzel);
  }

  Knoten suche (int Zahl, Knoten k)
  { if (k == null) return null;
    else if (k.Zahl == Zahl) return k;
    else { Knoten s = suche (Zahl, k.links);
          if (s != null) return s;
          else return suche (Zahl, k.rechts);
        }
  }

  void erzeuge_Blatt (int Zahl)
  { wurzel = erzeuge_Blatt (Zahl, wurzel);
  }

  Knoten erzeuge_Blatt (int Zahl, Knoten k)
  { if (k == null)
    return new Knoten (Zahl);
    else if (Zahl < k.Zahl)
    { k.links = erzeuge_Blatt (Zahl, k.links); return k;
    }
    else
    { k.rechts = erzeuge_Blatt (Zahl, k.rechts); return k;
    }
  }

  void durchlaufe ()
  { System.out.println ("Baum:"); durchlaufe (wurzel, 0);
  }

  void durchlaufe (Knoten k, int Einrueckung)
  { if (k != null)
    { durchlaufe (k.links, Einrueckung + 2);
      for (int i = 1; i <= Einrueckung; i++)
        System.out.print (" ");
      System.out.println (k.Zahl);
      durchlaufe (k.rechts, Einrueckung + 2);
    }
  }
}

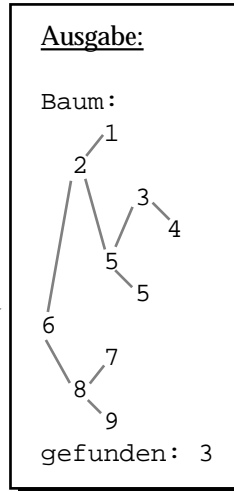
```

**Prog. 4-10** Klasse Baum

```

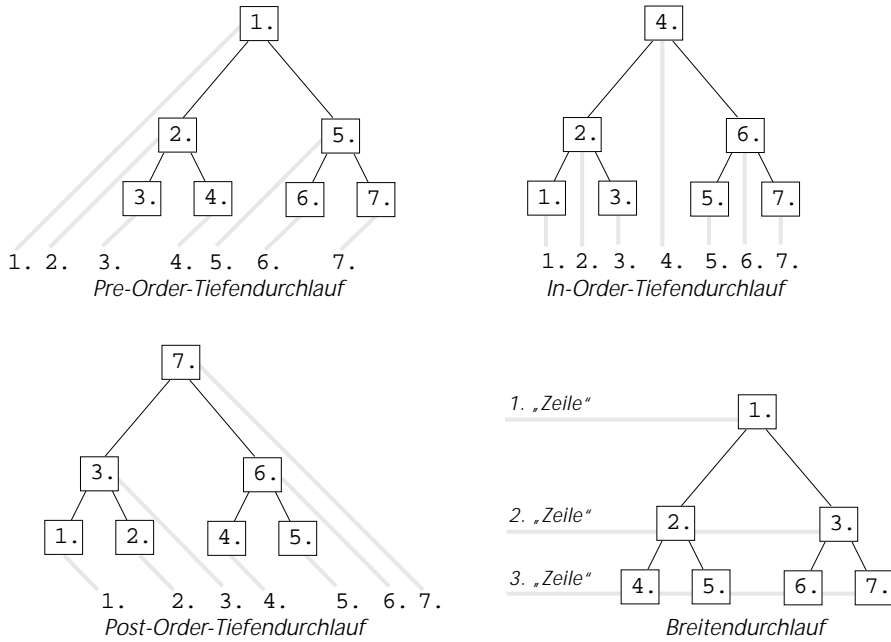
public static void main (String [] unbenutzt)
{
    Baum B = new Baum ();
    B.erzeuge_Blatt (6);
    B.erzeuge_Blatt (2);
    B.erzeuge_Blatt (5);
    B.erzeuge_Blatt (8);
    B.erzeuge_Blatt (7);
    B.erzeuge_Blatt (1);
    B.erzeuge_Blatt (3);
    B.erzeuge_Blatt (9);
    B.erzeuge_Blatt (4);
    B.erzeuge_Blatt (5);
    B.durchlaufe ();
    Knoten k = B.suche (3);
    if (k != null)
        System.out.println ("gefunden:" + k.Zahl);
    else
        System.out.println ("nicht gefunden: 3");
}

```



**Prog. 4-11** Aufbau eines Binärbaumes und erzeugte Ausgabe

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 4-17** Arten des Durchlaufs durch einen Binärbaum

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

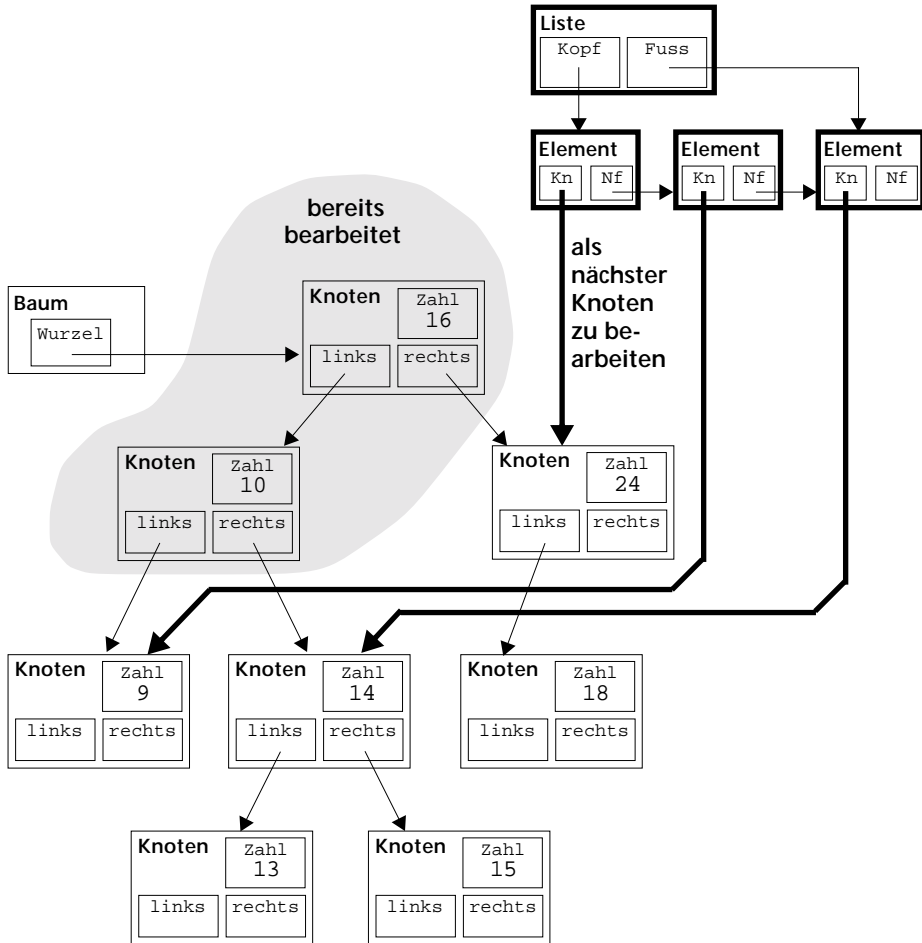
```
void Pre_Order_Tiefendurchlauf (Knoten k)
{ if (k != null)
  { System.out.println (k.Zahl);           // 1. Knoten k
    Pre_Order_Tiefendurchlauf (k.links);   // 2. Linker Unterbaum
    Pre_Order_Tiefendurchlauf (k.rechts);  // 3. Rechter Unterbaum
  }
}

void In_Order_Tiefendurchlauf (Knoten k)
{ if (k != null)
  { In_Order_Tiefendurchlauf (k.links);    // 1. Linker Unterbaum
    System.out.println (k.Zahl);          // 2. Knoten k
    In_Order_Tiefendurchlauf (k.rechts);  // 3. Rechter Unterbaum
  }
}

void Post_Order_Tiefendurchlauf (Knoten k)
{ if (k != null)
  { Post_Order_Tiefendurchlauf (k.links);  // 1. Linker Unterbaum
    Post_Order_Tiefendurchlauf (k.rechts); // 2. Rechter Unterbaum
    System.out.println (k.Zahl);          // 3. Knoten k
  }
}
```

**Prog. 4-12** Methoden des Tiefendurchlaufs durch einen Binärbaum

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 4-18** Breitendurchlauf durch einen Binärbaum

```

class Element
{ Knoten Kn;  Element Nf;

  Element (Knoten Kn, Element Nf)
  { this.Kn = Kn;  this.Nf = Nf;
  }
}

class Liste
{ Element Kopf, Fuss;

  Liste ()
  { Kopf = Fuss = null;
  }

  void erzeuge_Fuss (Knoten k)
  { if (k != null)
    if (Kopf == null) Kopf = Fuss = new Element (k, null);
    else               Fuss = Fuss.Nf = new Element (k, null);
  }

  void entferne_Kopf ()
  { if (Kopf != null && Kopf != Fuss) Kopf = Kopf.Nf;
    else                               Kopf = Fuss = null;
  }
}

class Baum
{ ...

  void Breitendurchlauf ()
  { Liste L = new Liste ();
    L.erzeuge_Fuss (Wurzel);
    while (L.Kopf != null)
    { System.out.println (L.Kopf.Kn.Zahl);
      L.erzeuge_Fuss (L.Kopf.Kn.links);
      L.erzeuge_Fuss (L.Kopf.Kn.rechts);
      L.entferne_Kopf ();
    }
  }
}

```

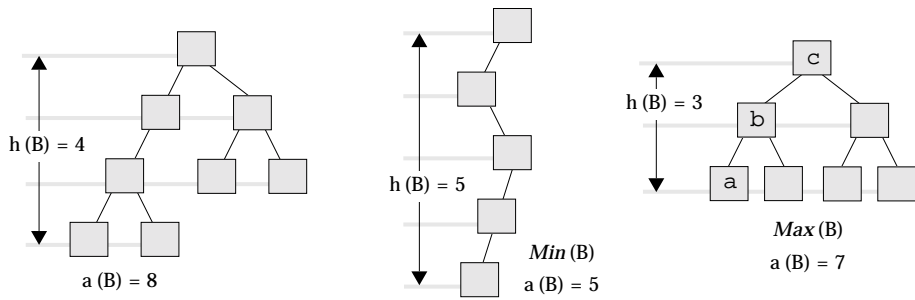
**Ausgabe:**

```

16
10
24
9
14
18
13
15

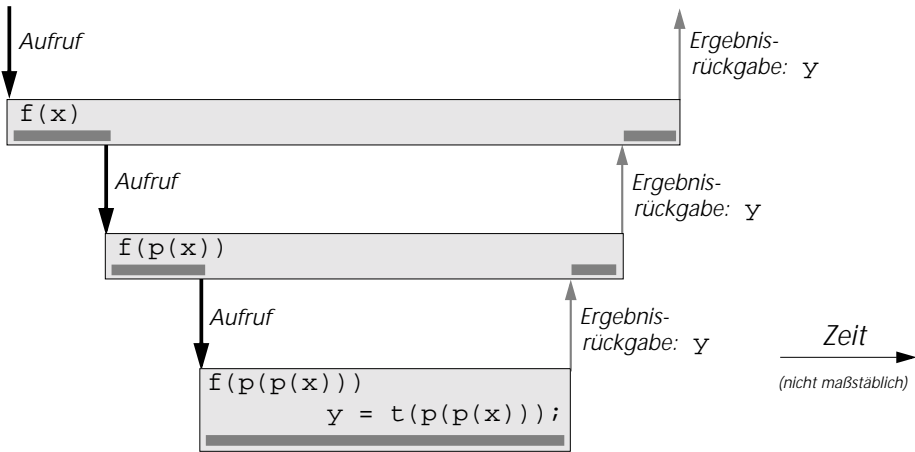
```

**Prog. 4-13** *Breitendurchlauf durch einen Binärbaum*



**Abb. 4-19** Höhe und Knotenanzahl von Binärbäumen

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000



**Abb. 4-20** Schlichte Rekursion

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

```

int ggT (int x1, int x2)
{ int y1 = x1, y2 = x2, z1, z2;
  while ( y1 != y2 )
  { if (y1 > y2) { z1 = y1 - y2; z2 = y2; }
    else       { z1 = y1; z2 = y2 - y1; }
    y1 = z1; y2 = z2;
  }
  return y1;
}

```

Entsprechungen im  
Pseudoprogramm:

B(x)

z = p(y);

y = z;

t(y)

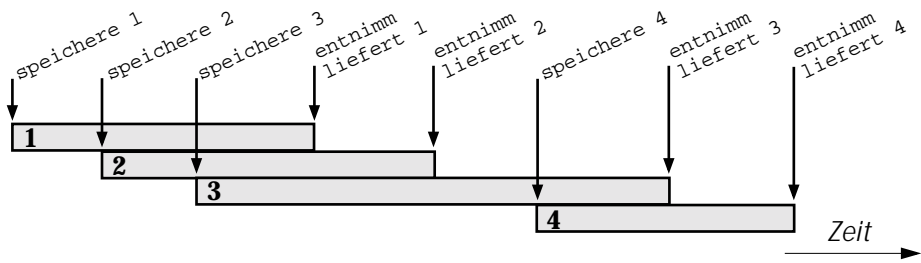
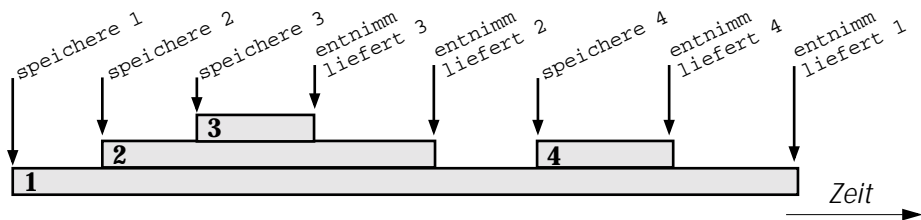
**Prog. 4-14** Iterative Berechnung des größten gemeinsamen Teilers

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
String Wort (String Zeile, int WortNr)
{ if (Zeile == null || WortNr < 1)
  return "";
  String Kern = Zeile.trim ();
  int LeerIndex = Kern.indexOf (" ");
  if (LeerIndex < 0)
    if (WortNr == 1) return Kern;
    else           return "";
  else
    if (WortNr == 1)
      return Kern.substring (0, LeerIndex);
    else
      { String Rest = Kern.substring (LeerIndex, Kern.length ());
        return Wort (Rest, WortNr - 1);
      }
}
```

**Prog. 5-1** Lesen eines Wortes aus einer Zeile

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

**Puffer: FIFO****Stapel: LIFO****Abb. 5-1** Puffer und Stapel

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class Puffer
{ Liste L;

  Puffer () // Konstruktor.
  { L = new Liste ();
  }

  void speichere (int Zahl)
  { if (Zahl >= 0) L.erzeuge_Fuss (Zahl);
  }

  int entnimm ()
  { Element e = L.entnimm ();
    if (e != null) return e.Zahl;
    else          return -1;
  }
}
```

**Prog. 5-2** Puffer

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Stapel
{ Liste L;

  Stapel () // Konstruktor.
  { L = new Liste ();
  }

  void speichere (int Zahl)
  { if (Zahl >= 0) L.erzeuge_Kopf (Zahl);
  }

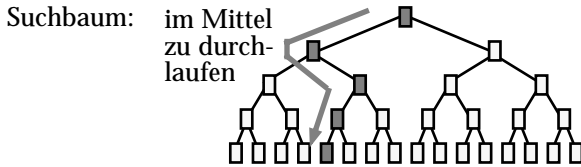
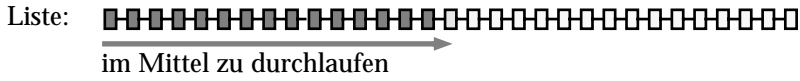
  int entnimm ()
  { Element e = L.entnimm ();
    if (e != null) return e.Zahl;
    else          return -1;
  }
}
```

**Prog. 5-3** *Stapel*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Anzahl der Einträge	Liste	Suchbaum
7	3,5	3
31	15,5	5
127	63,5	7
1023	511,5	10
16383	8191,5	14

veranschaulicht



**Abb. 5-2** Suchaufwand in einer Liste und einem Suchbaum

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

---

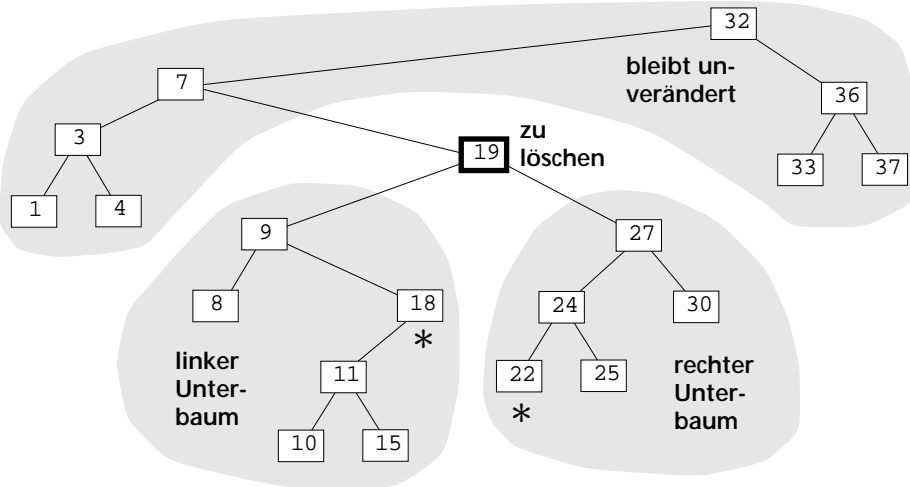
```
Knoten suche (int Zahl)
{ return suche (Zahl, Wurzel);
}
```

```
Knoten suche (int Zahl, Knoten k)
{ if      (k == null)      return null;
  else if (Zahl < k.Zahl)  return suche (Zahl, k.links);
  else if (Zahl > k.Zahl)  return suche (Zahl, k.rechts);
  else                    return k;
}
```

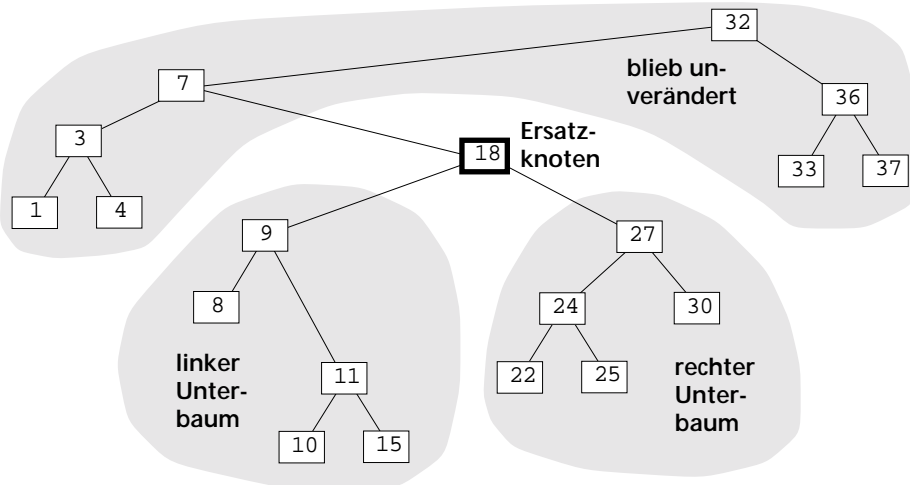
**Prog. 5-4** *Methoden zur Suche in einem Suchbaum*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

Vor dem Löschen:



Nach dem Löschen:



**Abb. 5-3** Vor (oben) und nach (unten) dem Löschen eines Knotens aus einem Suchbaum (Die Objektvariablen sind nicht vollständig dargestellt.)

```
void entferne (int Zahl)
{ Wurzel = entferne (Zahl, Wurzel);
}
```

```
Knoten entferne (int Zahl, Knoten k)
```

<pre>{ if (k == null)   return null;   else if (Zahl &lt; k.Zahl)   { k.links = entferne (Zahl, k.links); return k;   }   else if (Zahl &gt; k.Zahl)   { k.rechts = entferne (Zahl, k.rechts); return k;   } }</pre>	Suche des zu löschenden Knotens
<pre>else if (k.links == null)   return k.rechts; else if (k.rechts == null)   return k.links;</pre>	

```
else
{ Knoten Ersatz = k.links;
  while (Ersatz.rechts != null) Ersatz = Ersatz.rechts;
  Ersatz.links = entferne (Ersatz.Zahl, k.links);
  Ersatz.rechts = k.rechts; k.links = k.rechts = null;
  return Ersatz;
}
}
```

