Freie Universität Berlin

Java-Programmierkurs

Anweisungen zur Ablaufsteuerung

WS 2012/2013

Prof. Dr. Margarita Esponda-Argüero

Anweisungen zur Ablaufsteuerung

if

switch

while

do-while

for

Anweisungen zur Ablaufsteuerung

```
if ( Ausdruck )
   { Anweisungen }
else
   { Anweisungen }
```

```
switch ( Ausdruck ) {
    case Konstante1: Anweisungen break;
    case Konstante2: Anweisungen break;
        . . . . . usw.
    default :  Anweisungen
}
```

```
while ( Bedingung )
{
        Anweisungen
 }
```

```
do  {
            Anweisungen
}
while ( Bedingung ) ;
```

Semikolon

```
for ( Initialisierung ; Bedingung ; Inkrement )
    {   Anweisungen   }
```

if-else-Anweisung

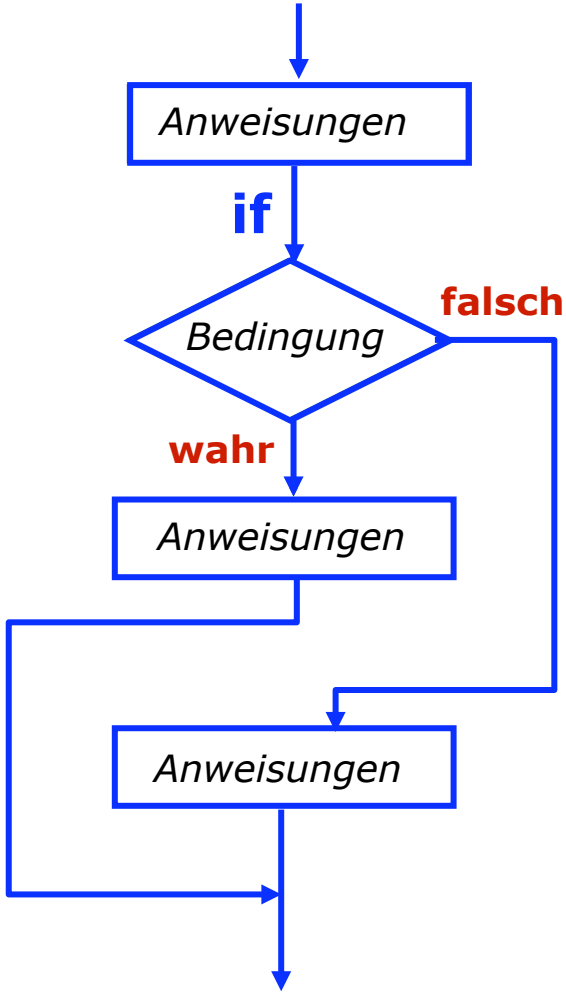
Wahrheitswert → boolean

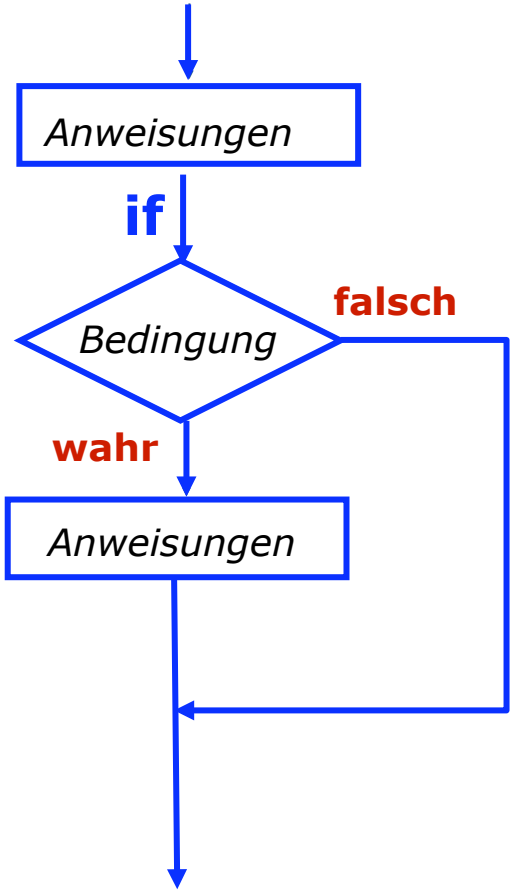
Runde Klammern

```
if (Ausdruck)
   { Anweisungen }
else
   { Anweisungen }
```

if-else-Anweisung

Kontrollfluss





if-else-Anweisung

```
if( count == max )
{
   count = 0;
}
```

```
if( count == max )
   count = 0;
```

```
if (a < b)
   max = b;
else
   max = a;
```

Am Ende der **if**-Anweisung ist der Inhalt der Variablen max gleich der größten Zahl zwischen a und b.

if-else-Anweisung

```
if( punkte >= 90 )
     {  note = 1; }
else { if ( punkte >= 80 )
          { note = 2; }
       else  { if ( punkte >= 70 )
                  { note = 3; }
               else  { if ( punkte >= 60 )
                          { note = 4; }
                       else   { note = 5;}
                     }
             }
     }
```

if-else-Anweisung

```
...
if( punkte >= 90 )
    note = 1;
else if ( punkte >= 80 )
    note = 2;
else if ( punkte >= 70 )
    note = 3;
else if ( punkte >= 60 )
    note = 4;
else note = 5;
...
```

Pathologisches Beispiel

```
count = 13;

if (count == 5);
  {   count ++;
      count += 5;
  }
count--;
```

count? **18**

Kein Semikolon nach der Bedingung!