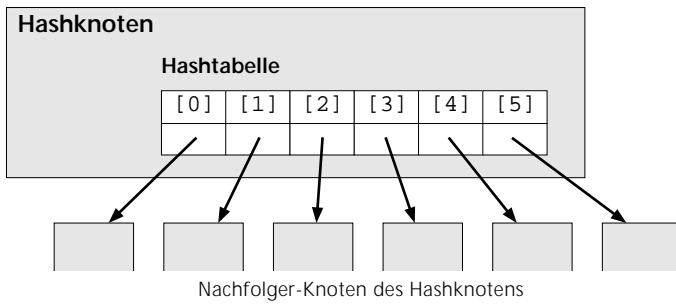
### **Prog. 5-5** Löschen eines Knotens aus einem Suchbaum

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
public class Haupt
{
    public static void main (String [] unbenutzt)
    { char Steuerzeichen; String Eingabe, Text;
      Baum Suchbaum = new Baum (); Knoten gefunden;
      do { Eingabe = EingabeString ();
          System.out.println ("Eingabe: " + Eingabe);
          if (Eingabe.length () < 2)
              Steuerzeichen = '.';
          else
          { Steuerzeichen = Eingabe.charAt (0);
            Text = Eingabe.substring (1, Eingabe.length ());
            gefunden = Suchbaum.suche (Text);
            if (gefunden != null)
                System.out.println ("déjà vu: " + gefunden.Text);
            if (Steuerzeichen == '+' && gefunden == null)
                Suchbaum.erzeuge_Blatt (Text);
            else if (Steuerzeichen == '-')
                Suchbaum.entferne (Text);
            }
          }
      while (Steuerzeichen != '.');
      System.out.println ("Alle Knoten des Suchbaums:");
      Suchbaum.durchlaufe ();
    }
}
```

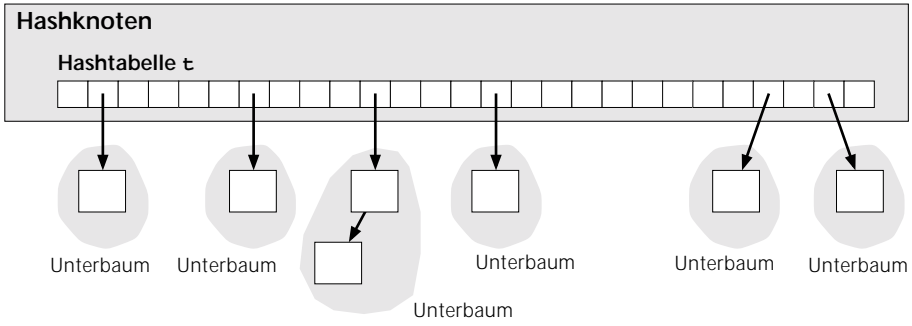
### **Prog. 5–6** Einfache Anwendung eines Suchbaums

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 5-4** Hashtabelle in einem Hashknoten

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 5-5** Große Hashtabelle an der Wurzel, viele Unterbäume sind leer

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Knoten
{ String Eintrag; Knoten links, rechts;

  Knoten (String Eintrag)
  { this.Eintrag = Eintrag; links = rechts = null;
  }
}
```

**Prog. 5-7** Knoten, der einen Text speichert

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

class Hashknoten
{ Knoten [] t; int n, signifikant;

  Hashknoten (int n, int signifikant)
  { this.n = n; this.signifikant = signifikant;
    t = new Knoten [n];
    for (int j = 0; j < n; j++) t [j] = null;
  }

  int h (String Eintrag) // Hashfunktion
  { int i = 0, f = 503,
    m = Math.min (Eintrag.length (), signifikant);
    for (int j=0; j<m; j++) i = f * i + (int) Eintrag.charAt (j);
    return Math.abs (i) % n;
  }

  void sortiere_ein (String Eintrag)
  { int i = h (Eintrag);
    t [i] = sortiere_ein (Eintrag, t [i]);
  }

  Knoten sortiere_ein (String Eintrag, Knoten k)
  { if (k == null)
    return new Knoten (Eintrag);
    else if (Eintrag.compareTo (k.Eintrag) < 0)
    { k.links = sortiere_ein (Eintrag, k.links); return k;
    }
    else
    { k.rechts = sortiere_ein (Eintrag, k.rechts); return k;
    }
  }

  Knoten suche (String Eintrag)
  { return suche (Eintrag, t [h (Eintrag)]);
  }

  Knoten suche (String Eintrag, Knoten k)
  { if (k == null)
    return null;
    else if (Eintrag.equals (k.Eintrag))
    return k;
    else if (Eintrag.compareTo (k.Eintrag) < 0)
    return suche (Eintrag, k.links);
    else
    return suche (Eintrag, k.rechts);
  }
}

```

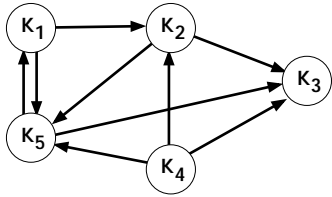
**Prog. 5–8** Hashknoten mit Methoden zum Einsortieren und zur Suche

```
void durchlaufe ()
{ System.out.println ("Hashtabelle:");
  for (int i = 0; i < n; i++)
    if (t [i] != null)
      { System.out.println (i + ":  "); durchlaufe (t [i]);
        }
}

void durchlaufe (Knoten k)
{ if (k != null)
  { durchlaufe (k.links);
    System.out.println (k.Eintrag);
    durchlaufe (k.rechts);
  }
}
```

**Prog. 5-9** Durchlauf durch eine Hashtabelle

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



Adjazenzlisten-Darstellung des Graphen:

$K_1 \rightarrow K_2, K_5$

$K_2 \rightarrow K_5, K_3$

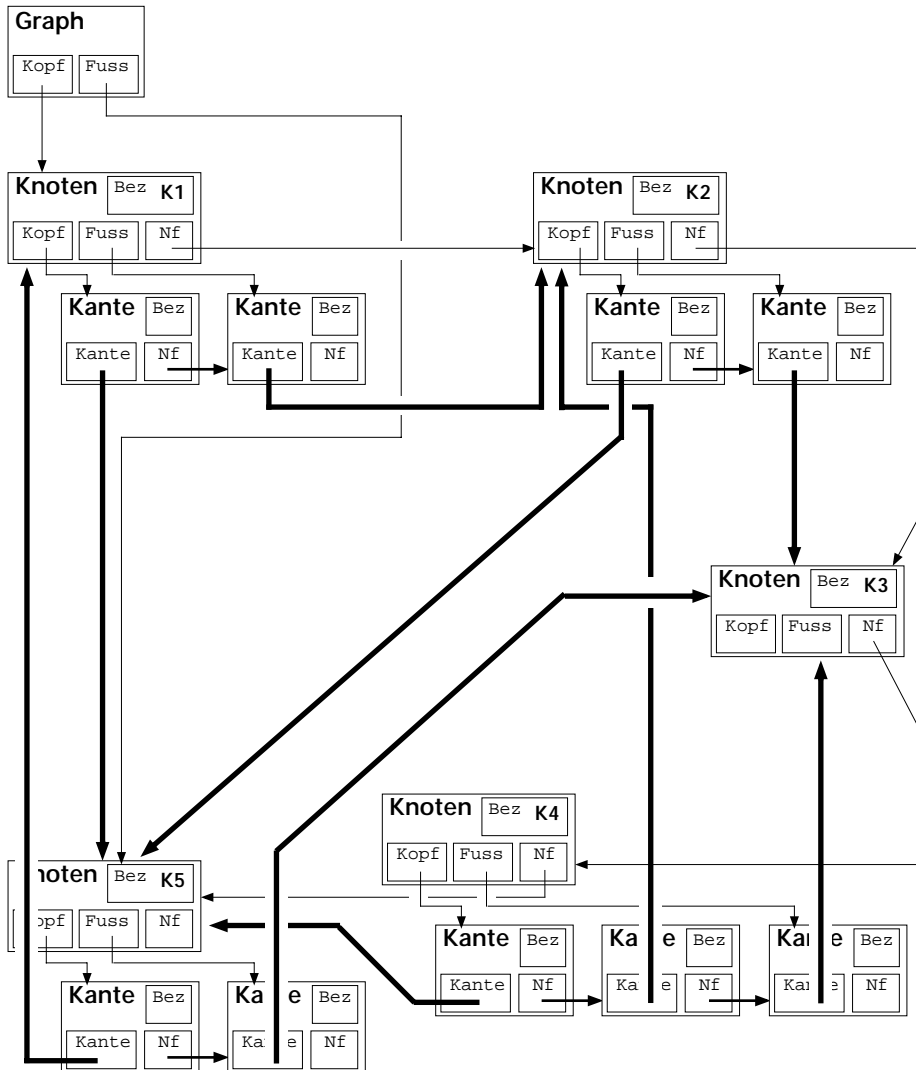
$K_3 \rightarrow$

$K_4 \rightarrow K_5, K_2, K_3$

$K_5 \rightarrow K_1, K_3$

**Abb. 5-6** Graph aus fünf Knoten und neun Kanten

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 5-7** Darstellung eines Graphen durch eine Datenstruktur

```
class Knoten
{ String Bez;    Knoten Nf;    Kante Kopf, Fuss;

  Knoten (String Bez)
  { this.Bez = Bez;    Nf = null;    Kopf = Fuss = null;
  }
}
```

```
class Kante
{ String Bez;    Kante Nf;    Knoten Kante;

  Kante (String Bez, Knoten Kante)
  { this.Bez = Bez;    Nf = null;    this.Kante = Kante;
  }
}
```

### **Prog. 5-10** *Klassen Knoten und Kante*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

class Graph
{ Knoten Kopf, Fuss;

  Graph ()
  { Kopf = Fuss = null;
  }

  Knoten suche_Knoten (String Bez)
  { Knoten k = Kopf;
    while (k != null && ! k.Bez.equals (Bez))    k = k.Nf;
    return k;
  }

  Kante suche_Kante (String von, String nach)
  { Knoten vonK = suche_Knoten (von),
    nachK = suche_Knoten (nach);
    if (vonK != null)
    { Kante ka = vonK.Kopf;
      while (ka != null && ka.Kante != nachK)    ka = ka.Nf;
      return ka;
    }
    else
      return null;
  }

  void durchlaufe ()
  { System.out.println ("Graph:");    durchlaufe (Kopf);
  }

  void durchlaufe (Knoten k)
  { if (k != null)
    { System.out.print (k.Bez + " -->");    durchlaufe (k.Kopf);
      System.out.println ();                durchlaufe (k.Nf);
    }
  }

  void durchlaufe (Kante ka)
  { if (ka != null)
    { if (ka.Bez.length () > 0)
      System.out.print (" [" + ka.Bez + "]" + ka.Kante.Bez);
      else
      System.out.print (" " + ka.Kante.Bez);
      durchlaufe (ka.Nf);
    }
  }
}

```

**Prog. 5-11** Klasse Graph, erster Teil

```

void erzeuge_Knoten (String Bez)
{ if (Kopf == null) Kopf = Fuss = new Knoten (Bez);
  else
    Fuss = Fuss.Nf = new Knoten (Bez);
}

void erzeuge_Kante (String von, String nach, String Bez)
{ Knoten vonK = suche_Knoten (von),
  nachK = suche_Knoten (nach);
  if (vonK != null && nachK != null)
    if (vonK.Kopf == null)
      vonK.Kopf = vonK.Fuss = new Kante (Bez, nachK);
    else
      vonK.Fuss = vonK.Fuss.Nf = new Kante (Bez, nachK);
}

void entferne_Knoten (String Bez)
{ Kopf = entferne_Knoten (Bez, Kopf);
}

Knoten entferne_Knoten (String Bez, Knoten k)
{ if (k != null)
  if (k.Bez.equals (Bez))
    { entferne_hinf (Kopf, k); return k.Nf;
    }
  else
    { k.Nf = entferne_Knoten (Bez, k.Nf); return k;
    }
  else return null;
}

void entferne_hinf (Knoten vonK, Knoten nachK)
{ if (vonK != null)
  { vonK.Kopf = entferne_Kante (vonK.Kopf, nachK);
    entferne_hinf (vonK.Nf, nachK);
  }
}

void entferne_Kante (String von, String nach)
{ Knoten vonK = suche_Knoten (von),
  nachK = suche_Knoten (nach);
  if (vonK != null && nachK != null)
    vonK.Kopf = entferne_Kante (vonK.Kopf, nachK);
}

Kante entferne_Kante (Kante ka, Knoten nachK)
{ if (ka != null)
  if (ka.Kante == nachK)
    return ka.Nf;
  else
    { ka.Nf = entferne_Kante (ka.Nf, nachK); return ka;
    }
  else return null;
}
}

```

**Prog. 5-12** Klasse Graph, zweiter Teil

---

# 6

```
class Bahnreise
{
    DatumsTyp Datum;
    Ort      Start, Ziel;
    ZugTyp   Zug;
}
```

```
class Flugreise
{
    DatumsTyp Datum;
    Ort      Start, Ziel;
    FlugTyp  Flug;
    FlugKlasse Klasse;
}
```

**Prog. 6-1** Klassendefinition für Objekte vom Typ *Bahnreise* und *Flugreise*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

---

```
class Reise
{
    DatumsTyp Datum;
    Ort        Start, Ziel;
}
```

**Prog. 6-2** *Definition der gemeinsamen Eigenschaften von Informationen über Reisen*

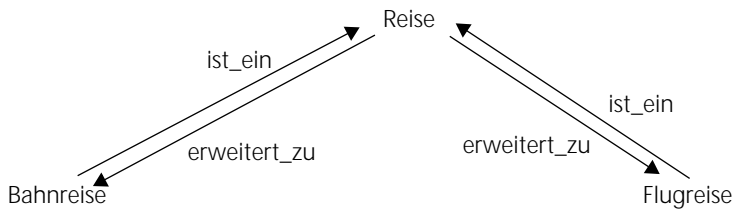
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Bahnreise extends Reise    | class Flugreise extends Reise
{ ZugTyp Zug;                   | { FlugTyp Flug;
}                                 |   FlugKlasse Klasse;
                                | }

```

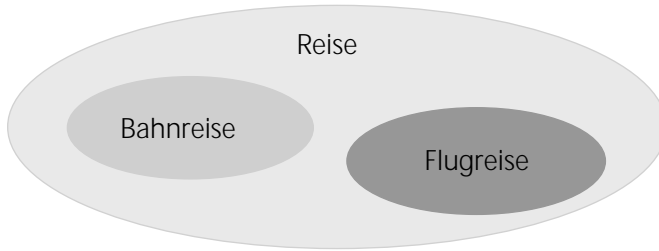
**Prog. 6–3** Die Klassendefinitionen für Flug- und Bahnreisen als Erweiterung der Klasse *Reise*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-1** Grafische Darstellung der Beziehungen zwischen der Oberklasse `Reise` und den Unterklassen `Bahnreise` sowie `Flugreise`

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6–2** Venn-Diagramm der Extensionen der Klassendefinitionen des Reisebeispiels

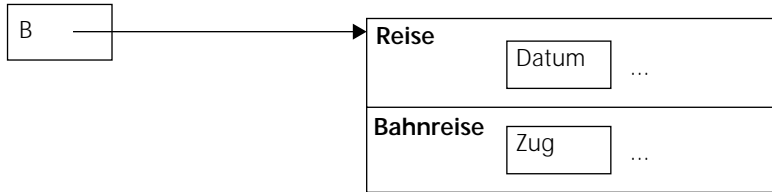
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
Bahnreise B;   B = new Bahnreise ();
Flugreise F;   F = new Flugreise ();
Reise         R;
F.Flug = new FlugTyp();
R = B; // ist erlaubt.
```

**Prog. 6-4** Beispiel für die Verwendung von Variablen der Oberklasse für den Verweis auf Objekte der Unterklasse

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
Bahnreise B;  
B = new Bahnreise();  
B.Zug = ... // erlaubter Zugriff  
B.Datum = ... // erlaubter Zugriff
```



**Abb. 6-3** Konzeptionelles Modell der Anordnung der Attribute in einem Objekt der Klasse Bahnreise

---

```
Bahnreise B;   B = new Bahnreise();
Flugreise F;   F = new Flugreise();
Reise       R;

int i = Eingabe(); // aus Datei, aus Textfeld oder von der Tastatur
if (i < 0) R = F;
else      R = B;
```

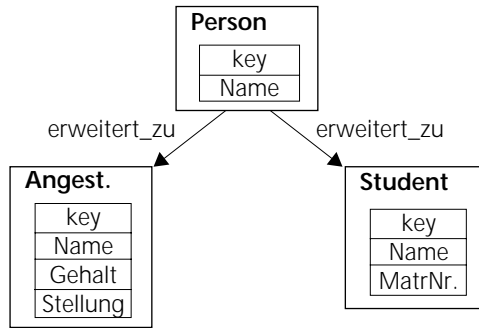
**Prog. 6-5** *Abhängigkeit der Programmausführung von der Eingabe*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
if (R instanceof Flugreise)
{ F = (Flugreise) R;    // ... und weitere Anweisungen.
}
else if (R instanceof Bahnreise)
{ B = (Bahnreise) R;    // ... und weitere Anweisungen.
}
else
{ System.out.println("R verweist auf ein Objekt unbekanntes Typs");
}
```

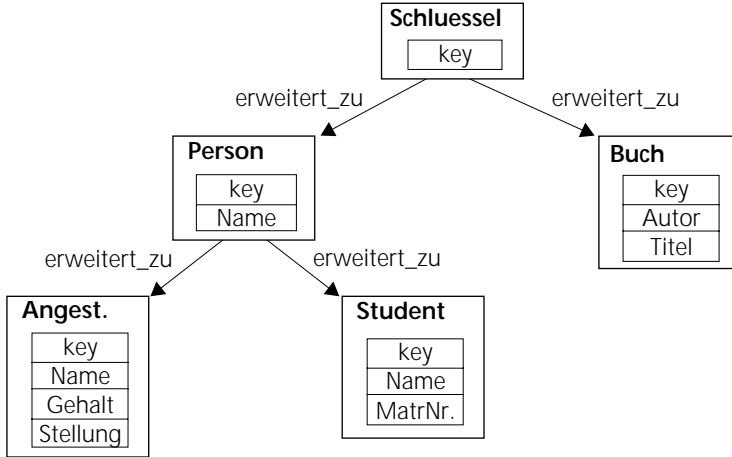
**Prog. 6-6** Typsichere Programmierung unter Verwendung von instanceof

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-4** Klassenhierarchie der Personen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-5** Klassenhierarchie des Bibliotheksbeispiels

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
class Schluessel
{ int key;
}

class Person extends Schluessel
{ String Name;
}

class Angestellter extends Person
{ int Gehalt;
  PositionsType Stellung;
}

class Student extends Person
{ Matrikelnummer MatNr;
}

class Buch extends Schluessel
{ Person Autor;
  TitelTyp Titel;
}
```

**Prog. 6-7** Fragment einer Java-Implementierung des Bibliotheksbeispiels aus Abb. 6-5. Die in den Definitionen benutzten, aber nicht definierten Namen seien an anderer Stelle entsprechend vereinbart.

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Schluessel
{ int key;

  // Konstruktor der Oberklasse:
  Schluessel (int k)
  { key = k;
  }
}
```

```
class Person extends Schluessel
{ String Name;

  // einfacher Konstruktor:
  Person(String DerName)
  { Name = DerName;
  }

  // Konstruktor bezieht Ober-
  // klasse mit ein:
  Person (String DerName, int k)
  { super(k); // muss als erstes
              // erfolgen.
    Name = DerName;
  }
}
```

**Prog. 6–8** Bibliotheksbeispiel (aus Prog. 6–7), angereichert um spezielle Konstruktoren

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class Punkt2D
{ int x,y;

  Punkt2D ()
  { this (0, 0);
  }

  Punkt2D (int x, int y)
  { this.x = x;  this.y=y;
  }
}

class Punkt3D extends Punkt2D
{ int z;

  Punkt3D ()
  { this (0, 0, 0);
  }

  Punkt3D (int x, int y)
  { this (x, y, 0);
  }

  Punkt3D (int x, int y, int z)
  { super (x, y);
    this.z = z;
  }
}
```

**Prog. 6-9** Beispiel für explizite Konstruktoraufrufe

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class X
{ int xWert = 255;
  int Wert;

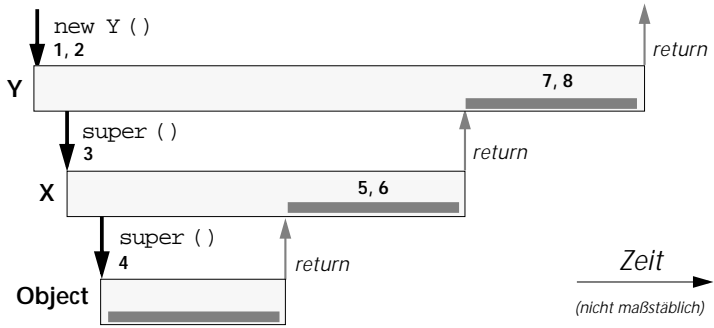
  X ()
  { Wert = xWert;
  }
}

class Y extends X
{ int yWert = 128;

  Y ()
  { Wert = Wert + yWert;
  }
}
```

**Prog. 6–10** Zur Erklärung der Initialisierungsaktionen im Falle von Unterklassen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-6** Darstellung der Ausführungsreihenfolge beim Aufruf des Konstruktors `new Y()` aus Prog. 6-10. Die Zahlen beziehen sich auf die Schritte in der Tabelle.