switch-Anweisung

```
switch (Ausdruck) {
    case Konstante1: Anweisungen break;
    case Konstante2: Anweisungen break;
        . . . . . usw.
    default :  Anweisungen
}
```

switch-Anweisung

short
int
long
char

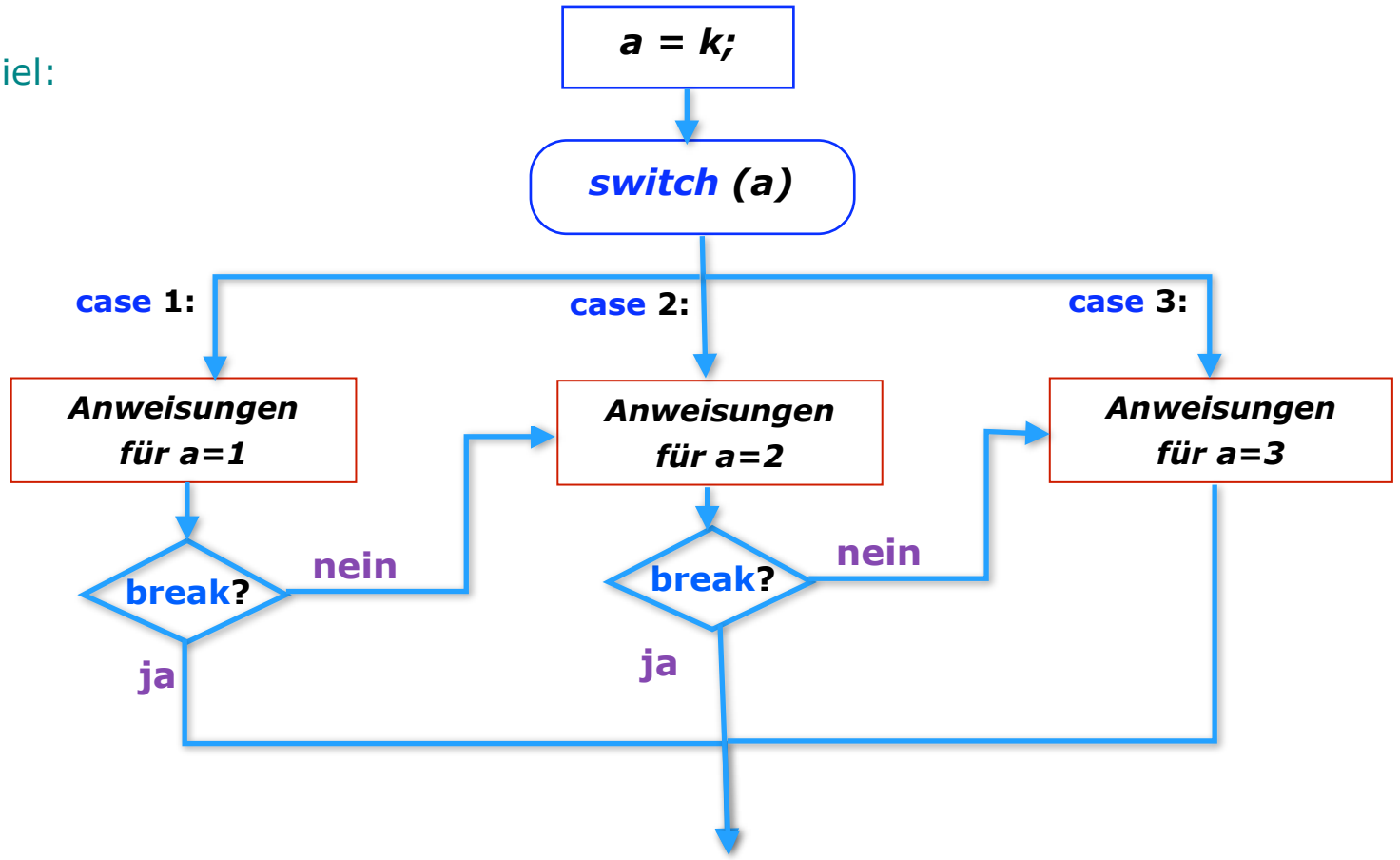
Die switch-Anweisung erlaubt die Verzweigung in Abhängigkeit vom Wert eines *ganzzahligen* Ausdrucks, der mit einer Reihe von Konstanten verglichen wird.

```
switch ( Ausdruck ) {
    case Konstante1 : {
                          Anweisung1
                          Anweisung2
                           . . .
                      }
                      break;
case Konstante2 :     Anweisungen
                      break;
            . . . . . usw.
    default:    Anweisungen
}
```

switch-Anweisung

Kontrollfluss

Beispiel:

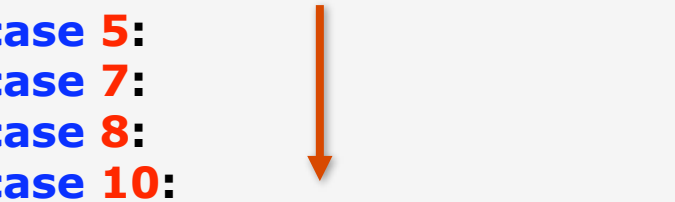


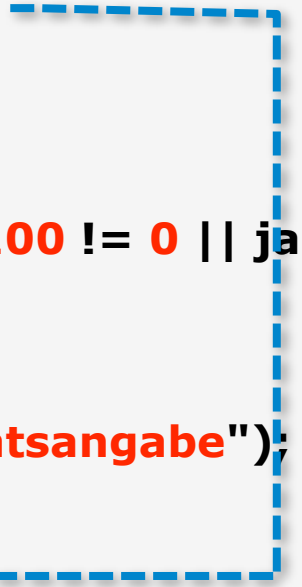
switch-Anweisung

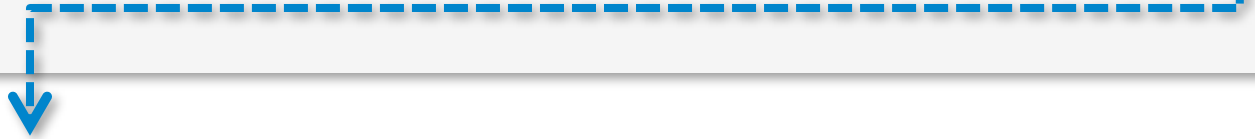
```
int zahl; String word;
...
switch( zahl ) {
   case 1:  word = "eins";     break;
   case 2:  word = "zwei";     break;
   case 3:  word = "drei";     break;
   case 4:  word = "vier";     break;
   case 5:  word = "fünf";     break;
   case 6:  word = "sechs";    break;
   case 7:  word = "sieben";   break;
   case 8:  word = "acht";     break;
   case 9:  word = "neun";     break;
   case 0:  word = "null";     break;
   default: word = "error";
}
```