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class Object
```

<code>public boolean equal (Object obj)</code>	dient zum Vergleich von Objekten
<code>protected Object clone ()</code>	dient zum Kopieren von Objekten
<code>public final Class getClass ()</code>	liefert eine Darstellung der Klasse des Objekts
<code>protected void finalize ()</code>	erledigt bestimmte Aufgaben bei der Beseitigung nicht mehr zugreifbarer Objekte
...	

```
}
```

**Abb. 6-7** Wichtige Methoden der Klasse Object

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
public class EinfacheErweiterung
{
    public static void main
        (String unbenutzt [])
    {
        A a = new A (26);
        B b = new B ();
        a.DruckeA();  b.DruckeB();
    }
}

class A
{ int x = 25;

  A () { }

  A (int y)
  { x = y;
  }

  void DruckeA ()
  { System.out.println
    ("Die Variable x hat den Wert "
    + x + " in einem Objekt der "
    + "Klasse A");
  }
}

class B extends A
{ String z = "String-Variable in "
  + "einem Objekt der "
  + "Klasse B";

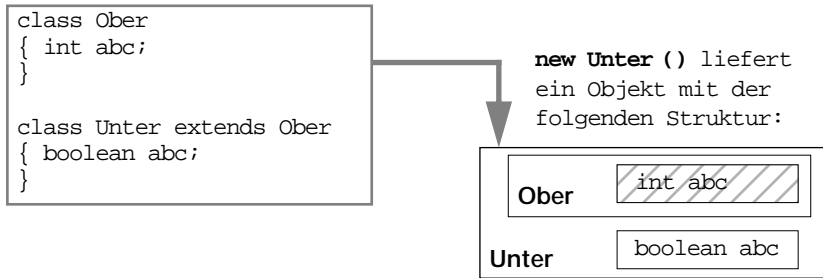
  void DruckeB ()
  { x = 2*x;
    System.out.println
      ("z enthaelt >" + z + "<");
    DruckeA ();
  }
}
```

**Ausgabe:**

Die Variable x hat den Wert 26 in einem Objekt der Klasse A  
z enthaelt >String-Variable in einem Objekt der Klasse B<  
Die Variable x hat den Wert 50 in einem Objekt der Klasse A

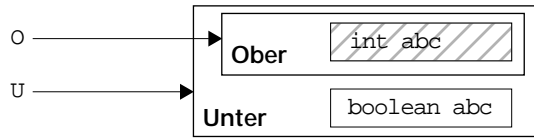
**Prog. 6–11** Ein einfaches Beispiel für die Erweiterung einer Klasse um zusätzliche Methoden

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 6-8** Gleiche Namen für Variablen in Unter- und Oberklasse

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-9** Zugriff auf die Variablen-Namensräume eines Objektes vom Typ `Unter`

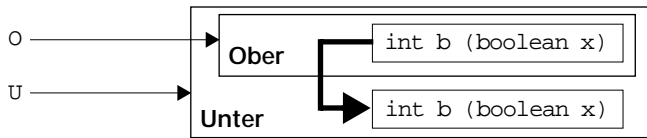
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Ober
{ ...
  int b (boolean x)
  { if (x) return 5;
    else return 6;
  }
}
```

```
class Unter extends Ober
{ ...
  int b (boolean y)
  { if (y) return -27;
    else return -900;
  }
}
```

**Prog. 6-12** Überschreiben einer Methodendeklaration in einer Klassenerweiterung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-10** Zugriff auf die Methoden-Namensräume eines Objekts vom Typ `Unter`

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
Unter Un = new Unter();
Ober Ob = Un;

System.out.println (Ob.b (true));
System.out.println (Un.b (true));
System.out.println ("-----");
Ob = new Ober ();
System.out.println (Ob.b (true));
System.out.println (Un.b (true));
```

**Erzeugte Ausgabe:**

```
-27
-27
-----
5
-27
```

**Prog. 6–13** *Erweitertes Beispiel zum Überschreiben von Methoden*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Ober
{
    int b (boolean x)
    { if (x) return 5;
      else return 6;
    }
}

class Unter extends Ober
{
    int b (String s)
    { return s.length ();
    }

    int b (int g)
    { return 2*g;
    }

    int b (boolean y)
    { if (y) return -27;
      else return -900;
    }
}
```

**Prog. 6-14** Überschreiben und Überladen von Methodennamen

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
Unter Un = new Unter();
Ober Ob = Un;

System.out.println (Ob.b (true));
System.out.println (Un.b (true));
System.out.println (Un.b ("abc"));
System.out.println (Un.b (5));
System.out.println ("-----");

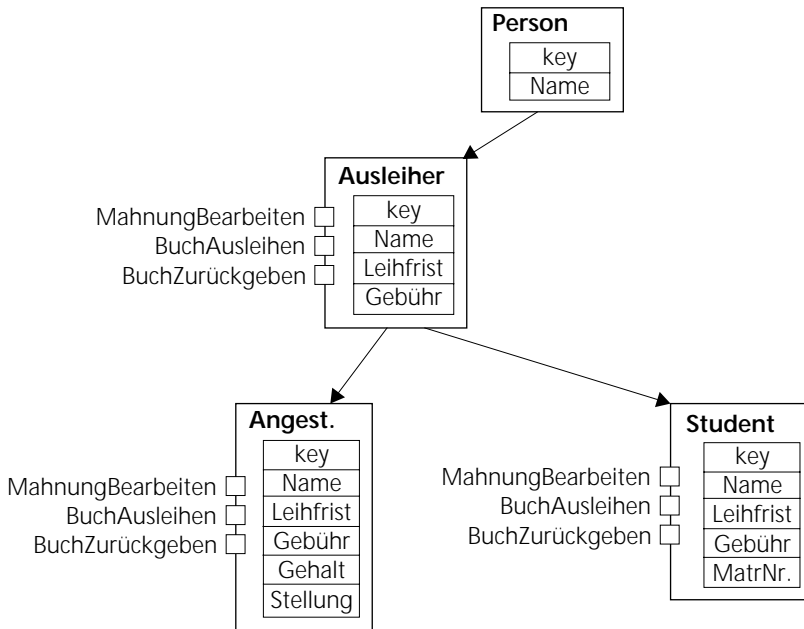
Ob = new Ober();
System.out.println (Ob.b (true));
System.out.println (Un.b (true));
```

Erzeugte Ausgabe:

```
-27
-27
3
10
-----
5
-27
```

**Prog. 6-15** Benutzung der Klassen aus Prog. 6-14

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



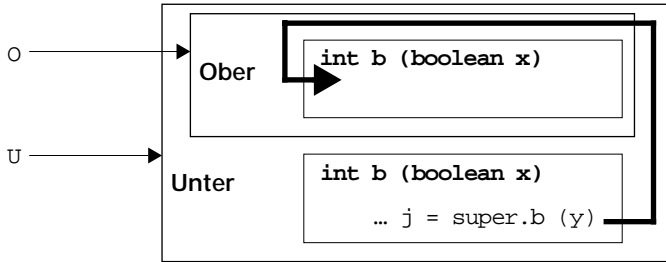
**Abb. 6–11** Erweiterung eines Ausschnitts des Bibliotheksmodells Abb. 6–4 (Seite 195) um Methoden und die Klasse **Ausleiher**

```
class Ausleiher extends Person
{ ...
  void MahnungBearbeiten()
  { if (LeihfristÜberschritten ())
    { x = BerechneGebühr ();    Gebühr = Gebühr + x;
      DruckeMahnung (Gebühr);
    }
  }
  ...
}

class Student extends Ausleiher
{ ...
  void MahnungBearbeiten()
  { if (LeihfristÜberschritten ())
    { x = BerechneGebühr ();    Gebühr = Gebühr + x;
      DruckeMahnung (Gebühr);  SperreExmatrikulation ();
    }
  }
  ...
}
```

**Prog. 6-16** Methode `MahnungBearbeiten` in den Klassen `Ausleiher` und `Student`

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-12** Zugriff von der Unterklasse auf eine überschriebene Methode der Oberklasse

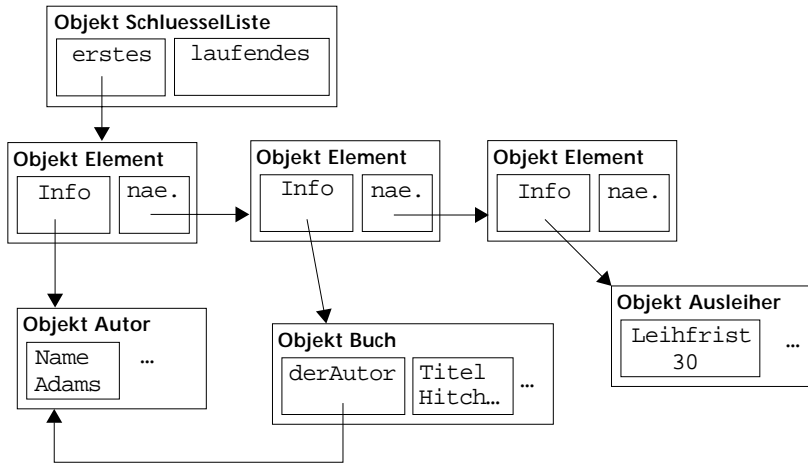
Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class Ober
{
    int b (boolean x)
    { if (x) return 5;
      else return 6;
    }
}
```

```
class Unter extends Ober
{
    int b (boolean y)
    { int j;
      j = super.b (y);
      if (y) return -27;
      else return -j;
    }
}
```

**Prog. 6-17** Die Verwendung von `super` als Oberklassenverweis

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6–13** Beispiel für den Aufbau der Liste, welche die Bibliotheksdatenbank darstellt

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class SchluesselListe
{
  class Element
  { Schluessel Info;
    Element naechstes;

    Element (Schluessel k)
    { Info = k;
    }
  } // end class Element

  Element erstes;
  Element laufendes;

  SchluesselListe (...)
  { ... }

  Schluessel finde (int zufinden)
  { ... }

  Schluessel dasErsteElement ()
  { ... }

  Schluessel dasNaechsteElement ()
  { ... }

  void einfuegen (Schluessel k)
  { ... }

  void entferne (Schluessel k)
  { ... }

  void DruckeAlle ()
  { ... }

  void DruckeAlleInfos ()
  { ... }
} // end class SchluesselListe
```

### **Prog. 6-18** Klasse SchluesselListe

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```

import java.util.Calendar;

class Ausleiher extends Person
{
    String Id ()
    { ...
    }

    int Leihfrist; // Ausleihfrist in Tagen
    int Gebuehr; // Aufgelaufene Gebuehr in Euro
    BuecherListe dieAusgeliehenenBuecher;

    Ausleiher ( ... )
    { super ( ... );
      Leihfrist = 30; dieAusgeliehenenBuecher = new BuecherListe (einaus);
    }

    void DruckeMahnung ()
    { ...zeigeAus ("Mahnung: " + Gebuehr + " Euro an Gebuehren" + " aufgelaufen");
    }

    void BerechneGebuehr (Calendar heute)
    { // pro ueberzogenem Tag einen Euro (Ein Jahr wird zu 360 Tagen gerechnet),
      // zusätzlich Porto. Berechne die Anzahl der Tage, die ueberzogen sind.
      int Porto = 1; // hypothetische Standardbriefgebuehr in Euro.
      Buch laufendesb;
      laufendesb = dieAusgeliehenenBuecher.dasErsteElement();

      while (laufendesb != null)
      { if (heute.after (laufendesb.RueckgabeDatum))
        { int jahr1 = heute.get (Calendar.YEAR);
          int jahr0 = laufendesb.RueckgabeDatum.get (Calendar.YEAR);
          int tag1 = heute.get (Calendar.DAY_OF_YEAR);
          int tag0 = laufendesb.RueckgabeDatum.get (Calendar.DAY_OF_YEAR);
          int diff = (jahr1-jahr0)*360 + (tag1-tag0);
          ...zeigeAus ( diff + " Tage ueberzogen fuer Ausleiher "
            + Id () + " und Buch " + laufendesb.Id () );
          Gebuehr = Gebuehr + diff; // sehr restriktiv und teuer!
        };
        laufendesb = dieAusgeliehenenBuecher.dasNaechsteElement ();
      }
      // Falls eine Gebuehr festgestellt wurde, noch das Porto addieren:
      if (Gebuehr > 0) Gebuehr = Gebuehr + Porto;
    }

    void MahnungBearbeiten (Calendar heute)
    { BerechneGebuehr (heute);
      if (Gebuehr > 0)
      { ...zeigeAus ("\nAn " + Name + " , " + Vorname + " Ihre Nummer: " + key);
        DruckeMahnung ();
      }
    }
}

```

**Prog. 6-19** Ausschnitt aus den Details der Klasse Ausleiher

Lehrbuch der Programmierung mit Java, Echtke Goedicke, Heidelberg, © dpunkt 2000

```

class Angestellter extends Ausleiher
{
    String Id ()
    {
        ...
    }

    int Gehalt;

    Angestellter ( ... )
    {
        ... ; Leihfrist = 120;
    }

    void MahnungBearbeiten (Calendar heute)
    {
        super.MahnungBearbeiten (heute);
        if (Gebuehr >0) BucheGebuehrab ();
    }

    void BucheGebuehrab ()
    {
        ...zeigeAus ("Gebuehr in Hoehe von " + Gebuehr + " Euro wird " "abgebucht\n");
        Gehalt = Gehalt - Gebuehr; Gebuehr = 0;
    }

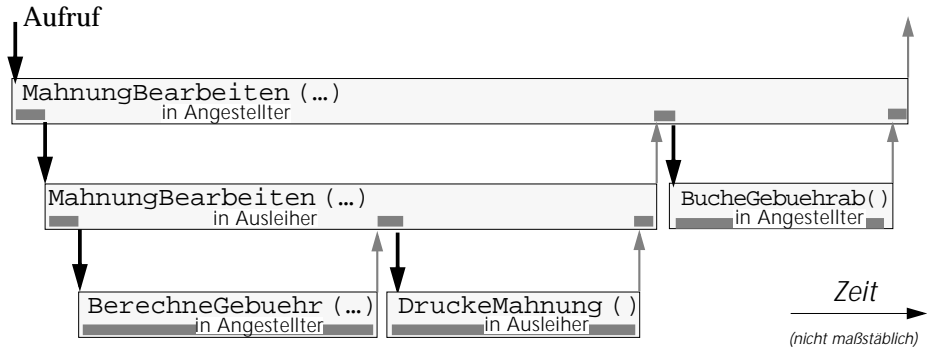
    void BerechneGebuehr(Calendar heute)
    {
        // pro ueberzogenem Tag einen Euro! Ein Jahr hat 360 Tage ...
        // aber kein Porto!
        // Berechne die Anzahl der Tage, die ueberzogen sind.
        Buch laufendesb;
        laufendesb = dieAusgeliehenenBuecher.dasErsteElement ();

        while (laufendesb != null)
        {
            if (heute.after (laufendesb.RueckgabeDatum))
            {
                int jahr1 = heute.get (Calendar.YEAR);
                int jahr0 = laufendesb.RueckgabeDatum.get (Calendar.YEAR);
                int tag1 = heute.get (Calendar.DAY_OF_YEAR);
                int tag0 = laufendesb.RueckgabeDatum.get (Calendar.DAY_OF_YEAR);
                int diff = (jahr1 - jahr0)*360 + (tag1 - tag0);
                ...zeigeAus ( diff + " Tage ueberzogen fuer Ausleiher "
                    + Id() + " und Buch " + laufendesb.Id () );
                Gebuehr = Gebuehr + diff; // sehr restriktiv und teuer!
            }
            laufendesb = dieAusgeliehenenBuecher.dasNaechsteElement ();
        }
    }
}

```

**Prog. 6-20** Ausschnitt aus den Details der Klasse Angestellter

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000



**Abb. 6-14** Abfolge der Methodenaufrufe bei der Abarbeitung der Methode `MahnungBearbeiten` in einem Objekt der Klasse `Angestellter`

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

class Student extends Ausleiher
{
    String Id ()
    { ...
    }

    int MatNr;

    Student ( ... )
    { ... Leihfrist = 60;
    }

    boolean KannExmatrikuliertWerden = true;

    void SperreExmatrikulation ()
    { KannExmatrikuliertWerden = false;
      ...zeigeAus ("Bitte erst die Gebuehren vor der Exmatrikulation begleichen!");
    }

    void MahnungBearbeiten (Calendar heute)
    { super.MahnungBearbeiten (heute);
      if (Gebuehr >0)
      { SperreExmatrikulation ();
        ...zeigeAus ( "Bitte die " + "aufgelaufene " + "Gebuehr von "
                      + Gebuehr + " einzahlen!");
      }
    }

    void BerechneGebuehr (Calendar heute)
    { // pro ueberzogenem Tag einen Euro!ein Jahr hat 360 Tage ...
      // aber kein Porto!
      // Berechne die Anzahl der Tage, die ueberzogen sind
      Buch laufendesb;
      laufendesb = dieAusgeliehenenBuecher.dasErsteElement();

      while (laufendesb != null)
      { if (heute.after (laufendesb.RueckgabeDatum))
        { int jahr1 = heute.get (Calendar.YEAR);
          int jahr0 = laufendesb.RueckgabeDatum.get (Calendar.YEAR);
          int tag1 = heute.get (Calendar.DAY_OF_YEAR);
          int tag0 = laufendesb.RueckgabeDatum.get (Calendar.DAY_OF_YEAR);
          int diff = (jahr1 - jahr0)*360 + (tag1 - tag0);
          ...zeigeAus ( diff + " Tage ueberzogen fuer Ausleiher "
                        + Id () + " und Buch " + laufendesb.Id () );
          Gebuehr = Gebuehr + diff; // sehr restriktiv und teuer!
        }
        laufendesb = dieAusgeliehenenBuecher.dasNaechsteElement ();
      }
    }
}

```

### **Prog. 6-21** Ausschnitt aus den Details der Klasse Student

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class Ober
{ String EinText;
  void TuWas()
  { }
}
```

...

```
Ober o = new Ober () { void TuWas ()
                       { System.out.println (EinText);
                       }
};
```

Überschreiben der Methode TuWas



### **Prog. 6-22** Anonyme Klassenerweiterung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class command
{ String commandstring;
  command (String u)
  { commandstring = u;
  }
}

void go ()
{ ...
}

void go (String dieEingabe)
{ ...
}
```

← Kommando-  
name

← Konstruktor

← Kommando-  
methode(n)

The diagram shows the Java code for the Command class and two methods. Annotations with arrows point to specific parts of the code: 'Kommando-name' points to the class name 'command', 'Konstruktor' points to the constructor 'command (String u)', and 'Kommando-methode(n)' points to both 'void go ()' and 'void go (String dieEingabe)'.

**Prog. 6-23** Klassendefinition der Klasse `command`

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
...
new command ("neuerAngestellter")
{
  void go (String e)
  {
    if (WortAnz (e, 1) < 3)
    {
      zeigeAus ( "Falsche Zahl von Parametern fuer Kommando: "
        + commandstring +
        + "\nbenoetigt: neuerAngestellter Vorname Nachname");
    }
    else
    {
      dieBibliothek.neuerAngestellter (Wort (e, 1, 2), Wort (e, 1, 3));
    }
  }
}

```

Prüfung auf Mindestzahl von Parametern

Anzeigen einer hilfreichen Fehlermeldung, falls Parameter fehlen

Extraktion des 2. und 3. Wortes aus der Kommandozeile

Aufruf der Methode zum Einfügen der Informationen über einen neuen Angestellten

**Prog. 6-24** Eine spezielle Version von `command` für das Einfügen eines neuen Angestellten

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

```
command [] thecommands =
{
    new command ("Ende")
    { void go (String e) { Ende();}
    },

    new command ("DruckeDB")
    { void go (String e) {dieBibliothek.DruckeAlle ();}
    },

    new command ("NeuerAusleiher")
    { void go (String e)
      { if (WortAnz (e, 1) < 3)
          zeigeAus ( "Falsche Zahl von Parametern fuer Kommando "
                    + commandstring + "\nbenoetigt: "
                    + "NeuerAusleiher Nachname Vorname");
        else
          dieBibliothek.neuerAusleiher (Wort (e, 1, 2), Wort (e, 1, 3));
      }
    },

    new command ("trageNeuesBuchEin")
    { void go (String e)
      { if (WortAnz (e, 1) < 4)
          zeigeAus ( "Falsche Zahl von Parametern fuer Kommando:"
                    + commandstring + "\nbenoetigt: "
                    + "trageNeuesBuchEin Titel Vorname Nachname");
        else
          dieBibliothek.trageNeuesBuchEin
            (Wort (e, 1, 2), Wort (e, 1, 3), Wort (e, 1, 4));
      }
    },

    ""
};
```

### **Prog. 6-25** Objekte für die Speicherung von Kommandos

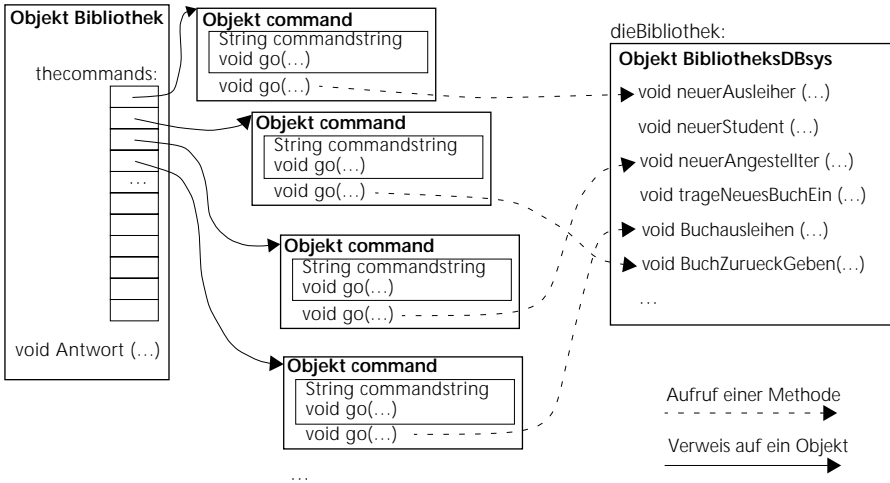
Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

```
public void Antwort (String Eingabe)
{ int i; boolean fertig;
  if (SatzAnz (Eingabe) >= 1)
  { if (WortAnz (Eingabe, 1) >= 1)
    { i = 0; fertig = false;
      while ( ! fertig)
      { if (i >= thecommands.length)
        fertig = true;
        elseif (thecommands[i].commandstring.compareTo(Wort (Eingabe, 1, 1)) == 0)
          thecommands [i].go (Eingabe); fertig = true;
        else
          i++;
        }
        if (i >= thecommands.length)
          zeigeAus ("Kein Kommando: " + Wort (Eingabe, 1, 1));
        }
        else {zeigeAus("Nicht genug Eingabezeichen?" + Eingabe);}
      }
    }
  else {zeigeAus("Leere Eingabe?" + Eingabe);}
}

public void Hilfe()
{ zeigeAus(" Dies ist das Bibliotheksprogramm ... die Kommandos lauten:");
  for (int i = 0; i < thecommands.length; i++)
    zeigeAus (thecommands [i].commandstring);
}
```

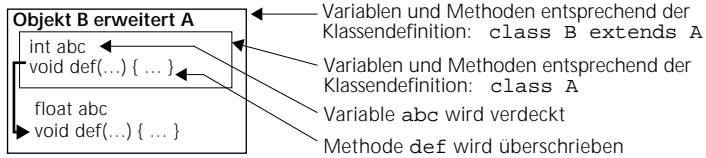
**Prog. 6-26 Methoden Antwort und Hilfe zur Bearbeitung einer Kommandozeile**

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

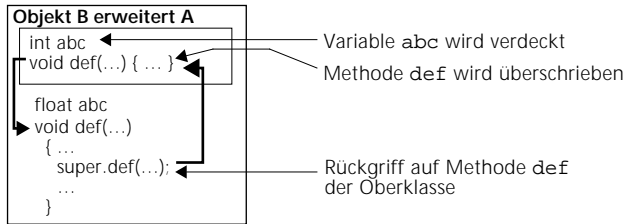


**Abb. 6-15** Verweisstruktur und Methodenaufrufe des Bibliothekssystems

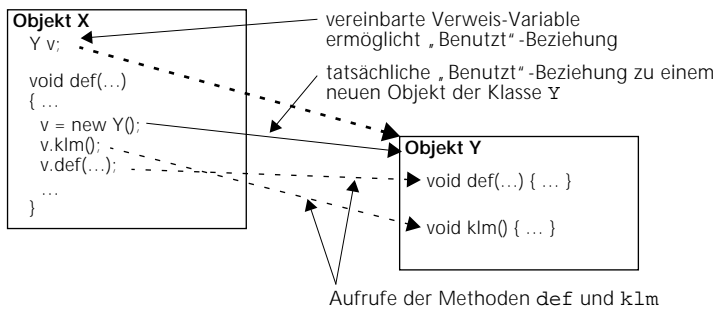
### Statische Erweiterung (benannt oder anonym)



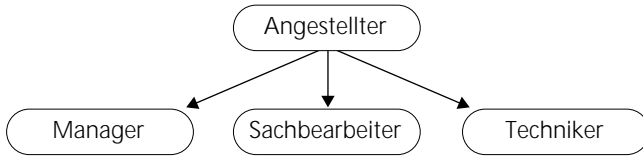
### Dynamische Erweiterung



### Objekt vom Typ X „benutzt“ ein Objekt vom Typ Y

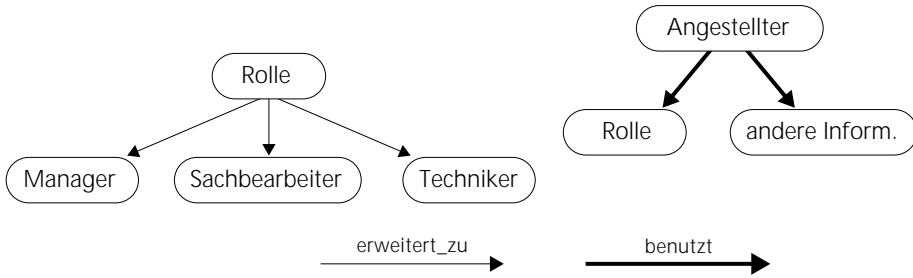


**Abb. 6-16** Die bisher vorgestellten Formen von Beziehungen zwischen Klassen und deren Objekten



**Abb. 6-17** Begriffliche Hierarchie der Mitarbeiter in einer Firma

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 6-18** Ein Beispiel für Erweiterung durch Delegation

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```

class BibliotheksDBsys
{
    SchluesselListe dieBibliotheksDB;
    // die Liste aller Buecher, Ausleiher und Autoren
    ...
    BibliotheksDBsys (...)
    { ... dieBibliotheksDB = new SchluesselListe (einaus); }

    void neuerAusleiher (String N, String V)
    { // keine Plausibilitaetsueberpruefung ...
      Ausleiher derAusleiher = new Ausleiher (... , N, V);
      dieBibliotheksDB.einfuegen (derAusleiher);
    }

    void neuerAngestellter (String N, String V)
    { // keine Plausibilitaetsueberpruefung ...
      Angestellter derAusleiher = new Angestellter (... , N, V);
      dieBibliotheksDB.einfuegen (derAusleiher);
    }

    void neuerStudent (String N, String V)
    { // keine Plausibilitaetsueberpruefung ...
      Student derAusleiher = new Student (... , N, V);
      dieBibliotheksDB.einfuegen (derAusleiher);
    }

    void DruckeAlle ()
    { dieBibliotheksDB.DruckeAlle ()
    }

    void DruckeAlleInfos()
    { dieBibliotheksDB.DruckeAlleInfos ();
    }

    void DruckeAlleMahnungen ()
    { Schluessel dasElement = dieBibliotheksDB.dasErsteElement ();
      while (dasElement != null)
      { if (dasElement instanceof Ausleiher)
        { ((Ausleiher)dasElement).MahnungBearbeiten(heute);
          dasElement = dieBibliotheksDB.dasNaechsteElement();
        }
      }
    }

    ... dieBibliotheksDB = ... ; // SchluesselListe-Objekt wird zugewiesen.
    ...
}

```

### **Prog. 6-27** Fragmente der Klasse BibliotheksDBsys

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```

void trageNeuesBuchEin (String Titel, String Vorname_, String Nachname_)
{ // suche Autor .... falls nicht vorhanden direkt einfuegen ...
  Schluessel dasElement = dieBibliotheksDB.dasErsteElement ();
  boolean fertig = false;
  Autor derAutor = null;
  Buch dasBuch;

```

```

while ( ! fertig)                                     Suche nach dem Autor. Falls nicht vorhanden, füge ein neues Autor-Objekt ein.
{ if (dasElement == null)
  { fertig = true; // neuen Autor einfuegen.
    derAutor = new Autor (EA, Nachname_,Vorname_);
    dieBibliotheksDB.einfuegen (derAutor);
  }
  else
  { if (dasElement instanceof Autor)
    { if ( (((Autor)dasElement).Vorname.compareTo (Vorname_)) == 0 &&
          (((Autor)dasElement).Name.compareTo (Nachname_)) == 0)
      { derAutor = (Autor) dasElement; fertig = true;
      }
    else
      dasElement = dieBibliotheksDB.dasNaechsteElement ();
    }
  else
    { dasElement = dieBibliotheksDB.dasNaechsteElement ();
    }
  }
}

```

```

// Nun ist ein Autor gefunden und das Buch kann eingetragen werden
// (hier keine Ueberpruefung, ob das Buch schon in der Liste ist, da mehrere
// Exemplare des Buchs existieren können und getrennt zu verwalten sind).

```

```

dasBuch = new Buch (EA, Titel, derAutor);           Füge das neue Buch ein.
dieBibliotheksDB.einfuegen (dasBuch);

```

```

derAutor.neuesBuch (dasBuch);                       Trage das neue Buch in die Bücherliste des Autors ein.

```

```

}

```

### Prog. 6-28 Methode trageNeuesBuchEin der Klasse BibliotheksDBsys

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

```
class IntList
{ class ListenElement
  { int derWert;
    ListenElement weiter;
  }

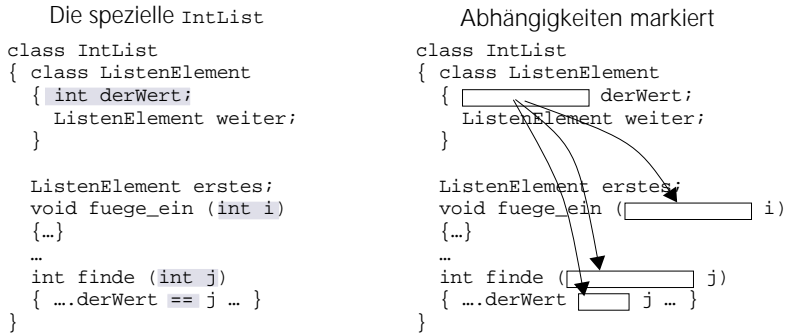
  ListenElement erstes;
  void fuege_ein (int i)
  {...}
  ...
  int finde (int j)
  { ...derWert == j ... }
}
```

```
class StrList
{ class ListenElement
  { String derWert;
    ListenElement weiter;
  }

  ListenElement erstes;
  void fuege_ein (String s)
  {...}
  ...
  String finde (String s)
  { ...derWert.compareTo (s)... }
}
```

**Prog. 7-1 Ähnliche Listenstrukturen**

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 7-1** Links: IntList im Original, rechts: Markierung der Abhängigkeiten zwischen dem allgemeinen Teil und dem speziellen Typ der Listen-Objekte

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class ErweiterungsListe
{
    class ListenElement // Lokale Klasse, für konkrete Verwendung zu erweitern.
    {
        ListenElement naechstes;
    }

    ListenElement erstes, laufendes;

    ErweiterungsListe () // Konstruktor.
    {
        erstes = null; laufendes = erstes;
    }

    void fuege_ein (ListenElement k) // Füge am Listenanfang ein.
    {
        k.naechstes = erstes; erstes = k;
    }

    boolean gleich (ListenElement x, ListenElement y) // Prüfe zwei Elemente
    {
        ... // auf Gleichheit.
    }

    ListenElement finde (ListenElement zufinden) // Finde ein bestimmtes
    {
        laufendes = erstes; // Element.
        while (laufendes != null)
            if (gleich (laufendes, zufinden)) return laufendes; // gefunden.
            else laufendes = laufendes.naechstes;
        return null; // nichts gefunden.
    }

    void entferne (ListenElement k)
    {
        if (erstes != null && erstes == k)
        {
            erstes = erstes.naechstes; k.naechstes = null;
        }
        else if (k != null)
        {
            laufendes = erstes;
            while (laufendes != null)
                if (laufendes.naechstes == k)
                {
                    laufendes.naechstes = k.naechstes; k.naechstes = null; return;
                }
                else
                    laufendes = laufendes.naechstes;
        }
    }
}
}
```

### **Prog. 7-2** Die Klasse ErweiterungsListe

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
IntlList li = new IntlList();
IntlList.IntListElement i = li.new IntListElement();
i.Info = 99;
li.fuege_ein(i);
System.out.println (( (IntlList.IntListElement)li.dasErsteElement ()).Info);
```

**Prog. 7-3** Beispiel für die Verwendung der Klasse IntlList auf der Basis der Klasse ErweiterungsListe aus Prog. 7-2

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

class ObjectListe
{
    class ListenElement // Lokale Klasse, nicht zu erweitern.
    {
        Object Info;
        ListenElement naechstes;

        ListenElement (Object Info, ListenElement naechstes) // Konstruktor für
        {
            this.Info = Info;    this.naechstes = naechstes;    // ListenElement.
        }
    }

    ListenElement erstes, laufendes;

    ObjectListe () // Konstruktor für ObjectListe.
    {
        erstes = null;    laufendes = erstes;
    }

    void fuege_ein (Object k) // Füge am Listenanfang ein.
    {
        erstes = new ListenElement (k, erstes);
    }

    boolean gleich (Object x, Object y) // Prüfe zwei Objekte
    {
        ... // auf Gleichheit.
    }

    Object finde (Object zufinden) // Finde ein bestimmtes Element.
    {
        laufendes = erstes;
        while (laufendes != null)
            if (gleich (laufendes.Info, zufinden))
                return laufendes.Info; // gefunden.
            else
                laufendes = laufendes.naechstes;
        return null; // nichts gefunden.
    }

    void entferne (Object k)
    {
        if (erstes != null)
        {
            if (erstes.Info == k)
                erstes = erstes.naechstes;
            else
            {
                laufendes = erstes;
                while (laufendes.naechstes != null)
                    if (laufendes.naechstes.Info == k)
                    {
                        laufendes.naechstes = laufendes.naechstes.naechstes;    return;
                    }
                else
                    laufendes = laufendes.naechstes;
            }
        }
    }
}

```

#### **Prog. 7-4** Die Klasse ObjectListe

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
class IntegerList
{ ObjectListe dieListe;

  IntegerList ()
  { dieListe = new ObjectListe ();
  }

  void fuege_ein (Integer k)
  { dieListe.fuege_ein (k);
  }

  void entferne (Integer k)
  { dieListe.entferne (k);
  }

  Integer finde (Integer k)
  { return (Integer) dieListe.finde (k);
  }
}
```

**Prog. 7-5** Beispiel für die Verwendung der Klasse `ObjectListe` aus Prog. 7-4

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
abstract class Element
{
    abstract boolean gleich (Element zuvergleichen);

    static int id = 0;
    int Id;

    Element ()
    { Id = id++;
    }
}
```

**Prog. 7-6 Die abstrakte Klasse Element**

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

class AllgemeineListe
{
    class ListenElement // Lokale Klasse, nicht zu erweitern.
    {
        Element Info; // Der Typ von Info ist die abstrakte Klasse Element.
        ListenElement naechstes;

        ListenElement (Element Info, ListenElement naechstes) // Konstruktor für
        {
            this.Info = Info; this.naechstes = naechstes; // ListenElement.
        }
    }

    ListenElement erstes, laufendes;

    AllgemeineListe () // Konstruktor für AllgemeineListe.
    {
        erstes = null; laufendes = erstes;
    }

    void fuege_ein (Element k) // Füge am Listenanfang ein.
    {
        erstes = new ListenElement (k, erstes);
    }

    Element finde (Element zufinden) // Finde ein bestimmtes Element.
    {
        laufendes = erstes;
        while (laufendes != null)
            if (laufendes.Info.gleich (zufinden))
                return laufendes.Info; // gefunden.
            else
                laufendes = laufendes.naechstes;
        return null; // nichts gefunden.
    }

    void entferne (Element k)
    {
        if (erstes != null)
        {
            if (erstes.Info.gleich (k))
                erstes = erstes.naechstes;
            else
            {
                laufendes = erstes;
                while (laufendes.naechstes != null)
                {
                    if (laufendes.naechstes.Info.gleich (k))
                    {
                        laufendes.naechstes = laufendes.naechstes.naechstes; return;
                    }
                    else
                        laufendes = laufendes.naechstes;
                }
            }
        }
    }

    Element erstesElement ()
    {
        laufendes = erstes; return laufendes;
    }

    Element naechstesElement ()
    {
        if (laufendes != null) laufendes = laufendes.naechstes;
        return laufendes;
    }
}

```

Die abstrakte Methode gleich aus der abstrakten Klasse Element wird hier benutzt.

### Prog. 7-7 Die Klasse AllgemeineListe

```
class Bruch extends Element
{
    Bruch ()
    { this (0, 1);
    }

    Bruch (int Z, int N)
    { Zaehler = Z; Nenner = N;
      if (Nenner == 0)
          System.out.println ("Es wurde ein unzulässiger Bruch erzeugt.");
    }

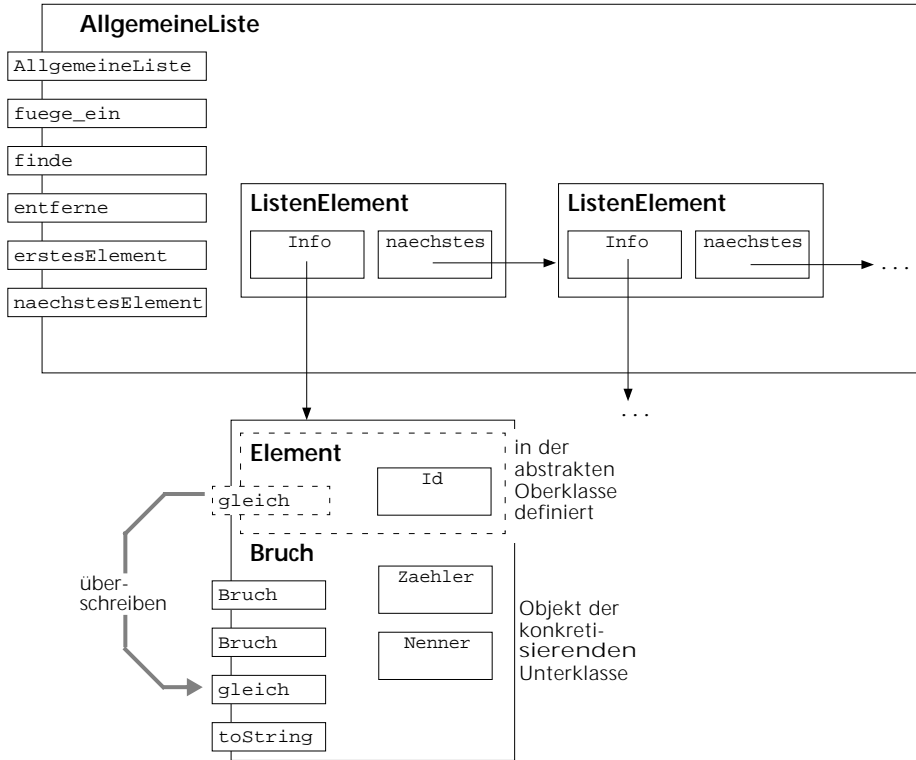
    int Zaehler, Nenner;

    boolean gleich (Element b)
    { if ( ! (b instanceof Bruch) )
      { System.out.println ("Es wurde kein gültiges Objekt vom Typ Bruch " +
                            "vergleichen.");
        return false;
      }
      else
      { Bruch rat = (Bruch) b;
        if (rat.Nenner * Nenner == 0)
            { System.out.println ("Es wurden keine gültigen Brüche verglichen.");
              return false;
            }
        else
            return (rat.Zaehler * Nenner == rat.Nenner * Zaehler);
      }
    }

    public String toString ()
    { return Zaehler + "/" + Nenner;
    }
}
```

**Prog. 7-8** Die Klasse Bruch als Erweiterung der abstrakten Klasse Element

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 7-2** Verwendung der Klasse `AllgemeineListe`

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
AllgemeineListe Brueche = new AllgemeineListe();
Bruch b = new Bruch (10, 12), r;

for (int i = 1; i <= 5; i++) Brueche.fuege_ein (new Bruch (i, i+1));

System.out.println ("Die Liste enthält die folgenden Brüche:");
r = (Bruch) Brueche.dasErsteElement ();
while (r != null)
{ System.out.print (r + " "); r = (Bruch) Brueche.dasNaechsteElement ();
}
System.out.println (" ");
System.out.print ("Der Bruch " + b + " ist in der Liste ");
if (Brueche.finde (b) != null) System.out.print ("vorhanden");
else System.out.print("nicht vorhanden");
```