switch-Anweisung

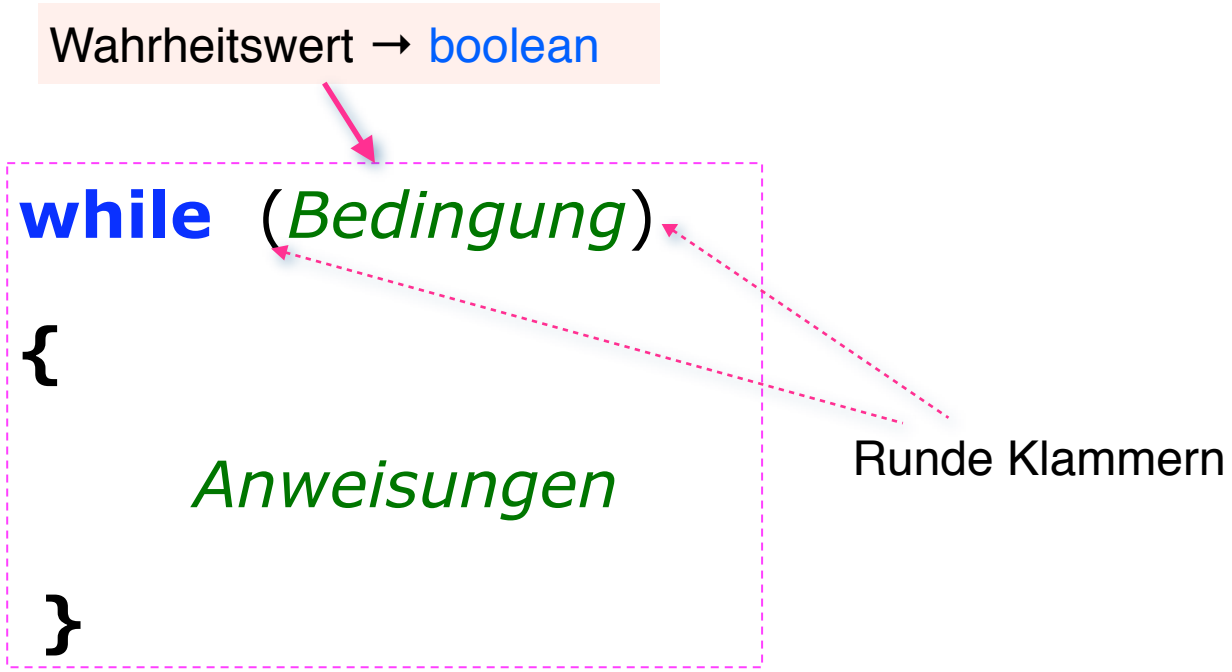
```
int monat, jahr, tage;
...
switch( monat ) {
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12:  tage = 31; break;
  case 4:
  case 6:
  case 9:
  case 11:  tage = 30; break;
  case 2:   if ( jahr % 4 = = 0 && ( jahr % 100 != 0 || jahr % 400 = = 0 ) )
                 tage = 29;
            else
                 tage = 28; break;
  default: System.out.println("falsche Monatsangabe");
}
```





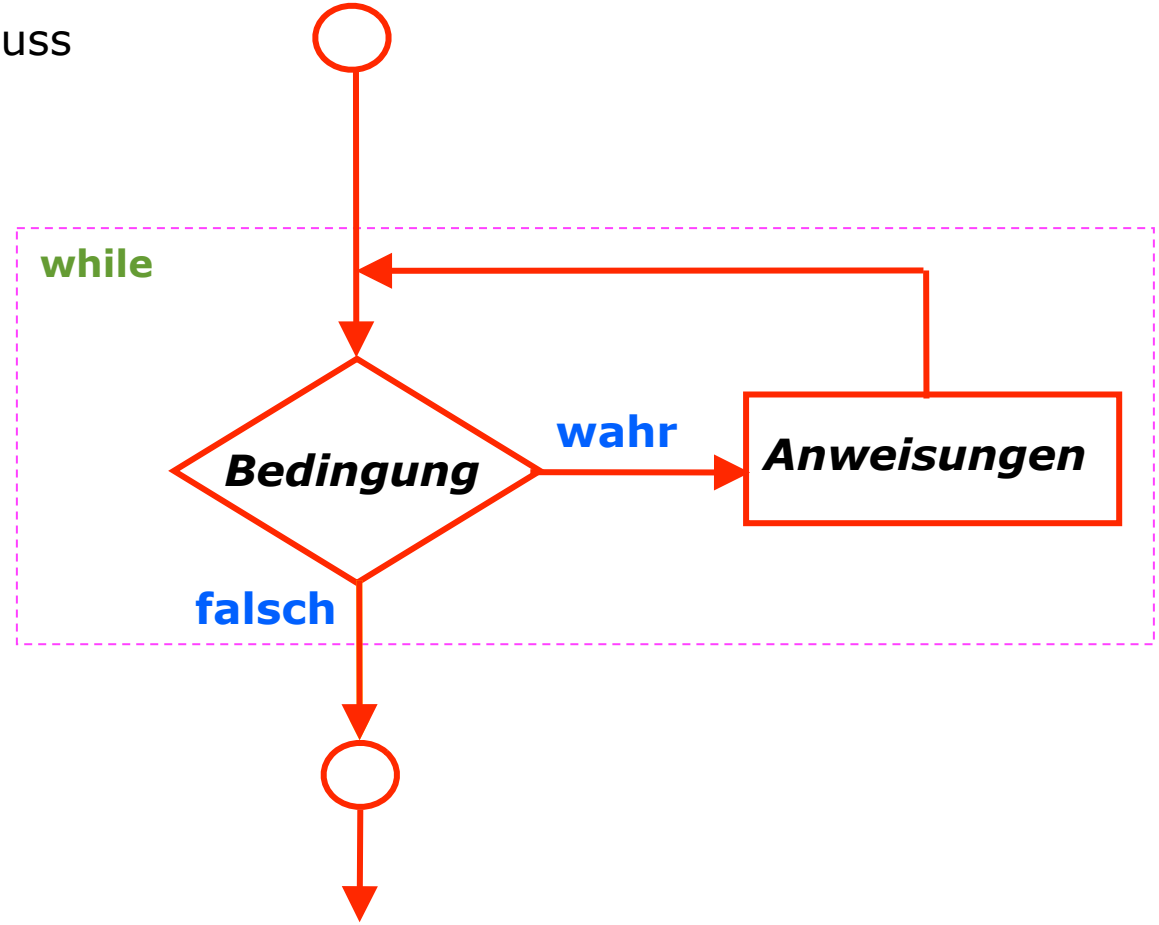


while-Schleife



while-Anweisung

Kontrollfluss



while-Anweisung

```
public class Gluecksspieler {

   public static void main( String[] args ){
         int bargeld = Integer.parseInt(args[0]);

         while ( bargeld > 0 ) {
                  if  (( (int)(Math.random()*1000)%2)==0)
                           ++bargeld;
                  else
                           --bargeld;
                  System.out.println( bargeld );
         }
         System.out.println( "You're a big loser! \n" );
    }
}
```