**Prog. 7-9** Beispiel für die Verwendung der Klasse `AllgemeineListe` zur Verwaltung von Objekten der Klasse `Bruch`

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*

```
class Rechteck implements geometrisches_Objekt
{ public float Umfang () { ... }
  public float Flaeche () { ... }
  ... // evtl weitere Methoden , auch solche,
  ... // die nicht in der Schnittstelle verlangt werden.
}
```

**Prog. 7–10** Beispiel für die Verwendung eines Interfaces: Die Klasse Rechteck implementiert das Interface geometrisches\_Objekt (Abb. 7–3)

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
interface geometrisches_Objekt
{ float Umfang ();
  float Flaeche ();
  ...
}
```

Schlüsselwort  
Name der Interface-Definition  
abstrakte Methodendefinitionen

**Abb. 7-3** Ein Beispiel für eine Interface-Definition

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

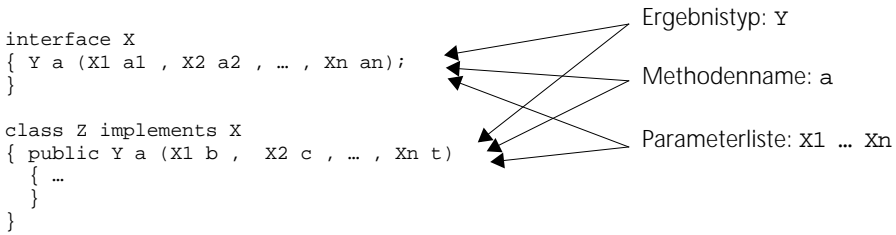
```
interface FarbManagement
{ int Weiss    = 0;
  int Schwarz  = 1;
  int Rot      = 2;
  void LegeFarbeFest (int farbe) ;
  int DerZeitigeFarbe () ;
}

class RGB implements FarbManagement
{ void LegeFarbeFest (int farbe)
  { ... verwandle farbe in das RGB - Modell
    ... setze farbe im RGB - Modell
  }

  int DerZeitigeFarbe ()
  { ... verwandle RGB --> Farbe des Interfaces FarbManagement
  }
}
```

**Prog. 7-11** Beispiel für Variablendeklaration in einer Interface-Definition

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 7-4** Beziehung zwischen Interface-Definition und einer implementierenden Klassendefinition

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
interface I
{ void a ();
  void b (int i);
}

interface J
{ void c ();
  void b (float f);
}

class B
{ ...
}

class C extends B implements I, J
{
  public void a () { ... // I erfordert die Realisierung von a in C.
  }

  public void c () { ... // J erfordert die Realisierung von c in C.
  }

  public void b (int i) { int g = i; ... // I erfordert die Realisierung
                               // von b mit int-Parameter in C.
  }

  public void b (float g) { float z = g; ... // J erfordert die Realisierung
                               // von b mit float-Parameter in C.
  }
}
```

**Prog. 7-12** Schematisches Beispiel für die Implementierung zweier Interfaces

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*

```
interface Element
{
    boolean gleich (Element zuvergleichen);
}
```

**Prog. 7-13** *Interface-Definition* Element für die Klasse AllgemeineListe

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class Bruch implements Element
{
    Bruch ()
    { this (0, 1);
    }

    Bruch (int Z, int N)
    { Zaehler = Z;  Nenner = N;
      if (Nenner == 0)
        System.out.println ("Es wurde ein unzulässiger Bruch erzeugt.");
    }

    int Zaehler, Nenner;

    public boolean gleich (Element b)
    { if (! (b instanceof Bruch) )
      { System.out.println ("Es wurde kein gültiges Objekt vom Typ Bruch " +
                            "verglichen.");
        return false;
      }
      else
      { Bruch rat = (Bruch) b;
        if (rat.Nenner * Nenner == 0)
          { System.out.println ("Es wurden keine gültigen Brüche verglichen");
            return false;
          }
          else
            return (rat.Zaehler * Nenner == rat.Nenner * Zaehler);
        }
      }

    public String toString ()
    { return Zaehler + "/" + Nenner;
    }
}
```

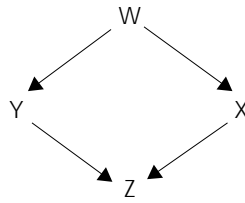
**Prog. 7-14** Die Klasse `Bruch` implementiert das Interface `Element`

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

*Definitionen von Interfaces und Klassen*

```
interface W { ... }  
interface X extends W { ... }  
interface Y extends W { ... }  
class Z implements X,Y { ... }
```

*Erweiterungsbeziehungen*



**Abb. 7-5** Beziehungsgeflecht bei mehrfacher Erweiterung von Interfaces

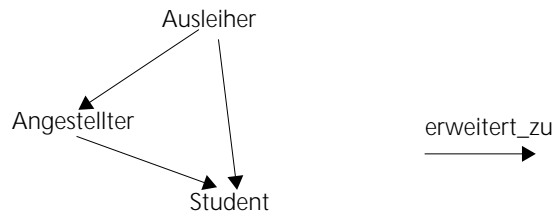
*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
interface SkatBlatt { int Kartenzahl = 32; }
interface PokerBlatt { int Kartenzahl = 52; }

class Spielkarten implements SkatBlatt, PokerBlatt
{ ...
  SkatBlatt.Kartenzahl ...
  ...
  PokerBlatt.Kartenzahl ...
  ...
}
```

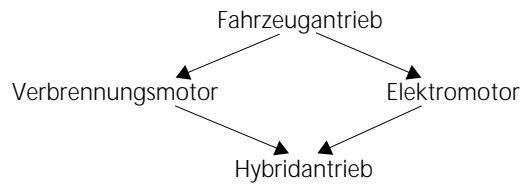
**Prog. 7-15** *Auflösung von identischen Variablennamen aus unterschiedlichen Interfaces*

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 7-6** Realistischere Modellierung der Beziehungen zwischen Ausleiher, Student und Angestelltem

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000



**Abb. 7-7** Struktur der Modellierung verschiedenartiger Fahrzeugantriebe

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
interface Fahrzeugantrieb
{ int Leistung ();          // in kW
  int Gewicht ();          // in kg
  int Drehmoment ();       // in Nm
}

interface Verbrennungsmotor extends Fahrzeugantrieb
{ float Verbrennungsraum ();
  Kraftstoff zuTanken ();
  float Verbrauch ();      // in km / 1000 ccm
}

interface Elektromotor extends Fahrzeugantrieb
{ Stromtyp VerwendeteStromArt (); //Geich/Wechsel
  boolean KannBremsEnergieWandeln ();
}

interface Hybridantrieb extends Verbrennungsmotor, Elektromotor
{ Koppelung verwendeteKoppelung ();
}
```

**Prog. 7-16** Interfaces zur Darstellung der Fahrzeugantrieb-Modellierung aus Abb. 7-7

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
class OttoMotor implements Verbrennungsmotor
{ int Leistung () { ... }
  float Verbrennungsraum () { ... }
  ...
}

class DrehstromMotor implements Elektromotor
{ int Leistung () { ... }
  boolean KannBremsEnergieWandeln () { ... }
  ...
}

class Otto_Drehstrom implements Hybridantrieb
{ OttoMotor      derVerbrennungsantrieb;
  DrehstromMotor derElektroantrieb;

  int Leistung ()
  { // Berechne die Leistung aus den Leistungsdaten der Komponenten:
    return ( ... derVerbrennungsantrieb.Listung () ...
              derElektroantrieb.Listung());
  }

  float Verbrennungsraum ()
  { return derVerbrennungsantrieb.Verbrennungsraum;
  }

  boolean KannBremsEnergieWandeln ()
  { return derElektroantrieb.kannBremsEnergieWandeln ();
  }
  ...
}
```

**Prog. 7-17** Definition konkreter Klassen auf der Basis der Interface-Struktur aus Prog. 7-16

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```

class Liste implements Liste_I
{
    class ListenElement
    {
        Element Info;
        ListenElement naechstes;

        ListenElement(Element Info)
        {this(Info,null);}

        ListenElement (Element Info, ListenElement naechstes) // Konstruktor für
        { this.Info = Info;   this.naechstes = naechstes;      // ListenElement.
        }
    }

    ListenElement erstes;

    Liste () // Konstruktor für Liste.
    { erstes = null;}

    public void fuege_ein (Element k)
    { erstes = new ListenElement (k, erstes);
    }

    public java.util.Enumeration dieElemente ()
    { return new java.util.Enumeration ()
    { ListenElement laufendes = erstes;

        public boolean hasMoreElements ()
        { return laufendes != null;
        }

        public Object nextElement ()
        { if (hasMoreElements ())
          { Element h = laufendes.Info;   laufendes = laufendes.naechstes;
            return h;
          }
          else
            return null;
        }
    };
    }

    public Element finde (Element zufinden)
    { if (zufinden != null)
      { java.util.Enumeration alleElemente = dieElemente ();
        Element dasElement;
        while (alleElemente.hasMoreElements ())
        { dasElement = (Element) alleElemente.nextElement ();
          if (dasElement != null)
            { if (dasElement.gleich (zufinden))
              return dasElement;
            }
            else return null;
          }
        }
      return null;
    }

    public boolean leer ()
    { return erstes == null;
    }
}

```

**Prog. 7-18** Die Definition einer einfachen Liste zum Suchen von Elementen

```
interface ElementKG extends Element
{
    boolean kleinerGleich (ElementKG zuvergleichen);
}
```

**Prog. 7-19** Das Interface für Listenelemente mit der Ordnung kleinerGleich

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

---

```
interface SortierteListe_I
{
    void fuege_ein (ElementKG einzufuegen);
    java.util.Enumeration dieElemente ();
    ElementKG finde (ElementKG z);
    boolean leer ();
}
```

**Prog. 7-20** *Das Interface SortierteListe\_I*

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```

class SortierteListe implements SortierteListe_I
{
    Liste dieListe;

    SortierteListe ()
    { dieListe = new Liste ();
    }

    public void fuege_ein (ElementKG k)
    { if (dieListe.leer ())
      dieListe.fuege_ein (k);
      else
      { Liste.ListenElement hh = dieListe.erstes;
        if (k.kleinerGleich ( (ElementKG) hh.Info))
        { Liste.ListenElement h = dieListe.new ListenElement (k);
          h.naechstes = dieListe.erstes;  dieListe.erstes = h;
        }
        else
        { boolean fertig = false;
          while ( ! fertig)
          { if (hh.naechstes == null)
            { fertig = true;
              hh.naechstes = dieListe.new ListenElement (k);
              hh.naechstes.naechstes = null;
            }
            else if (k.kleinerGleich ( (ElementKG) hh.naechstes.Info))
            { Liste.ListenElement hhh = dieListe.new ListenElement (k);
              hhh.naechstes = hh.naechstes;
              hh.naechstes = hhh;  fertig = true;
            }
          }
          else
            hh = hh.naechstes;
        }
      }
    }
}

public boolean leer ()
{ return dieListe.leer ();
}

public java.util.Enumeration dieElemente ()
{ return dieListe.dieElemente ();
}

public ElementKG finde (ElementKG zufinden)
{ return (ElementKG) dieListe.finde (zufinden);
}
}

```

### **Prog. 7-21** Die Klasse SortierteListe

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
class Bruch_le extends Bruch implements ElementKG
{
    Bruch_le (int z, int n)
    { super (z, n);
    }

    public boolean kleinerGleich (ElementKG b)
    { Bruch_le rat;
      if ( ! (b instanceof ElementKG))
      { System.out.println ("Es wurde kein gültiges Objekt vom Typ"
        + " Bruch_le verglichen");
        return false;
      }
      else
      { rat = (Bruch_le) b;
        if (rat.Nenner * Nenner == 0)
        { System.out.println ("Es wurden keine gültigen Brüche verglichen");
          return false;
        }
        else
          return (Zaehler * rat.Nenner <= rat.Zaehler * Nenner);
      }
    }
}
```

**Prog. 7-22** Klasse `Bruch_le` als Erweiterung der Klasse `Bruch` (siehe Prog. 7-14)

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
java.util.Enumeration dieBrueche;
SortierteListe Reihe = new SortierteListe ();
Bruch_le r;

for (int i = 1; i <= 5; i++)
{ Reihe.fuege_ein (new Bruch_le (i, i + 1));
};

dieBrueche = Reihe.dieElemente ();

while (dieBrueche.hasMoreElements ())
{ r = (Bruch_le) dieBrueche.nextElement ();
  System.out.print (r + " ");
}
System.out.println("");

Reihe.fuege_ein (new Bruch_le ( 1,  4));
Reihe.fuege_ein (new Bruch_le ( 5,  8));
Reihe.fuege_ein (new Bruch_le ( 5,  6));
Reihe.fuege_ein (new Bruch_le ( 7,  8));
Reihe.fuege_ein (new Bruch_le (11, 12));

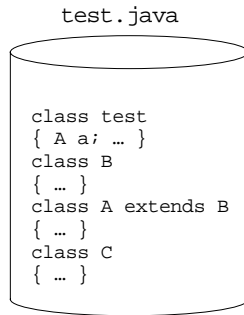
dieBrueche = Reihe.dieElemente ();

while (dieBrueche.hasMoreElements ())
{ r = (Bruch_le) dieBrueche.nextElement ();
  System.out.print (r + " ");
}
```

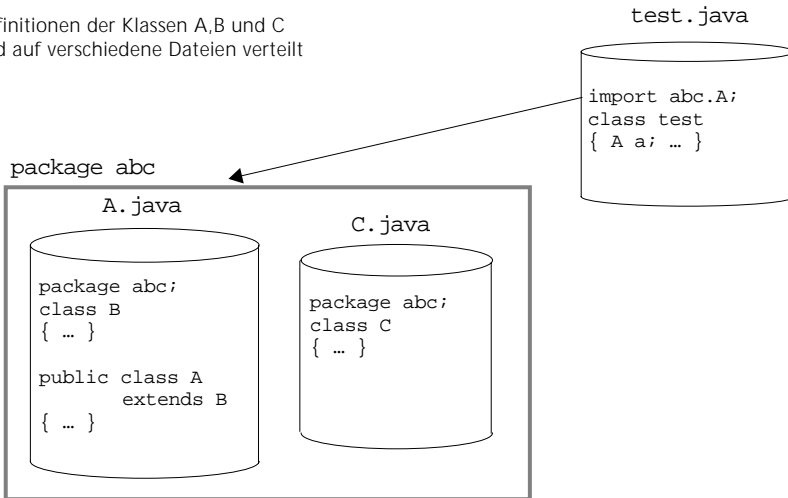
### **Prog. 7-23** Die Benutzung der Klassen `Bruch_le` und `SortierteListe`

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

bisher: alle Definitionen in einer Datei



hier: Definitionen der Klassen A,B und C sind auf verschiedene Dateien verteilt

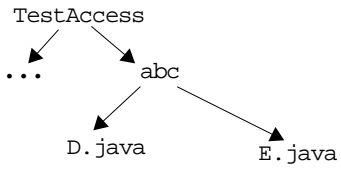


**Abb. 8-1** Die physische Struktur von Java-Programmen als Pakete

```
>mkdir TestAccess ← Erzeugen eines neuen Verzeichnisses
>cd TestAccess ← In das neue Verzeichnis wechseln
>mkdir abc ← Erzeugen eines neuen Verzeichnisses
>cd abc ← In das neue Verzeichnis wechseln
>vi D.java ← Die Datei D.java editieren
.....
>vi E.java ← Die Datei E.java editieren
.....
>cd .. ← In das übergeordnete Verzeichnis
TestAccess wechseln
>javac abc/E.java ← Übersetzen der beiden Dateien
>javac abc/D.java ←
```

**Prog. 8-1** Unix-Kommandos zum Erzeugen von Programmteilen für ein Paket

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 8-2** Verzeichnisstruktur bei der Verwendung des Pakets abc

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
// Inhalt der Datei D.java:

package abc;

public class D
{ E the_e_object;    // Benutzung der Klasse E als Datentyp einer Variablen.

  public D ()
  { System.out.println ("Ein Objekt der Klasse D wurde erzeugt.");
  }

  public void g ()
  { System.out.println ("g aufgerufen in einem Objekt der Klasse D.");
  }

  public E h ()      // Benutzung der Klasse E als Rückgabedatentyp von h.
  { E local_E;      // Benutzung der Klasse E als Datentyp einer Variablen.
    local_E = new E ();
    local_E.h ();
    return local_E;
  }
}
```

**Prog. 8–2** Definition der Klasse D in Paket abc

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
// Inhalt der Datei E.java:

package abc;

public class E
{ D the_d_object = new D ();      // Benutzung der Klasse D als Datentyp
                                // einer Variablen.

  public E ()
  { System.out.println ("Ein Objekt der Klasse E wurde erzeugt.");
  }

  public void g ()
  { System.out.println ("g aufgerufen in einem Objekt der Klasse E.");
    the_d_object.g ();
  }

  void h ()
  { System.out.println ("h aufgerufen in einem Objekt der Klasse E.");
  }
}
```

**Prog. 8-3** Definition der Klasse E in Paket abc

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

import abc.*;
public class TestAccess
{
    public static void main (String unbenutzt [])
    { System.out.println ("Hallo!");
      System.out.println ("Test von Klasse D.");
      D d = new D ();
      d.g ();
      E e = d.h ();
      e.g ();

      // e.h ();
      // wäre unzulässig, da diese
      // Methode h nur innerhalb des
      // Pakets abc verfuegbar ist.
    }
}

```

Import aller öffentlichen Datentypen aus dem Paket `abc`.

Der Aufruf von `h` erzeugt ein neues Objekt der Klasse `E`, welches wiederum ein Objekt der Klasse `D` erzeugt (siehe Definitionen von `E` in Prog. 8-3 und `D` in Prog. 8-2).

Der Aufruf von `g` wird erst lokal im Objekt `e` dann in dem zugeordneten `D`-Objekt ausgeführt.

Dieser Aufruf ist nicht erlaubt, da die Methode `h` nicht `public` ist.

**Abb. 8-3** Verwendung der Klassen `E` und `D` außerhalb des Paketes `abc`

```
Hallo!  
Test von Klasse D.  
Ein Objekt der Klasse D wurde erzeugt.  
g aufgerufen in einem Objekt der Klasse D.  
Ein Objekt der Klasse D wurde erzeugt.  
Ein Objekt der Klasse E wurde erzeugt.  
h aufgerufen in einem Objekt der Klasse E.  
g aufgerufen in einem Objekt der Klasse E.  
g aufgerufen in einem Objekt der Klasse D.
```

**Abb. 8-4** Ausgabe des Programms aus Abb. 8-3

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

## Paket Listen

### Interfaces der Klassenimplementierungen

Liste\_I.java  
SortierteListe\_I.java

### Interfaces der Elementklassen

Element.java  
ElementKG.java

### Klassenimplementierungen des Pakets **Listen**

Liste.java  
SortierteListe.java

### Benutzung des Pakets **Listen**

BruchListe.java

**Abb. 8-5** Links: sechs Dateien des Listen-Pakets, gegliedert in drei Gruppen zu je zwei Dateien, rechts: Datei des Programms, das eine Liste benutzt

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000

```
package Listen;

public interface Element
{ boolean gleich (Element zuvergleichen);
}
```

**Prog. 8-4** *Interface Element innerhalb des Pakets Listen*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
package Listen;

public interface ElementKG extends Element
{
    boolean kleinerGleich (ElementKG zuvergleichen);
}
```

**Prog. 8-5** Interface `ElementKG` innerhalb des Pakets `Listen`

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
package Listen;

interface Liste_I
{ void fuege_ein (Element k);
  boolean leer ();
  java.util.Enumeration dieElemente ();
  Element finde (Element z);
}
```

**Prog. 8-6** Interface Liste\_I innerhalb des Pakets Listen

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
package Listen;

interface SortierteListe_I
{ void fuege_ein (ElementKG k);
  boolean leer ();
  java.util.Enumeration dieElemente ();
  ElementKG finde (ElementKG z);
}
```