Die Math-Klasse

```
static double  sin (double a)
static double  cos (double a)
static double  tan (double a)
```

```
static double  log (double a)
static double  pow (double a, double b)
static double  exp (double a)
static double  sqrt (double a)
```

```
static double  random ()
static long    round (double a)
static double  ceil (double a)
static double  floor (double a)
```

Anwendungsbeispiel:

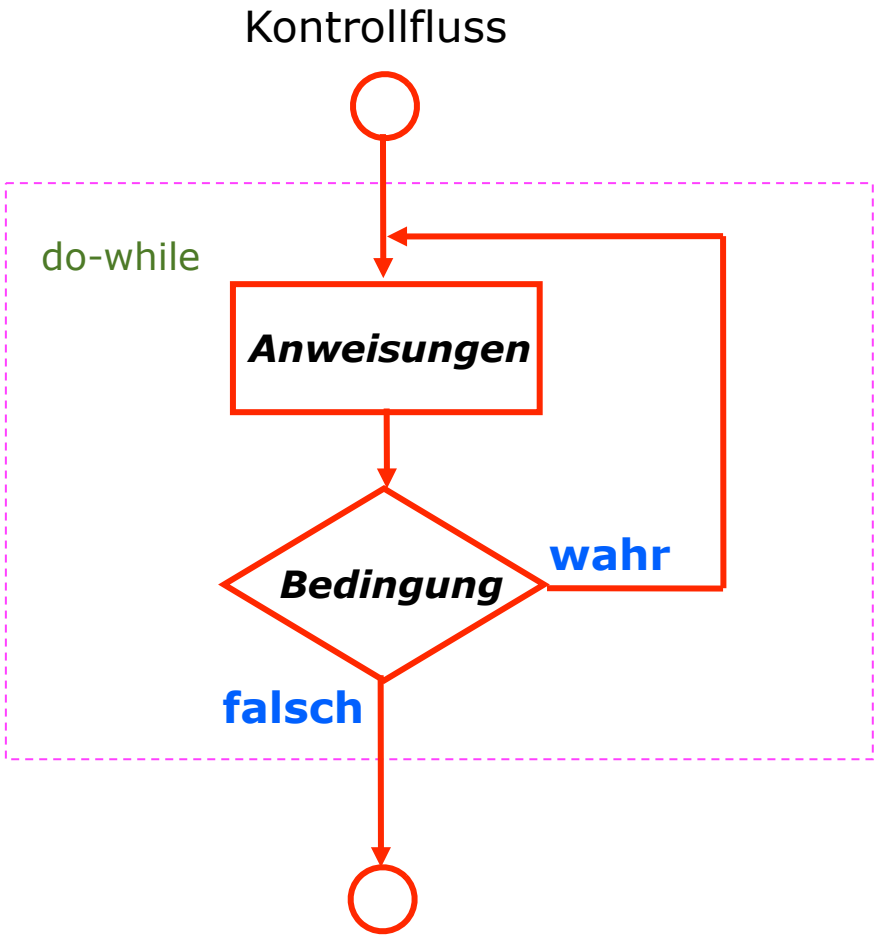
```
double x,y;
  . . . ..
double zufallszahl = Math.random();
   . . .  .
x = Math.sin (y);
   . . . . .
```

do-while-Anweisung

```
do {
      Anweisungen
   }
while (Bedingung) ;
```

Bei der do-while-Anweisung wird zunächst der Schleifen-Rumpf ausgeführt und *anschließend* die Bedingung überprüft.

do-while-Anweisung



Die Keyboard-Klasse

```
public static int readInt();
public static double readDouble();
public static boolean readBool();
public static float readFloat();
public static short readShort();
public static long readLong();
public static byte readByte();
public static String readText();
public static BigInteger readBigInteger();
```

Ohne Gewähr!

Einfache Klasse ohne Fehlerbehandlung

Die Methoden stürzen ab, wenn die Eingabe mit dem Datentyp der Methoden nicht übereinstimmt.

Die Keyboard-Klasse

```
public class TestKeyboard {

      public static void main(String[] args ){
            int n, factorial = 1;
            int counter = 0;
            do{
                  System.out.print("n = ");
                  n = Keyboard.readInt();
                  counter = n;
                  while (counter>0){
                        factorial = factorial * counter;
                        counter--;
                  }
               System.out.println(n+"! = "+factorial);
               factorial = 1;
            } while (n>=0);
      }
}// end of class TestKeyboard
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
14! = 1278945280
15! = 2004310016
16! = 2004189184
17! = -288522240
```

Überlauf!

for-Anweisung

```
for ( Initialisierung ; Bedingung ; Inkrement )
{
        Anweisungen
}
```

for-Schleifen verwenden wir, wenn die Einschränkungen der Schleife (*Initialisierung, Bedingung und Inkrementierung*) bekannt sind.

1. Die ***Initialisierung*** wird einmal zu Beginn ausgeführt.
2. Der Schleifenrumpf wird ausgeführt, solange die ***Bedingung*** erfüllt ist.
3. Nach jedem Durchlauf des Rumpfes wird die Anweisung im ***Inkrement*** einmal ausgeführt und die ***Bedingung*** erneut geprüft.

for-Anweisung

```
...
for (int i=0; i<=100; i++ ) {
  System.out.println( i*i );
}
...
```