**Prog. 8-7** Interface `SortierteListe_I` innerhalb des Pakets `Listen`

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

package Listen;

public class Liste implements Liste_I
{
    class ListenElement
    { Element Info; ListenElement naechstes;

        ListenElement(Element Info)
        {this(Info,null);}

        ListenElement (Element Info, ListenElement naechstes)
        { this.Info = Info; this.naechstes = naechstes;
        }
    }

    ListenElement erstes;

    public Liste ()
    { erstes = null;
    }

    public void fuege_ein (Element k)
    { erstes = new ListenElement (k, erstes);
    }

    public java.util.Enumeration dieElemente ()
    { return new java.util.Enumeration ()
    { ListenElement laufendes = erstes;

        public boolean hasMoreElements ()
        { return laufendes != null;
        }

        public Object nextElement ()
        { if (hasMoreElements ())
        { Element h = laufendes.Info; laufendes = laufendes.naechstes;
        return h;
        }
        else return null;
        }
    };
    }

    public Element finde (Element zufinden)
    { if (zufinden != null)
    { java.util.Enumeration alleElemente = dieElemente ();
    Element dasElement;
    while (alleElemente.hasMoreElements ())
    { dasElement = (Element) alleElemente.nextElement ();
    if (dasElement != null)
    { if (dasElement.gleich (zufinden)) return dasElement;
    }
    else return null;
    }
    }
    return null;
    }

    public boolean leer ()
    { return erstes == null;
    }
}

```

**Prog. 8-8** Klasse `Liste` implementiert das Interface `Liste_I` im Paket `Listen`

```

package Listen;

public class SortierteListe implements SortierteListe_I
{
    Liste dieListe;

    public SortierteListe ()
    { dieListe = new Liste();
    }

    public void fuege_ein (ElementKG k)
    { if (dieListe.leer ())
      dieListe.fuege_ein (k);
      else
      { Liste.ListenElement hh = dieListe.erstes;
        if (k.kleinerGleich ( (ElementKG) hh.Info))
        { Liste.ListenElement h = dieListe.new ListenElement (k);
          h.naechstes = dieListe.erstes; dieListe.erstes = h;
        }
        else
        { boolean fertig = false;
          while ( ! fertig)
          { if (hh.naechstes == null)
            { fertig = true;
              hh.naechstes = dieListe.new ListenElement (k);
              hh.naechstes.naechstes = null;
            }
            else if (k.kleinerGleich ( (ElementKG) hh.naechstes.Info))
            { Liste.ListenElement hhh = dieListe.new ListenElement (k);
              hhh.naechstes = hh.naechstes;
              hh.naechstes = hhh; fertig = true;
            }
            else
            { hh= hh.naechstes;
            }
          }
        }
      }
    }

    public boolean leer ()
    { return dieListe.leer ();
    }

    public java.util.Enumeration dieElemente ()
    { return dieListe.dieElemente();
    }

    public ElementKG finde (ElementKG zufinden)
    { return (ElementKG) dieListe.finde (zufinden);
    }
}

```

**Prog. 8-9** Klasse *SortierteListe* implementiert das Interface *SortierteListe\_I* im Paket *Listen*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

// BruchListe
import Listen.*;

public class BruchListe
{
    public static void main (String unbenutzt [])
    {
        System.out.println ("Hier ist das Programm zur Verwaltung von Brüchen.");
        Liste Brueche = new Liste();
        Bruch rat = new Bruch (10, 12), r;
        java.util.Enumeration dieBrueche;

        for (int i = 1; i <= 5; i++)
            Brueche.fuege_ein (new Bruch (i, i + 1));
        System.out.println ("Die Liste enthaelt die folgenden Brüche:");
        dieBrueche = Brueche.dieElemente ();

        while (dieBrueche.hasMoreElements ())
        {
            r = (Bruch) dieBrueche.nextElement ();
            System.out.print (r + " ");
        }
        System.out.print ("\n und der Bruch " + rat + " ist in der Liste");
        if (Brueche.finde (rat) != null) System.out.print (" vorhanden.");
        else System.out.print (" nicht vorhanden.");

        SortierteListe Reihe = new SortierteListe ();

        System.out.println ("Nun werden sortierte Listen verwendet.");

        for (int i = 1; i <= 5; i++)
            Reihe.fuege_ein (new Bruch_le (i, i + 1));

        dieBrueche = Reihe.dieElemente ();

        while (dieBrueche.hasMoreElements ())
        {
            r = (Bruch_le) dieBrueche.nextElement ();
            System.out.print (r + " ");
        }

        Reihe.fuege_ein (new Bruch_le ( 1, 4));
        Reihe.fuege_ein (new Bruch_le ( 5, 8));
        Reihe.fuege_ein (new Bruch_le ( 5, 6));
        Reihe.fuege_ein (new Bruch_le ( 7, 8));
        Reihe.fuege_ein (new Bruch_le (11, 12));

        dieBrueche = Reihe.dieElemente ();

        while (dieBrueche.hasMoreElements ())
        {
            r = (Bruch_le) dieBrueche.nextElement ();
            System.out.print (r + " ");
        }
    }
}

```

**Prog. 8–10** Fragment der Klasse `BruchListe`, welches das Paket `Listen` importiert (verwendet `Bruch` aus Prog. 7–14 und `Bruch_le` aus Prog. 7–22 in unveränderter Form)

```
try { ... // Normalblock }
catch (AusnahmeArt1 Parameter1) { ... // Ausnahmebehandler 1 }
catch (AusnahmeArt2 Parameter2) { ... // Ausnahmebehandler 2 }
```

← Block von Anweisungen, die potenziell eine Ausnahme erzeugen können

← Behandlung von Ausnahmen vom Typ AusnahmeArt1

← Behandlung von Ausnahmen vom Typ AusnahmeArt2

**Abb. 8-6** Struktur der try-catch-Anweisung

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
static int Eingabe ()
{ String s = ""; int i;
  try { s = new java.io.DataInputStream (System.in).readLine ();
        i = java.lang.Integer.parseInt (s);
      }
  catch (java.io.IOException e)
    { System.out.println (e);
    }
  catch (java.lang.NumberFormatException e)
    { System.out.println ("Keine ganze Zahl: " + e.getMessage ());
      i = 0;
    }
  return i;
}
```

**Prog. 8-11** Methode Eingabe mit einfacher Ausnahmebehandlung

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

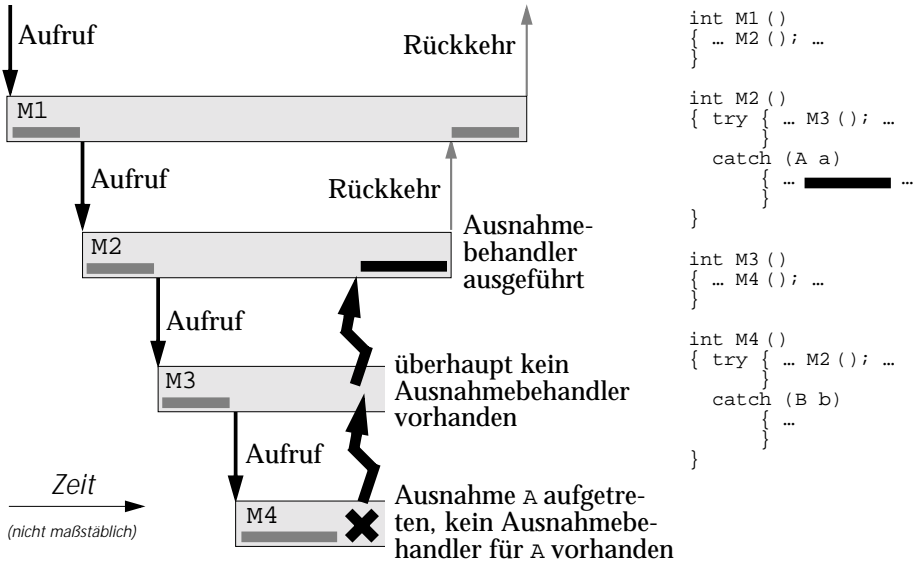
```
... // Liste wie in Prog. 4-6 bis 4-7 definiert.

public class Haupt
{
    public static void main (String [] unbenutzt)
    { Liste L = new Liste ();
      ... // Anweisungen zum Aufbau der Liste.
      System.out.println (pruefeNachbarn (L));
    }

    static String pruefeNachbarn (Liste L)
    { try { Element x = L.Kopf;
          while (x != null)
            { if (x.Zahl == x.Nf.Zahl) return "Benachbart: " + x.Zahl;
              x = x.Nf;
            }
          return "Keine gleichen Zahlen in benachbarten Elementen.";
        }
      catch (NullPointerException leererVerweis)
        { return "Keine gleichen Zahlen in benachbarten Elementen.";
        }
    }
}
```

### **Prog. 8-12** Verwendung der `NullPointerException`

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 8-7** Weiterleitung einer Ausnahme  $A$ , die lokal nicht behandelbar ist

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
... // Liste wie in Prog. 4-6 bis 4-7 definiert.

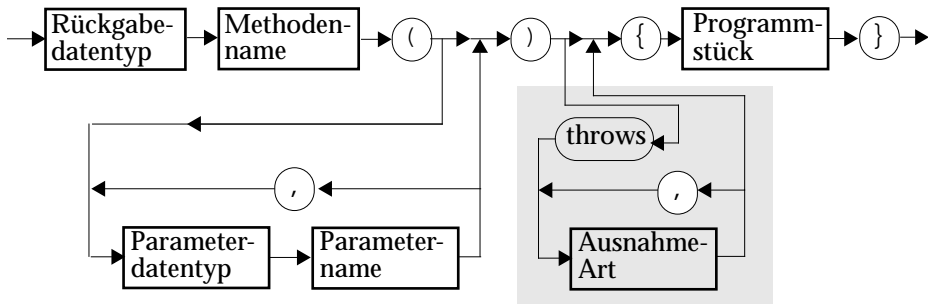
class ungerade extends java.lang.Throwable
{ int Anzahl;

  ungerade (int Anzahl)
  { this.Anzahl = Anzahl;
    System.out.println ("Ungerade Elementanzahl " + Anzahl);
  }
}

public class Haupt
{
  public static void main (String[] unbenutzt)
  { Liste L = new Liste (); int n;
    ... // Anweisungen zum Aufbau der Liste.
    try { ...
      n = ... // Anzahl der Listenelemente.
      if (n%2 == 1) throw new ungerade (n);
      ...
    }
    catch (ungerade uAusnahme)
    { L.erzeuge_Fuss (0); n = uAusnahme.Anzahl + 1;
    }
  }
}
```

**Prog. 8-13** Im Programm definierte Ausnahme ungerade

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 8-8** Syntax eines Methodenkopfs mit der `throws`-Liste (grau unterlegt)

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class ausgefallen extends java.lang.Throwable
{ ...
}

class ...
{ ...

    void M1 ()
    { ...
      try { ... M2 () ...
          }
      catch (ausgefallen a)
          { ... // Ausnahmebehandler für ausgefallen.
            }
    }

    void M2 () throws ausgefallen
    { ... M3 () ...
    }

    void M3 () throws ausgefallen
    { ...
      if ( ... Leitung_tot ... ) throw new ausgefallen ();
      ...
    }
}
```

**Prog. 8-14** Mittels `throw` weitergeleitete Ausnahme `ausgefallen`

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class BspThread extends java.lang.Thread
{ String Name; int Anzahl;

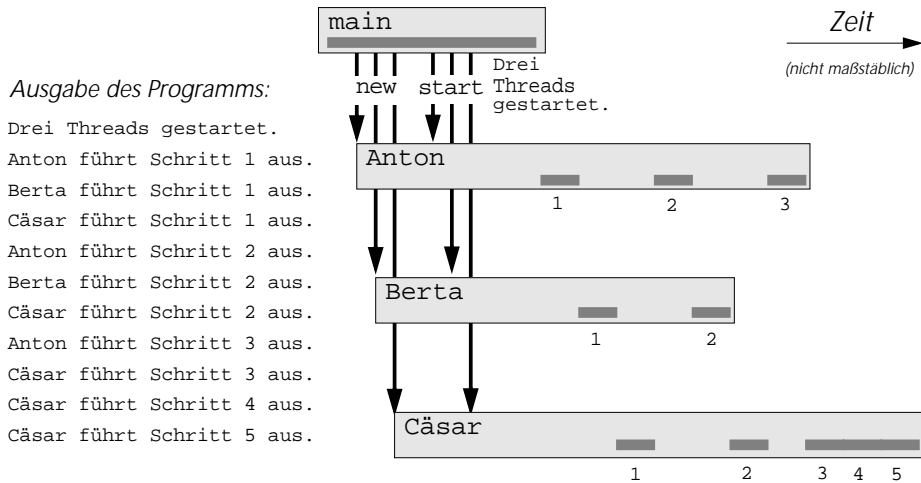
  BspThread (String Name, int Anzahl)
  { this.Name = Name; this.Anzahl = Anzahl;
  }

  public void run ()
  { for (int i = 1; i <= Anzahl; i++)
    { System.out.println (Name + " führt Schritt " + i + " aus.");
      yield ();
    }
  }
}

public class Haupt
{ public static void main (String [] unbenutzt)
  { BspThread a = new BspThread ("Anton", 3),
    b = new BspThread ("Berta", 2),
    c = new BspThread ("Cäsar", 5);
    a.start (); b.start (); c.start ();
    System.out.println ("Drei Threads gestartet.");
  }
}
```

**Prog. 8-15** Threads werden quasiparallel ausgeführt

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*



**Abb. 8-9** Ausgabe und Zeitverhalten der Threads von Prog. 8-15

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
class GemeinsameVariable
{ int Info = 100;

  int lies ()
  { return Info ;
  }

  void schreibe (int z)
  { Info = z;
  }
}
```

**Prog. 8-16** Einkapselung der gemeinsam benutzten int-Variablen Info

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

class Zugreifer extends java.lang.Thread
{ int Nr, delta, Zwischenwert;
  GemeinsameVariable gemeinsam;

  Zugreifer (int Nr, int delta, GemeinsameVariable gemeinsam)
  { this.Nr = Nr;  this.delta = delta;  this.gemeinsam = gemeinsam;
  }

  public void run ()
  { for (int i = 1; i <= 5; i++)
    { Zwischenwert = gemeinsam.lies ();
      drucke (Nr, "liest", Zwischenwert);      Zeitdauer ();
      Zwischenwert += delta;
      gemeinsam.schreibe (Zwischenwert);
      drucke (Nr, "schreibt", Zwischenwert);  Zeitdauer ();
    }
  }

  void drucke (int Nr, String Op, int Wert)
  { String Zugr = "Zugreifer " + Nr, Pfeil,
    Abstand = "          ";
    if (Nr == 1 && Op.equals ("liest")) Pfeil = " <----liest--- ";
    else if (Nr == 1) Pfeil = " --schreibt--> ";
    else if (Op.equals ("liest")) Pfeil = " ---liest----> ";
    else Pfeil = " <--schreibt-- ";
    if (Nr == 1) System.out.println (Zugr + Pfeil + Wert);
    else System.out.println (Abstand + Wert + Pfeil + Zugr);
  }

  void Zeitdauer ()
  { long Dauer = (long) (Math.random () * 300);
    try { Thread.currentThread ().sleep (Dauer);
      }
    catch (InterruptedException Ausnahme) {}
  }
}

```

**Prog. 8-17** Klasse der Threads, die auf GemeinsameVariable zugreifen

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
public class Haupt
{ public static void main (String [] unbenutzt)
  { GemeinsameVariable v = new GemeinsameVariable ();
    Zugreifer z1 = new Zugreifer (1, 1, v);
    Zugreifer z2 = new Zugreifer (2, 2, v);
    z1.start ();    z2.start ();
  }
}
```

**Prog. 8-18** Hauptprogramm für den Zugriff von zwei Threads auf eine gemeinsame Variable

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

```
Zugreifer 1 <----liest--- 100
100 ---liest----> Zugreifer 2
102 <--schreibt-- Zugreifer 2
Zugreifer 1 --schreibt--> 101
101 ---liest----> Zugreifer 2
103 <--schreibt-- Zugreifer 2
Zugreifer 1 <----liest--- 103
103 ---liest----> Zugreifer 2
105 <--schreibt-- Zugreifer 2
Zugreifer 1 --schreibt--> 104
104 ---liest----> Zugreifer 2
106 <--schreibt-- Zugreifer 2
Zugreifer 1 <----liest--- 106
Zugreifer 1 --schreibt--> 107
Zugreifer 1 <----liest--- 107
107 ---liest----> Zugreifer 2
109 <--schreibt-- Zugreifer 2
Zugreifer 1 --schreibt--> 108
Zugreifer 1 <----liest--- 108
Zugreifer 1 --schreibt--> 109
```

**Abb. 8-10** Ausgabe zu Prog. 8-18

```
Zugreifer 1 <----liest--- 100
              100 ---liest----> Zugreifer 2
              102 <--schreibt-- Zugreifer 2
Zugreifer 1 --schreibt--> 101
              101 ---liest----> Zugreifer 2
Zugreifer 1 <----liest--- 101
Zugreifer 1 --schreibt--> 102
              103 <--schreibt-- Zugreifer 2
              103 ---liest----> Zugreifer 2
Zugreifer 1 <----liest--- 103
Zugreifer 1 --schreibt--> 104
Zugreifer 1 <----liest--- 104
              105 <--schreibt-- Zugreifer 2
              105 ---liest----> Zugreifer 2
Zugreifer 1 --schreibt--> 105
Zugreifer 1 <----liest--- 105
              107 <--schreibt-- Zugreifer 2
Zugreifer 1 --schreibt--> 106
              106 ---liest----> Zugreifer 2
              108 <--schreibt-- Zugreifer 2
```

**Abb. 8-11** Andere Ausgabe zu Prog. 8-18

```
class GemeinsameVariable
{ int Info = 100;

  synchronized int [] update (int Summand)
  { int [] Aenderung = new int [2];   Aenderung [0] = Info;
    Info = Info + Summand;           Aenderung [1] = Info;
    return Aenderung;
  }
}
```

**Prog. 8-19** *Veränderung der int-Variablen Info durch eine synchronized-Methode*

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

class Zugreifer extends java.lang.Thread
{
    int Nr, delta;
    GemeinsameVariable gemeinsam;

    Zugreifer (int Nr, int delta, GemeinsameVariable gemeinsam)
    {
        this.Nr = Nr;    this.delta = delta;    this.gemeinsam = gemeinsam;
    }

    public void run ()
    {
        for (int i = 1; i <= 5; i++)
        {
            int [] alt_neu = gemeinsam.update (delta);
            drucke (Nr, "vor", alt_neu [0]);
            drucke (Nr, "nach", alt_neu [1]);    Zeitdauer ();
        }
    }

    void drucke (int Nr, String Op, int Wert)
    {
        String Zugr = "Zugreifer " + Nr, Pfeil,
            Abstand = " ";
        if (Nr == 1 && Op.equals ("vorher"))    Pfeil = " <-----vor---- ";
        else if (Nr == 1)    Pfeil = " ----nach----> ";
        else if (Op.equals ("vorher"))    Pfeil = " ----vor-----> ";
        else    Pfeil = " <----nach---- ";
        if (Nr == 1)    System.out.println (Zugr + Pfeil + Wert);
        else    System.out.println (Abstand + Wert + Pfeil + Zugr);
    }

    void Zeitdauer ()
    {
        long Dauer = (long) (Math.random () * 300);
        try { Thread.currentThread ().sleep (Dauer);
            }
        catch (InterruptedException Ausnahme) {}
    }
}