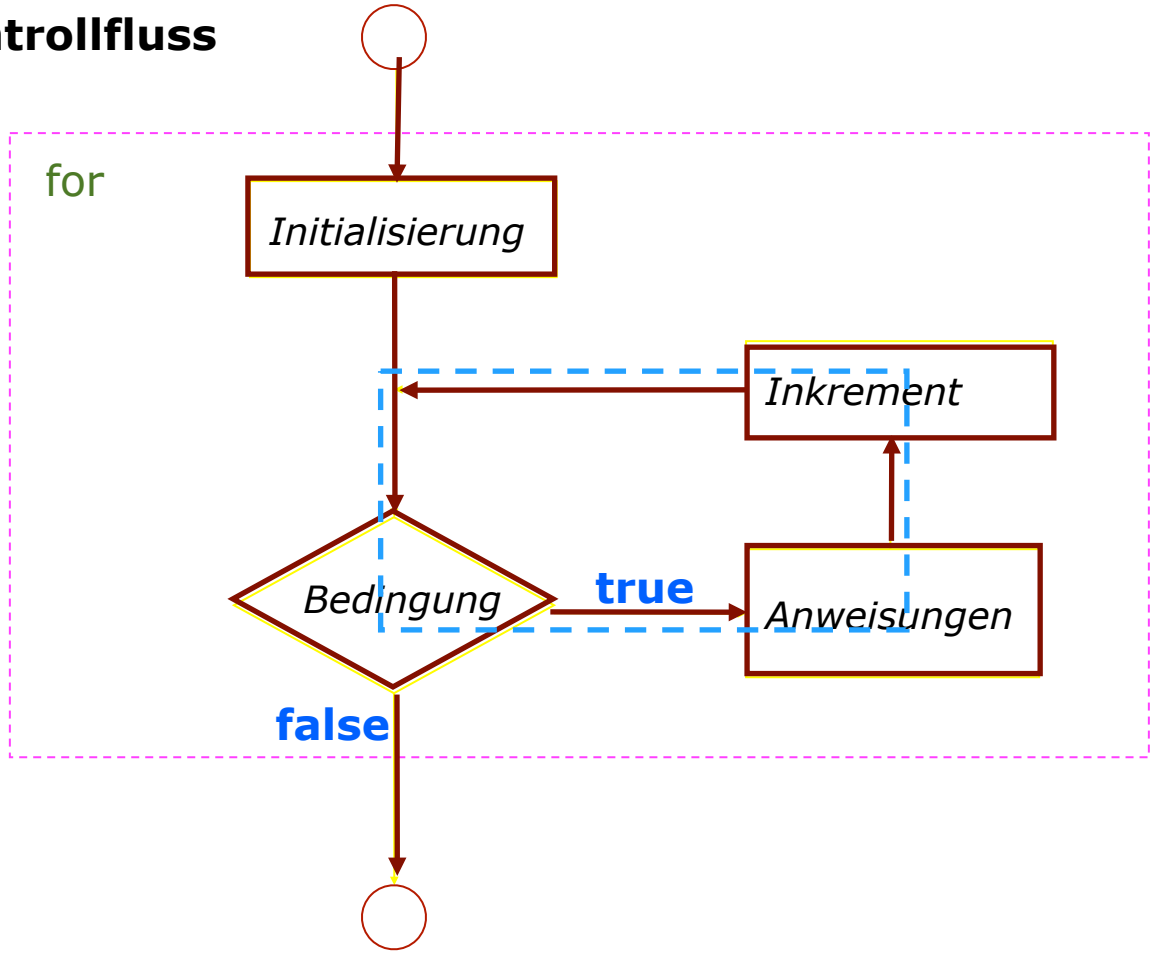
Die Initialisierung wird einmal zu Beginn ausgeführt.

Vor jedem Durchlauf des Rumpfes wird die Bedingung geprüft.

Nach jedem Durchlauf des Rumpfes wird die Anweisung im Inkrement einmal ausgeführt.

for-Anweisung

Kontrollfluss



for-Anweisung

```
...
for (int  i=0; i<=100; i++ ) {
    if ( i%7 == 0 )
        System.out.println( i );
}
...
```

Alle Zahlen zwischen 0 und 100, die **genau** durch 7 geteilt werden können, werden ausgegeben.

Ausgabe

```
 0  7  14  21  28  35  42  49  56  63  70  77  84  91  98
```

for-Anweisung

```
public  class  ForStatements  {
/*Hier berechnen wir die Fakultätsfunktion für alle Zahlen von 1 bis n.
   1!  2!  3!  4! … n!
*/
    public static void main(String[] args) {
        int n=17;
        int fac = 1;

        for  ( int index = 1;  index <= n;  index++  ) {
                fac  =  fac * index;
                System.out.println(index + "! = " + fac);
        }
    }
}
```

for-Anweisung

Freie Universität Berlin

```
... double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1/i*i;
}
double  pi = Math.sqrt(sum*6);
...
```

falsch!

2.449489742783178

$$R_n = \sum_{i=1}^{n} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \ldots + \frac{1}{n^2} \approx \frac{\pi^2}{6}$$

```
...
double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1/(i*i);
}
double  pi = Math.sqrt(sum*6);
...
```

falsch!

2.449489742783178

ArithmeticException

```
...
double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1.0/(i*i);
}
double  pi = Math.sqrt(sum*6);
...
```

falsch!

3.1414971639472147

Infinity

```
...
double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1/(1.0*i*i);
}
double  pi = Math.sqrt(sum*6);
```

richtig!

3.141583104326456

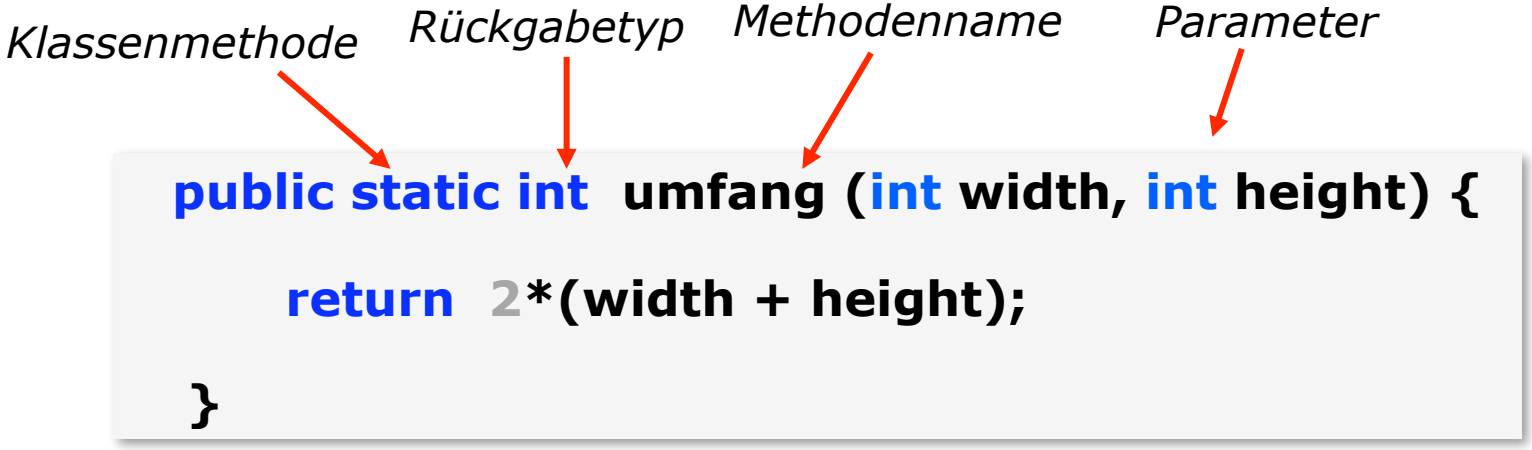
Klassenmethoden

Methoden enthalten den ablauffähigen Programmcode.

Es gibt keine **Methoden** außerhalb von Klassen.

Methoden dürfen nicht geschachtelt werden.

Beispiel:



return-Anweisung

Die return-Anweisung beendet frühzeitig die Ausführung einer Methode.

In einer Methode kann es mehrere return-Anweisungen geben.

Methoden, die als Funktionen definiert sind, d.h. ein Ergebnis liefern, müssen durch eine return-Anweisung dieses Ergebnis zurückgeben.

Wenn eine Methode kein Ergebnis zurückgibt, wird das Schlüsselwort **void** als Rückgabetyp verwendet, und eine return-Anweisung ist nicht erforderlich.

return-Anweisung

```
public static int  fakultaet (  int  zahl  ) {
    int fak = 1;
    if  ( ( zahl == 0 ) || ( zahl == 1 ) )
        return fak ;
    else {
            while ( zahl > 1 ) {
                fak = fak*zahl;
                zahl = zahl - 1;
            }
            return fak ;
        }
} // end of factorial
```

Die **return**-Anweisung beendet den Lauf der Funktion (Methode) und sorgt für die Übergabe des Ergebnisses.

Klassenmethoden

$$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

```
public class ForStatements {

        public static double pi_leibnitz( int n_max ){
                double sum = 0;
                for (int n = 0; n<=n_max; n++ ){
                        if ( n%2 == 0 )
                                sum = sum + (1.0)/(2*n+1);
                        else
                                sum = sum - (1.0)/(2*n+1);
                }
                return 4*sum;
        }
}
```

Klassenmethoden

Klassenmethoden werden als **static** deklariert und über den Klassennamen aufgerufen:

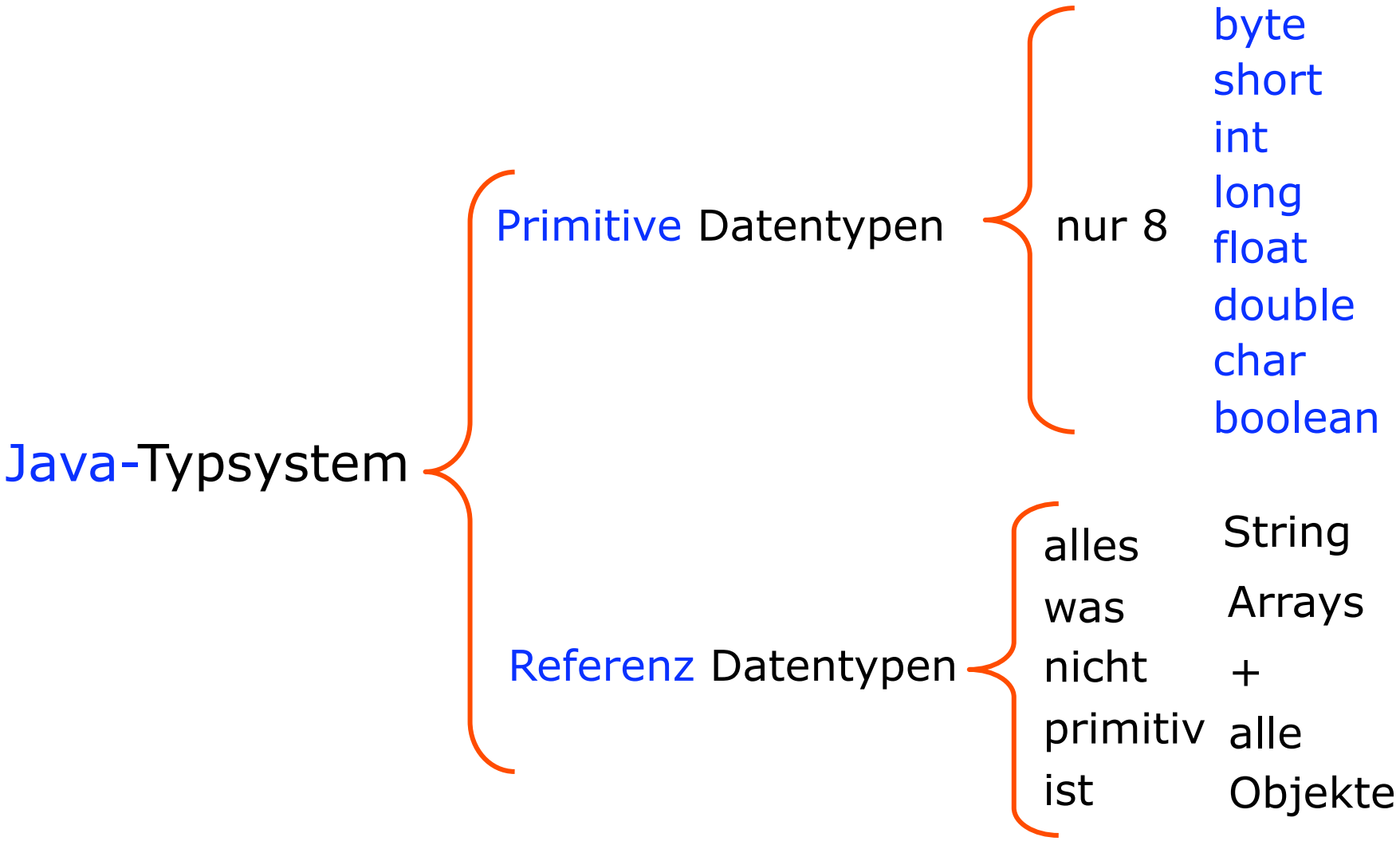
```
class FirstClass {
   static void factorial( int n ) {
      . . .
   }
}
```

innerhalb anderer Klassen →

```
...
FirstClass.factorial( 29 );
...
```

for-Anweisung

für Morgen


Java-Typsystem
Primitive Datentypen
nur 8
byte
short
int
long
float
double
char
boolean
Referenz Datentypen
alles
was
nicht
primitiv
ist
String
Arrays
+
alle
Objekte

Java-Typsystem

Java ist ***streng typisiert***, d.h. jeder Ausdruck hat einen wohldefinierten Typ.