```

**Prog. 8-20** Verbesserte Klasse der Zugreifer-Threads nutzt synchronized-Methode update

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

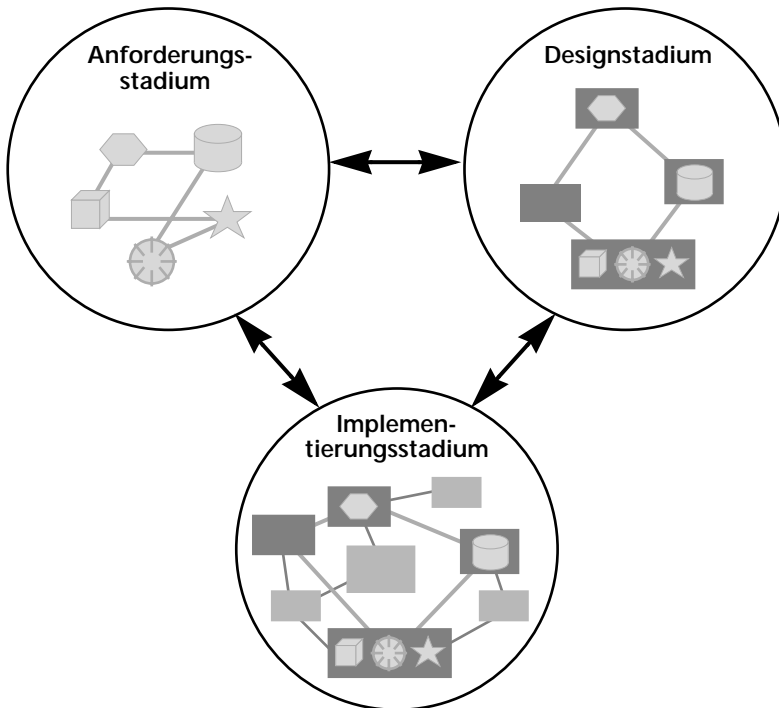
```
Zugreifer 1 <-----vor---- 100
Zugreifer 1 ----nach----> 101
                               101 ----vor-----> Zugreifer 2
                               103 <-----nach---- Zugreifer 2
Zugreifer 1 <-----vor---- 103
Zugreifer 1 ----nach----> 104
                               104 ----vor-----> Zugreifer 2
                               106 <-----nach---- Zugreifer 2
Zugreifer 1 <-----vor---- 106
Zugreifer 1 ----nach----> 107
                               107 ----vor-----> Zugreifer 2
                               109 <-----nach---- Zugreifer 2
Zugreifer 1 <-----vor---- 109
Zugreifer 1 ----nach----> 110
                               110 ----vor-----> Zugreifer 2
                               112 <-----nach---- Zugreifer 2
Zugreifer 1 <-----vor---- 112
Zugreifer 1 ----nach----> 113
                               113 ----vor-----> Zugreifer 2
                               115 <-----nach---- Zugreifer 2
```

**Abb. 8-12** Ausgabe des verbesserten Programms (Prog. 8-19 und Prog. 8-20)

*Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000*

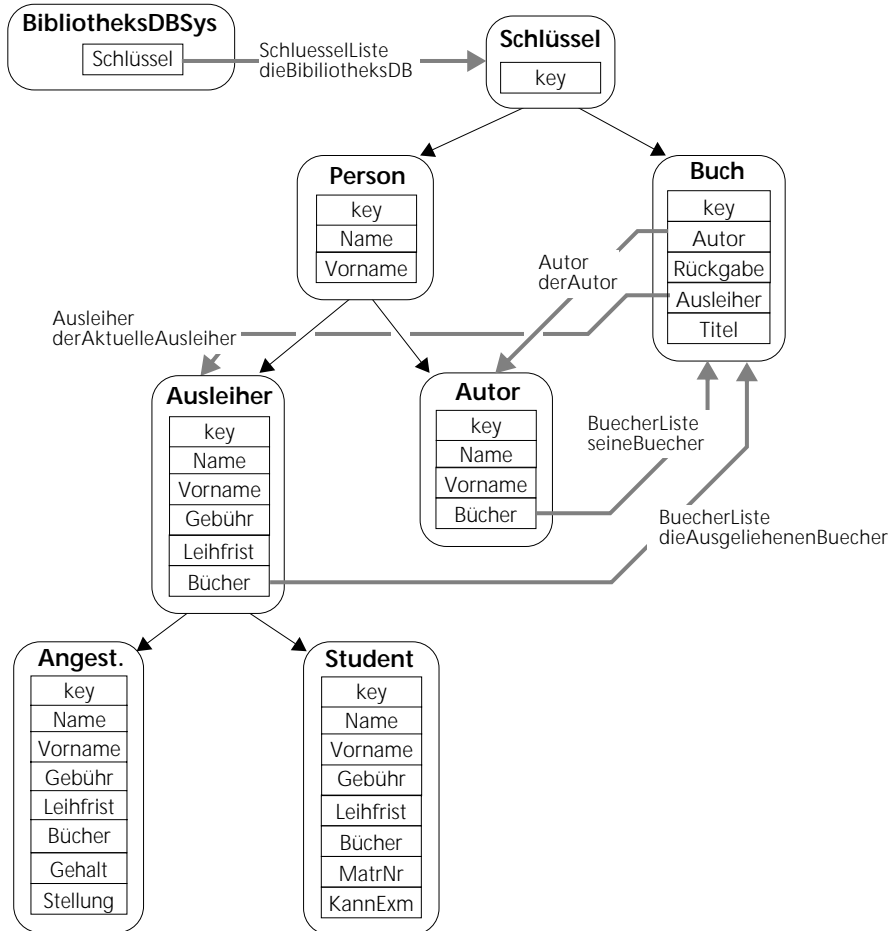
```
Zugreifer 1 <-----vor---- 100
Zugreifer 1 ----nach----> 101
                               101 ----vor-----> Zugreifer 2
                               103 <-----nach---- Zugreifer 2
Zugreifer 1 <-----vor---- 103
Zugreifer 1 ----nach----> 104
                               104 ----vor-----> Zugreifer 2
                               106 <-----nach---- Zugreifer 2
Zugreifer 1 <-----vor---- 106
Zugreifer 1 ----nach----> 107
Zugreifer 1 <-----vor---- 107
Zugreifer 1 ----nach----> 108
                               108 ----vor-----> Zugreifer 2
                               110 <-----nach---- Zugreifer 2
Zugreifer 1 <-----vor---- 110
Zugreifer 1 ----nach----> 111
                               111 ----vor-----> Zugreifer 2
                               113 <-----nach---- Zugreifer 2
                               113 ----vor-----> Zugreifer 2
                               115 <-----nach---- Zugreifer 2
```

**Abb. 8-13** Ausgabe eines weiteren Laufs des verbesserten Programms



**Abb. 9-1** Stadien der Softwareentwicklung

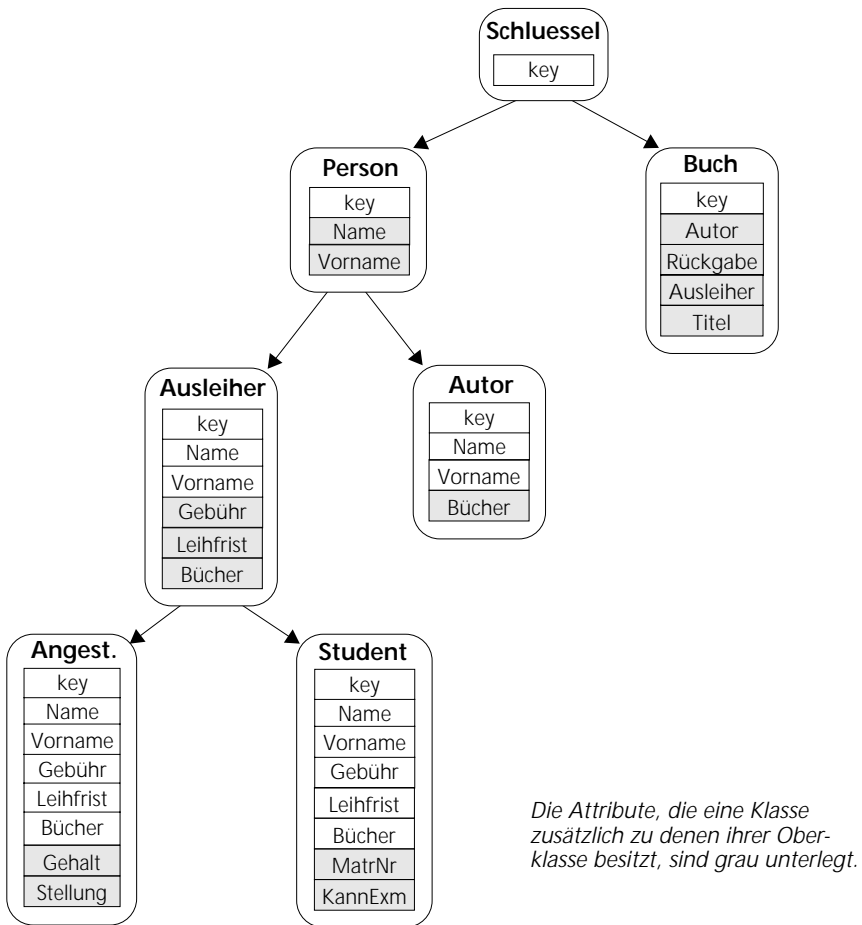
Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000



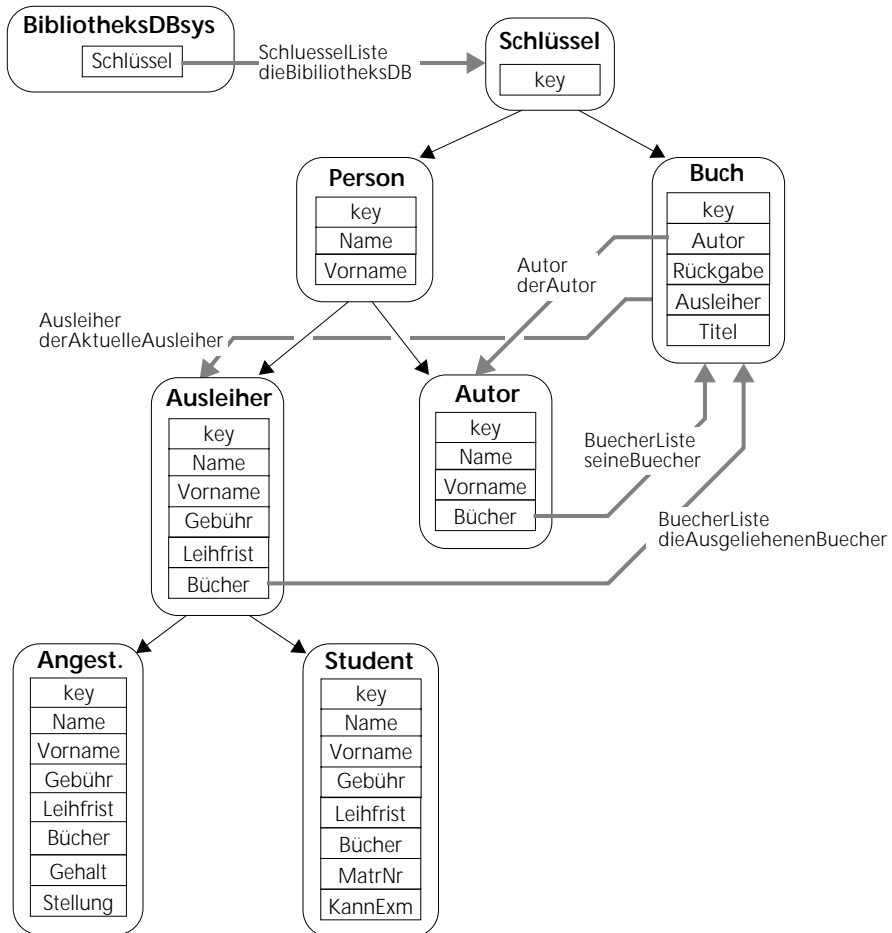
**Abb. 9-2** Ausschnitt aus dem Modell eines Bibliotheksverwaltungssystems

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

A



**Abb. A-1** Klassenhierarchie des Bibliotheksverwaltungssystems



**Abb. A-2** Anreicherung eines Ausschnitts des Bibliotheksmodells aus Abb. A-1 um die wichtigsten Benutzt-Beziehungen

```
import KEB.*;
```

```
class Schluessel  
{ String Id ()  
  { return "Typ Schluessel, Nr: " + key;  
  }  
}
```

Diese Methode liefert eine textuelle Information über Schluessel-Objekte

```
EinAusRahmen EA;  
int key;  
static int LetzterSchluessel = 0;
```

```
Schluessel (EinAusRahmen einaus)  
{ EA = einaus;  
  key = LetzterSchluessel++;  
}
```

Der Konstruktor setzt den Verweis **EA** auf ein **EinAusRahmen**-Objekt und erzeugt eine neue, eindeutige Schlüsselnummer, indem die statische Variable **LetzterSchluessel** inkrementiert wird.

```
void DruckeInfo ()  
{ EA.zeigeAus (Id ());  
}  
}
```

### **Prog. A-1** Die Klasse Schluessel

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

import KEB.*;

class SchluesselListe
{
    class Element ← lokale Klasse zur Speicherung der Listenstrukturen
    {
        Schluessel Info;
        Element naechstes;

        Element (Schluessel k)
        { Info = k;
        }
    }

    Element erstes, laufendes;
    EinAusRahmen EA;

    SchluesselListe(EinAusRahmen einaus)
    { EA = einaus;  erstes = null;  laufendes = erstes;
    }

    Schluessel dasErsteElement()
    { if (erstes == null)
      return null;
      else
      { laufendes = laufendes;
        return laufendes.Info; ← dient dazu, die Liste durchlaufen zu können
      }
    }

    Schluessel dasNaechsteElement()
    { if (laufendes == null)
      return null;
      else
      { laufendes = laufendes.naechstes;
        // Achtung: Nun kann laufendes auf null verweisen,
        // wenn es vorher auf das letzte Element gezeigt hat.
        if (laufendes == null) return null;
        else return laufendes.Info;
      }
    }

    void einfuegen (Schluessel k)
    { Element h = new Element (k);
      h.naechstes = erstes;
      erstes = h;
    }

    void DruckeAlle ()
    { Element el = erstes;
      while (el != null)
      { EA.zeigeAus (el.Info.Id());  el = el.naechstes;
      }
    }

    void DruckeAlleInfos ()
    { Element el = erstes;
      while (el != null)
      { el.Info.DruckeInfo();  el = el.naechstes;
      }
    }
} // end class SchluesselListe

```

entweder ist laufendes == null (im Fall der leeren Liste) oder laufendes zeigt auf das Element, das im vorangehenden Aufruf zurückgeliefert wurde (also aufpassen beim letzten Element).

Die Methode entferne ist in Prog. A-3 (Seite 336), die Methode finde in Prog. A-4 (Seite 337) dargestellt.

### Prog. A-2 Die Klassen SchluesselListe und Element

```
void entferne (Schluessel k)
{
    Element laufendes, vor;
    boolean fertig;
    // Entferne ein Element aus einer einfach verketteten Liste.
    // Erster Fall: Die Liste ist leer. Dann geschieht nichts.
    // Zweiter Fall: Die Liste ist nicht leer ...

    if (erstes != null)
    { // Das zu entfernende Element ist das erste der Liste.
        if (erstes.Info == k)
            erstes = erstes.naechstes;
        else
        { // Das zu entfernende Element ist nicht das erste der Liste.
            // Suche es durch Nachziehen des Zeigers vor.
            vor = erstes; laufendes = erstes.naechstes;
            fertig = false;

            while ( ! fertig)
            {
                if (laufendes == null) // Das zu entfernende Element
                    fertig = true; // ist nicht in der Liste.
                else if (laufendes.Info == k)
                { // Das Element ist gefunden und kann nun geloescht werden.
                    vor = laufendes.naechstes; laufendes.naechstes = null;
                    fertig = true;
                }
                else
                { vor = laufendes; laufendes = laufendes.naechstes;
                }
            }
        }
    }
}
```

**Prog. A-3** Methode entferne der Klasse SchluesselListe (Prog. A-2)

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

```
Schlüssel finde (int zufinden)
{ Schlüssel dasElement = dasErsteElement ();
  boolean fertig = false;
  Schlüssel gefunden = null;

  while ( ! fertig )
  { if (dasElement == null)
    { fertig = true;   gefunden = null;   // nichts gefunden.
    }
    else if (dasElement.key == zufinden)
    { fertig = true;   gefunden = dasElement;
    }
    else
      dasElement = dasNaechsteElement ();
  }
  return gefunden;
}
```

**Prog. A-4** Methode finde der Klasse SchlüsselListe (Prog. A-2 auf Seite 335)

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
import java.util.GregorianCalendar;
import KEB.EinAusRahmen;

class Buch extends Schluessel
{
    String Id ()
    { return "Typ Buch, Nr: " + key;
    }

    String Titel;
    Autor derAutor;

    GregorianCalendar RueckgabeDatum;

    Ausleiher derAktuelleAusleiher;

    Buch (EinAusRahmen ea, String DerTitel, Autor P)
    { super (ea); Titel = DerTitel; derAutor = P;
    }

    void DruckeInfo ()
    { EA.zeigeAus(Id() + ": " + Titel + " von Autor " + derAutor.Id ());
      if (derAktuelleAusleiher != null)
      { EA.zeigeAus ("ausgeliehen von:"); derAktuelleAusleiher.DruckeInfo ();
      }
      else
      EA.zeigeAus ("nicht ausgeliehen");
    }
}
```

Paket zur Darstellung des üblichen Kalenders, hier RückgabeDatum

### **Prog. A-5** Die Klasse Buch

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
import KEB.EinAusRahmen;

class BuecherListe
{ SchluesselListe dieBuecherListe;

  BuecherListe (EinAusRahmen ea)
  { dieBuecherListe = new SchluesselListe (ea);
  }

  void einfuegen (Buch b)
  { dieBuecherListe.einfuegen(b);
  }

  Buch dasErsteElement ()
  { return (Buch) dieBuecherListe.dasErsteElement ();
  }

  Buch dasNaechsteElement ()
  { return (Buch) dieBuecherListe.dasNaechsteElement ();
  }

  void entferne (Buch b)
  { dieBuecherListe.entferne (b);
  }

  Buch finde (int key)
  { Schluessel dasBuch = dieBuecherListe.finde (key);
    if (dasBuch == null) return null;
    else if (dasBuch instanceof Buch) return (Buch) dasBuch;
    else return null;
  }

  void DruckeAlle ()
  { dieBuecherListe.DruckeAlle ();
  }

  void DruckeInfos ()
  { dieBuecherListe.DruckeAlleInfos ();
  }
}
```

### **Prog. A-6** Die Klasse BuecherListe

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
import KEB.EinAusRahmen;

class Person extends Schluessel
{ String Name, Vorname;

  String Id ()
  { return "Typ Person, Nr: " + key;
  }

  Person (EinAusRahmen einaus, String N, String V)
  { super (einaus);  Name = N;  Vorname = V;
  }

  void DruckeInfo ()
  { EA.zeigeAus (Id () + ":" + Vorname + "," + Name);
  }
}
```

**Prog. A-7** Die Klasse Person

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
import KEB.EinAusRahmen;

class Autor extends Person
{ BuecherListe seineBuecher;

  String Id ()
  { return "Typ Autor, Nr:" + key;
  }

  void neuesBuch (Buch b)
  { seineBuecher.einfuegen (b);
  }

  Autor (EinAusRahmen ea, String N, String V)
  { super (ea,N,V);   seineBuecher = new BuecherListe (ea);
  }

  void DruckeInfo()
  { EA.zeigeAus (Id () + ":" + Vorname + ", " + Name +
    " hat die folgenden Bücher geschrieben:");
    seineBuecher.DruckeAlle ();
  }
}
```

**Prog. A-8** Die Klasse Autor

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```

import KEB.*;
import java.util.Calendar;

class Ausleiher extends Person
{
    String Id ()
    { return "Typ Ausleiher, Nr: " + key;
    }

    int Leihfrist;    // Ausleihfrist in Tagen.
    int Gebuehr;     // Aufgelaufene Gebühr in Euro.
    BuecherListe dieAusgeliehenenBuecher;

    Ausleiher (EinAusRahmen einaus, String N, String V)
    { super (einaus, N, V);   Leihfrist = 30;
      dieAusgeliehenenBuecher = new BuecherListe(einaus);
    }

    void DruckeMahnung ()
    { EA.zeigeAus ("Mahnung: " + Gebuehr + " Euro an Gebuehren" +
                  " aufgelaufen");
    }

    void BerechneGebuehr (Calendar heute)
    { // Pro ueberzogenem Tag einen Euro! Ein Jahr hat 360 Tage ...
      // Berechne die Anzahl der Tage, die ueberzogen sind.
      int Porto = 1; // Hypothetische Standardbriefgebuehr in Euro.
      Buch laufendesb = dieAusgeliehenenBuecher.dasErsteElement ();

      while (laufendesb != null)
      { if (heute.after (laufendesb.RueckgabeDatum))
        { int jahr1 = heute.get (Calendar.YEAR);
          int jahr0 = laufendesb.RueckgabeDatum.get (Calendar.YEAR);
          int tag1 = heute.get (Calendar.DAY_OF_YEAR);
          int tag0 = laufendesb.RueckgabeDatum.get (Calendar.DAY_OF_YEAR);
          int diff = (jahr1 - jahr0) * 360 + (tag1 - tag0);
          EA.zeigeAus (diff + " Tage ueberzogen für Ausleiher " + Id () +
                      " und Buch " + laufendesb.Id ());
          Gebuehr = Gebuehr + diff;    // sehr restriktiv und teuer.
        }
        laufendesb = dieAusgeliehenenBuecher.dasNaechsteElement ();
      }
      // Falls eine Gebuehr festgestellt wurde, noch das Porto addieren:
      if (Gebuehr > 0) Gebuehr = Gebuehr + Porto;
    }

    void MahnungBearbeiten (Calendar heute)
    { BerechneGebuehr (heute);
      if (Gebuehr > 0)
      { EA.zeigeAus ("\nAn " + Name + " , " + Vorname + " Ihre Nummer: " + key);
        DruckeMahnung ();
      }
    }
}