Die Einhaltung aller durch das Typsystem definierten Regeln wird

- zuerst statisch vom **Übersetzer** überprüft.
- und dann dynamisch vom **Interpreter**.

Neue Objekttypen werden durch Klassen definiert.

Variablen solcher Typen sind ***Referenzen:***

for-Anweisung

Die Initialisierung kann mehrere Initialisierungen von Variablen beinhalten.

Nach jedem Durchlauf können mehrere Variablen verändert werden.

```
...
  for ( i=0,j=20,k=5; i<=10; i++,j--,k+=5 )
  {
        System.out.println( (i+j)*k );
  }
...
```

for-Anweisung

For-Schleifen können beliebig verschachtelt werden.

```
...
for ( int i=1; i<=10; i++ ) {
    for ( int j=1; j<=10; j++ ) {
        ...
    }
    ...
}
...
```

for-Anweisung

Jede for-Anweisung lässt sich auch als while-Anweisung formulieren.

```
...
for (int n=1, int m = 1; n<=16; n++ ){
        m = m*n;
}
...
```

```
...
int n = 1;          /* Initialisierung */
int m = 1;
while ( n <= 16 ){
    m = m*n;
    n++;            /* Schleifenzähler */
}
...
```

for-Anweisung

Jeder der drei Ausdrücke in der for-Schleife kann auch fehlen; die Semikola müssen aber trotzdem gesetzt werden.

```
for (;;) {
   ...
}
```

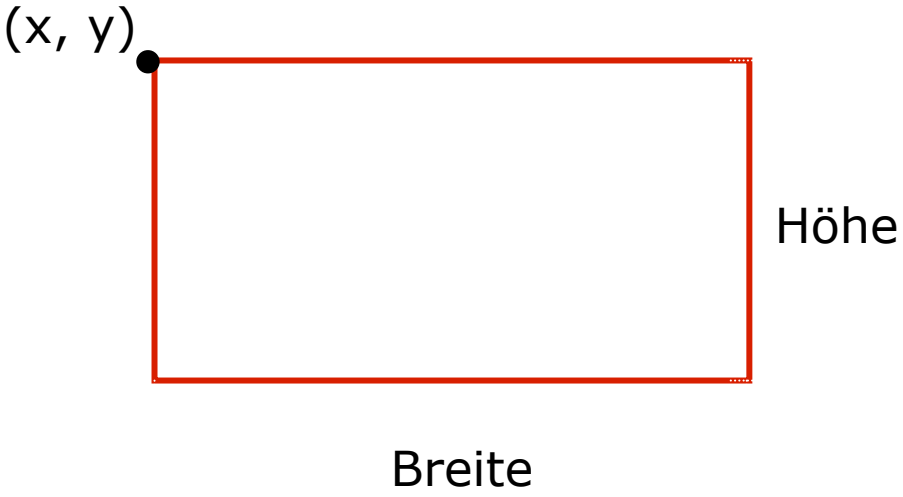
=

```
while ( true ) {
   ...
}
```

Endlosschleifen!!

Modellierung von Rechteck-Objekten

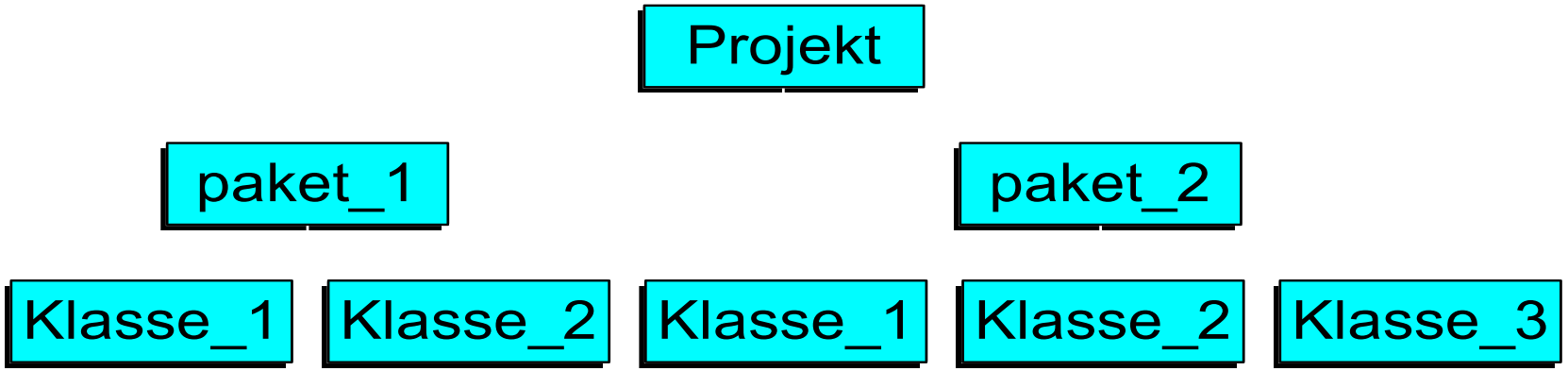
Eigenschaften



Operationen

- Fläche
- Umfang
- verkleinern
- vergrößern
- verschieben
- klonen

java-Anwendungen




```
packet   packet_1 ;
   class Klasse_1 {

      . . . . .

   }
```

Warum Objektorientierte Programmierung?