```

### Prog. A-9 Die Klasse Ausleiher

```

import KEB.EinAusRahmen;
import java.util.Calendar;

class Student extends Ausleiher
{ int MatNr; boolean KannExmatrikuliertWerden = true;

  String Id ()
  { return "Typ Student, Nr: " + key;
  }

  Student (EinAusRahmen einaus, String N, String V)
  { super (einaus, N, V); Leihfrist = 60;
  }

  void SperreExmatrikulation ()
  { KannExmatrikuliertWerden = false;
    EA.zeigeAus ("Bitte vor der Exmatrikulation die Gebühren begleichen!");
  }

  void MahnungBearbeiten (Calendar heute)
  { super.MahnungBearbeiten (heute); ← Benutzung der Methode in der
    if (Gebuehr > 0)                               Oberklasse
    { SperreExmatrikulation ();
      EA.zeigeAus ("Bitte die aufgelaufene Gebühr von " + Gebuehr +
        " einzahlen!");
    }
  }

  void BerechneGebuehr (Calendar heute)
  { // Pro ueberzogenem Tag einen Euro, aber kein Porto.
    // Berechne die Anzahl der Tage, die ueberzogen sind.
    Buch laufendesb = dieAusgeliehenenBuecher.dasErsteElement();

    while (laufendesb != null)
    { if (heute.after (laufendesb.RueckgabeDatum))
      { int jahr1 = heute.get (Calendar.YEAR);
        int jahr0 = laufendesb.RueckgabeDatum.get (Calendar.YEAR);
        int tag1 = heute.get (Calendar.DAY_OF_YEAR);
        int tag0 = laufendesb.RueckgabeDatum.get (Calendar.DAY_OF_YEAR);
        int diff = (jahr1 - jahr0) * 360 + (tag1 - tag0);
        EA.zeigeAus (diff + " Tage ueberzogen für Ausleiher " + Id () +
          " und Buch " + laufendesb.Id ());
        Gebuehr = Gebuehr + diff;
      }
      laufendesb = dieAusgeliehenenBuecher.dasNaechsteElement ();
    }
  }
}

```

### Prog. A-10 Die Klasse Student

```

import KEB.*;
import java.util.Calendar;

class Angestellter extends Ausleiher
{ int Gehalt;

String Id ()
{ return "Typ Angestellter, Nr: " + key;
}

Angestellter (EinAusRahmen einaus, String N, String V)
{ super (einaus, N, V); Leihfrist = 120;
}

String Stellung;
void MahnungBearbeiten (Calendar heute)
{ super.MahnungBearbeiten (heute); ← Benutzung der Methode in der
  if (Gebuehr > 0) BucheGebuehrab (); ← Oberklasse
}

void BucheGebuehrab ()
{ EA.zeigeAus("Gebühr in Höhe von " + Gebuehr + " Euro wird abgebucht.\n");
  Gehalt = Gehalt - Gebuehr; Gebuehr = 0;
}

void BerechneGebuehr (Calendar heute)
{ // Pro ueberzogenem Tag einen Euro, aber kein Porto.
  // Berechne die Anzahl der Tage, die ueberzogen sind.
  Buch laufendesb = dieAusgeliehenenBuecher.dasErsteElement ();

  while (laufendesb != null)
  { if (heute.after (laufendesb.RueckgabeDatum))
    { int jahr1 = heute.get (Calendar.YEAR);
      int jahr0 = laufendesb.RueckgabeDatum.get (Calendar.YEAR);
      int tag1 = heute.get (Calendar.DAY_OF_YEAR);
      int tag0 = laufendesb.RueckgabeDatum.get (Calendar.DAY_OF_YEAR);
      int diff = (jahr1 - jahr0) * 360 + (tag1 - tag0);
      EA.zeigeAus (diff + " Tage ueberzogen für Ausleiher " + Id () +
        " und Buch " + laufendesb.Id ());
      Gebuehr = Gebuehr + diff;
    }
    laufendesb = dieAusgeliehenenBuecher.dasNaechsteElement ();
  }
}
}

```

wird indirekt durch `MahnungBearbeiten` in der Oberklasse aufgerufen.

### Prog. A-11 Die Klasse Angestellter

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

```

import KEB.*;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.text.SimpleDateFormat;
import java.util.Date;

class BibliotheksDBsys
{
    SchluesselListe dieBibliotheksDB; // die Liste aller Buecher, Ausleiher
                                    // und Autoren.

    EinAusRahmen EA;
    GregorianCalendar heute = new GregorianCalendar ();
    GregorianCalendar demnaechst;
    SimpleDateFormat dateformat = new SimpleDateFormat ("dd.MM.yyyy");

    BibliotheksDBsys (EinAusRahmen einaus)
    { EA = einaus;   dieBibliotheksDB = new SchluesselListe (einaus);
    }

    void einfuegen (Schluessel k)
    { dieBibliotheksDB.einfuegen (k);
    }

    void neuerAusleiher(String N, String V) // Ohne
    { Ausleiher derAusleiher = new Ausleiher (EA, N, V); // Plausibili-
      dieBibliotheksDB.einfuegen (derAusleiher); // tatspruefung.
    }

    void neuerAngestellter (String N, String V) // Ohne
    { Angestellter derAusleiher = new Angestellter (EA, N, V); // Plausibili-
      dieBibliotheksDB.einfuegen (derAusleiher); // tatspruefung.
    }

    void neuerStudent (String N, String V) // Ohne
    { Student derAusleiher = new Student (EA, N, V); // Plausibili-
      dieBibliotheksDB.einfuegen (derAusleiher); // tatspruefung.
    }

    void DruckeAlle ()
    { dieBibliotheksDB.DruckeAlle ();
    }

    void DruckeAlleInfos ()
    { dieBibliotheksDB.DruckeAlleInfos ();
    }

    void DruckeAlleMahnungen ()
    { Schluessel dasElement = dieBibliotheksDB.dasErsteElement ();
      while (dasElement != null)
      { if (dasElement instanceof Ausleiher)
        ((Ausleiher) dasElement).MahnungBearbeiten (heute);
        dasElement = dieBibliotheksDB.dasNaechsteElement ();
      }
    }
}

```

Die weiteren Methoden sind in Programm A-13 bis A-16 dargestellt.

### **Prog. A-12** Die Klasse BibliotheksDBsys

```

void Buchausleihen (int Buchnummer, int Ausleihernummer)
{ // Finde heraus, ob das gewünschte Buch vorhanden und ausleihbar sowie
  // der angegebene Ausleiher existent ist. Wenn ja, trage die entsprechen-
  // den Informationen ein.
  // Prüfe zuerst Plausibilität der Buch-Bezeichnung:
  Schlüssel dasElement = dieBibliotheksDB.finde (Buchnummer);
  Buch dasBuch = null;
  if (dasElement == null)
    EA.zeigeAus ("Das Buch mit Nummer " + Buchnummer +
      " ist leider nicht vorhanden.");
  else if (dasElement instanceof Buch)
  { dasBuch = (Buch) dasElement;
    if (dasBuch.derAktuelleAusleiher != null)
    { dasBuch = null;
      EA.zeigeAus ("Leider ist das Buch " + Buchnummer +
        " bereits ausgeliehen.");
    }
  }
  else
  { dasBuch = null;
    EA.zeigeAus ("Das Buch mit Nummer " + Buchnummer +
      " ist leider nicht vorhanden.");
  }
  if (dasBuch != null)
  { // Nun kann nach dem Ausleiher gesucht werden.
    Ausleiher derAusleiher = null;
    dasElement = dieBibliotheksDB.finde (Ausleihernummer);
    if (dasElement == null)
      EA.zeigeAus ("Ein Ausleiher mit der Nummer " + Ausleihernummer +
        " kann nicht gefunden werden.");
    else if (dasElement instanceof Ausleiher)
    { derAusleiher = (Ausleiher) dasElement;
      if (derAusleiher.Gebuehr != 0)
      { EA.zeigeAus ("Der Ausleiher "+ Ausleihernummer +
        " muss erst seine Gebühren in Höhe von " +
        ((Ausleiher) dasElement).Gebuehr + " begleichen.");
        derAusleiher = null;
      }
    }
  }
  else
    EA.zeigeAus ("Ein Ausleiher mit der Nummer " + Ausleihernummer +
      " kann nicht gefunden werden.");
  if (derAusleiher != null)
  { // Nun sind alle zusammen: Buch und Ausleiher.
    derAusleiher.dieAusgeliehenenBuecher.einfuegen (dasBuch);
    dasBuch.derAktuelleAusleiher = derAusleiher;
    dasBuch.RueckgabeDatum = (GregorianCalendar) heute.clone ();
    EA.zeigeAus ("Das Buch wird heute am " +
      dateFormat.format (dasBuch.RueckgabeDatum.getTime ()) +
      " ausgeliehen");
    dasBuch.RueckgabeDatum.add (Calendar.DATE, derAusleiher.Leihfrist);
    EA.zeigeAus (" und muss am " +
      dateFormat.format (dasBuch.RueckgabeDatum.getTime ()) +
      " zurückgegeben werden.");
  }
}
}
}

```

### Prog. A-13 Methode Buchausleihen in der Klasse BibliotheksDBsys

```
void SetzeDatum(String neuesdatum)
{ // Setze neues Datum und berechne die Gebühren.
  Date neuesheute = new Date ();
  boolean schluss = false;
  try { neuesheute = dateformat.parse (neuesdatum);
    }
  catch (java.text.ParseException pe)
  { EA.zeigeAus ("ParseException: falsches Datum " + pe);  schluss = true;
    }
  if ( ! schluss)
  { heute.setTime (neuesheute);
    EA.zeigeAus ("Es ist heute der " +
      dateformat.format (heute.getTime ()) +
      ". Die ausgeliehenen Bücher werden jetzt überprueft.");
    DruckeAlleMahnungen ();
  }
}
```

**Prog. A-14** Methode SetzeDatum in der Klasse BibliotheksDBsys

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```
void BuchZurueckGeben (int Buchnummer)
{ // Finde heraus, ob das Buch existiert und ausgeliehen ist.
  Schluessel dasElement = dieBibliotheksDB.finde (Buchnummer);
  Buch dasBuch = null;
  if (dasElement == null)
    EA.zeigeAus ("Ein Buch mit der Nummer " + Buchnummer +
                " existiert nicht.");
  else if (dasElement instanceof Buch)
    dasBuch = (Buch) dasElement;
  else
    EA.zeigeAus ("Ein Buch mit der Nummer " + Buchnummer +
                " existiert nicht.");
  // Falls dasBuch vorhanden ist, wird es aus der Liste der
  // ausgeliehenen Bücher gestrichen.
  if (dasBuch != null)
  { Ausleiher derAusleiher = dasBuch.derAktuelleAusleiher;
    derAusleiher.dieAusgeliehenenBuecher.entferne (dasBuch);
    dasBuch.derAktuelleAusleiher = null;
  }
}
```

**Prog. A-15** Methode BuchZurueckGeben in der Klasse BibliotheksDBsys

*Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000*

```

void trageNeuesBuchEin (String Titel, String Vorname_, String Nachname_)
{ // Suche den Autoren. Falls in der Datenbank noch unbekannt,
  // den Autoren in die Datenbank einfügen.
  Schluessel dasElement = dieBibliotheksDB.dasErsteElement ();
  boolean fertig = false;
  Autor derAutor = null;
  Buch dasBuch;

  while ( ! fertig)
  { if (dasElement == null)
    { fertig = true;      // Autor in die Datenbank einfügen.
      derAutor = new Autor (EA, Nachname, Vorname_);
      dieBibliotheksDB.einfuegen (derAutor);
    }
    else if (dasElement instanceof Autor)
    { if ( ( ( (Autor) dasElement).Vorname.compareTo (Vorname_)) == 0 &&
          ( ( (Autor) dasElement).Name.compareTo (Nachname_ ) ) == 0      )
      { derAutor = (Autor) dasElement;   fertig = true;
        }
      else
        dasElement = dieBibliotheksDB.dasNaechsteElement ();
    }
    else
      dasElement = dieBibliotheksDB.dasNaechsteElement ();
  }

  // Nun ist ein Autor-Objekt gefunden bzw. neu erzeugt worden. Das Buch
  // kann jetzt eingetragen werden. Hier wird nicht geprüft, ob das Buch
  // in der Datenbank schon vorhanden ist, da eine Bibliothek mehrere
  // Exemplare eines Buches anschaffen kann. Dies gilt sogar für die
  // stets unterfinanzierten Uni-Bibliotheken.

  dasBuch = new Buch (EA, Titel, derAutor);
  dieBibliotheksDB.einfuegen (dasBuch);
  derAutor.neuesBuch (dasBuch);
}

```

**Prog. A-16** Methode `trageNeuesBuchEin` in der Klasse `BibliotheksDBsys`

*Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000*

```

import KEB.*;

public class Bibliothek extends EinAusRahmen
{
    BibliotheksDBsys dieBibliothek = new BibliotheksDBsys (this);

    public static void main (String args [])
    { System.out.println ("Hier ist das Bibliothekssystem.");
      Anfang (new Bibliothek (), "Bibliothek", 15, 1);
    }

    command [] thecommands =
    {
        new command ("Ende")
        { void go (String e)
          { Ende ();
          }
        },

        new command ("DruckeDB")
        { void go (String e)
          { dieBibliothek.DruckeAlle ();
          }
        },

        new command ("NeuerAusleiher")
        { void go (String e)
          { if (WortAnz (e, 1) < 3)
              zeigeAus ("Falsche Zahl von Parametern für Kommando" +
                        commandstring);
            else
              dieBibliothek.neuerAusleiher (Wort (e, 1, 2), Wort (e, 1, 3));
          }
        },

        new command ("trageNeuesBuchEin")
        { void go (String e)
          { if (WortAnz (e, 1) < 4)
              zeigeAus ("Falsche Zahl von Parametern für Kommando:" +
                        commandstring +
                        "\nbenötigt: trageNeuesBuchein Titel Vorname Nachname");
            else
              dieBibliothek.trageNeuesBuchEin (Wort (e, 1, 2),
                                                Wort (e, 1, 3), Wort (e, 1, 4));
          }
        },

        new command ("neuerStudent")
        { void go (String e)
          { if (WortAnz (e, 1) < 3)
              zeigeAus ("Falsche Zahl von Parametern für Kommando:" +
                        commandstring +
                        "\nbenötigt: neuerStudent Vorname Nachname");
            else
              dieBibliothek.neuerStudent (Wort (e, 1, 2), Wort (e, 1, 3));
          }
        },
    },
}

```

Erzeugen eines Objektes BibliotheksDBsys und „Durchreichen“ des Objektes vom Typ EinAusRahmen

Darstellung der Kommandos

### Prog. A-17 Erster Teil der Klasse Bibliothek

```
new command ("neuerAngestellter")
{
  void go (String e)
  {
    if (WortAnz (e, 1) < 3)
      zeigeAus ("Falsche Zahl von Parametern für Kommando:" +
               commandstring +
               "\nbenoetigt: neuerAngestellter Vorname Nachname");
    else
      dieBibliothek.neuerAngestellter (Wort (e, 1, 2), Wort (e, 1, 3));
  }
},

new command ("DruckeAlleInfos")
{
  void go (String e)
  {
    dieBibliothek.DruckeAlleInfos ();
  }
},

new command ("Buchausleihen")
{
  void go (String e)
  {
    if (WortAnz (e, 1) < 3)
      zeigeAus ("Falsche Zahl von Parametern für Kommando:" +
               commandstring +
               "\nbenoetigt: Buchausleihen Buchnummer Ausleihernummer");
    else
      dieBibliothek.Buchausleihen (liesInt (e, 1, 2), liesInt (e, 1, 3));
  }
},

new command ("SetzeDatum")
{
  void go (String e)
  {
    if (WortAnz (e, 1) < 2)
      zeigeAus ("Falsche Zahl von Parametern für Kommando:" +
               commandstring + "\nbenoetigt: SetzeDatum Datum");
    else
      dieBibliothek.SetzeDatum (Wort (e, 1, 2));
  }
},

new command ("BuchZurueckGeben")
{
  void go (String e)
  {
    if (WortAnz (e, 1) < 2)
      zeigeAus ("Falsche Zahl von Parametern fuer Kommando:" +
               commandstring +
               "\nbenoetigt: BuchZurueckGeben Buchnummer");
    else
      dieBibliothek.BuchZurueckGeben (liesInt (e, 1, 2));
  }
},
};
```

### **Prog. A-18** Zweiter Teil der Klasse Bibliothek

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
public void Hilfe ()
{ zeigeAus ("Dies ist das Bibliotheksprogramm .... die Kommandos lauten:");
  for (int i = 0; i < thecommands.length; i++)
    zeigeAus (thecommands[i].commandstring);
}

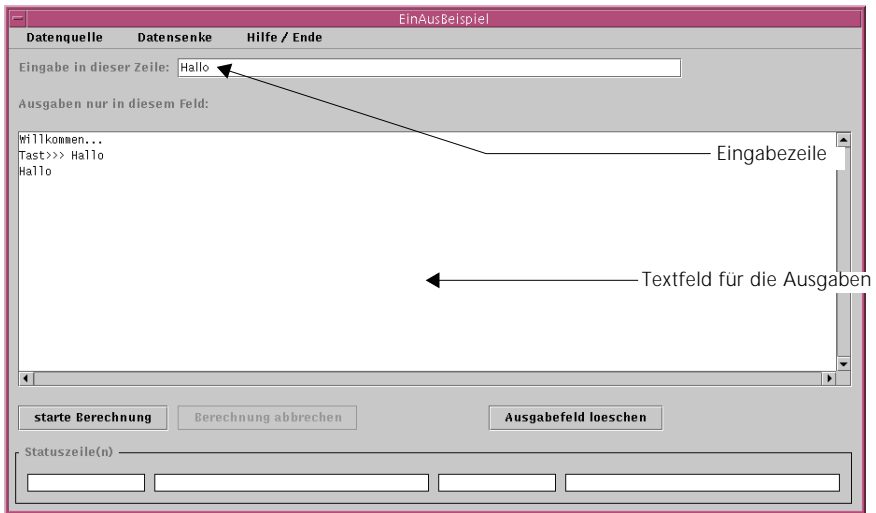
public void Antwort (String Eingabe)
{ int i; boolean fertig;
  if (SatzAnz (Eingabe) >= 1)
  { if (WortAnz (Eingabe, 1) >= 1)
    { i = 0; fertig = false;
      while ( ! fertig )
      { if (i >= thecommands.length)
        fertig = true;
        else if (thecommands [i].commandstring.compareTo
                  (Wort (Eingabe, 1, 1)) == 0)
          { thecommands [i].go (Eingabe); fertig = true;
            }
          else
            i++;
        };
      if (i >= thecommands.length)
        zeigeAus ("Kein Kommando: " + Wort (Eingabe, 1, 1));
    }
    else
      zeigeAus ("Nicht genügend Eingabezeichen? " + Eingabe);
  }
  else
    zeigeAus ("Leere Eingabe? " + Eingabe);
}

public void Ende()
{ zeigeAus ("Schluss jetzt ..."); super.Ende ();
}
}
```

**Prog. A-19** Dritter Teil der Klasse Bibliothek

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

# B



**Abb. B-1** Dialogfenster eines Objektes der Klasse `EinAusRahmen`

Lehrbuch der Programmierung mit Java, Echte Goedicke, Heidelberg, © dpunkt 2000

```
import KEB.*;

public class EinAusBeispiel extends EinAusRahmen
{
    public static void main (String [] unbenutzt)
    {
        Anfang (new EinAusBeispiel (), "EinAusBeispiel", 15, 1);
    }

    public void Hilfe ()
    {
        zeigeAus (" Hier kommt die Hilfe ..." +
            "\n Jeder Text wird direkt wieder ausgegeben.");
    }

    public void Antwort (String Eingabe)
    {
        // Bearbeite den Eingabe-String, führe damit eine Berechnung durch
        // und gib die Ergebnisse aus. In diesem einfachen Beispiel wird die
        // Eingabe unverändert wieder ausgegeben.
        zeigeAus (Eingabe);
    }

    public void Ende ()
    {
        zeigeAus ("Schluss jetzt ...");    super.Ende ();
    }
}
```

**Prog. B-1** Benutzung der Klasse EinAusRahmen durch das Programm EinAusBeispiel

Lehrbuch der Programmierung mit Java, Echtele Goedicke, Heidelberg, © dpunkt 2000