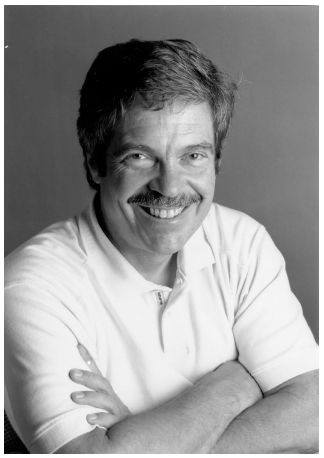
Hauptproblem bei prozeduraler Programmierung

Trennung zwischen Prozeduren (Funktionen) und Daten

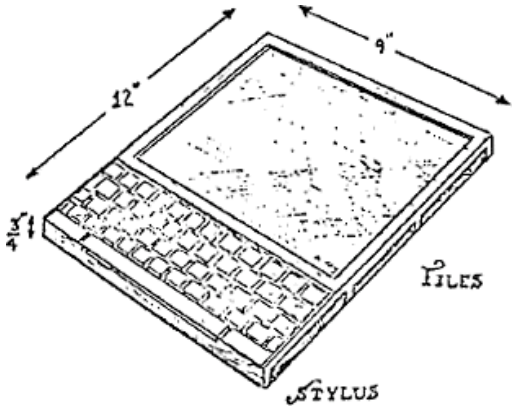
- welche Daten sollen lokal und welche global existieren?
- Probleme, einen geeigneten Zugriffsschutz auf die globalen Daten zu finden
- Probleme der realen Welt sind schwer prozedural zu simulieren
- ungeeignet für große Softwaresysteme
- schlechte Wiederverwendbarkeit der Software

Erste Objektorientierte Ideen

Alan Kay



- 1968. Xerox
- erste objektorientierte Konzepte für die Gestaltung von Benutzeroberflächen
- **Dynabook-**Konzept



- **Smalltalk-**Programmiersprache

„The best way to predict the future is to invent it."

Was ist ein Objekt?

Objekte sind verwandt mit:

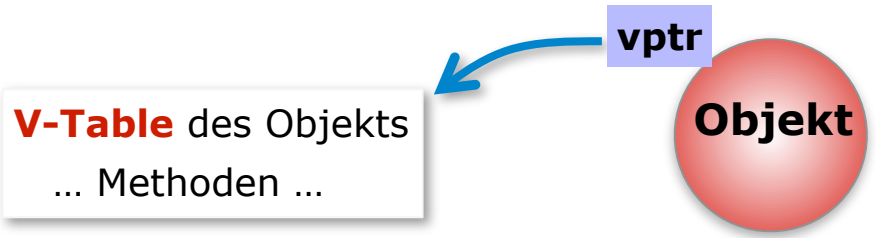
C/C++ struct-Datentyp

```
struct  Student {
    int matrikelnummer;
    int alter;
    int semester;
};
```

Pascal Records

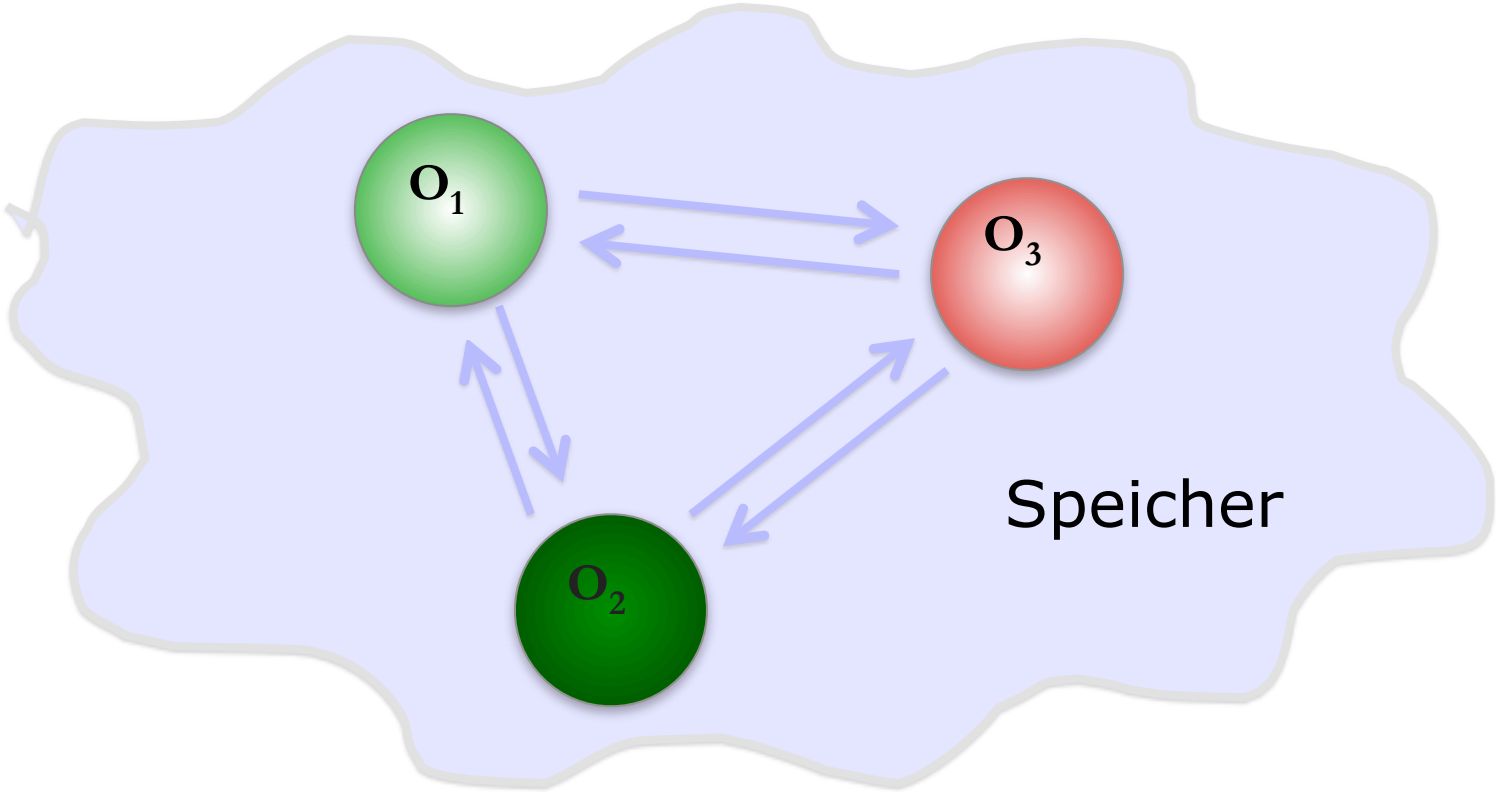
```
TYPE datum = RECORD
        tag, monat, jahr : INTEGER;
END;
```

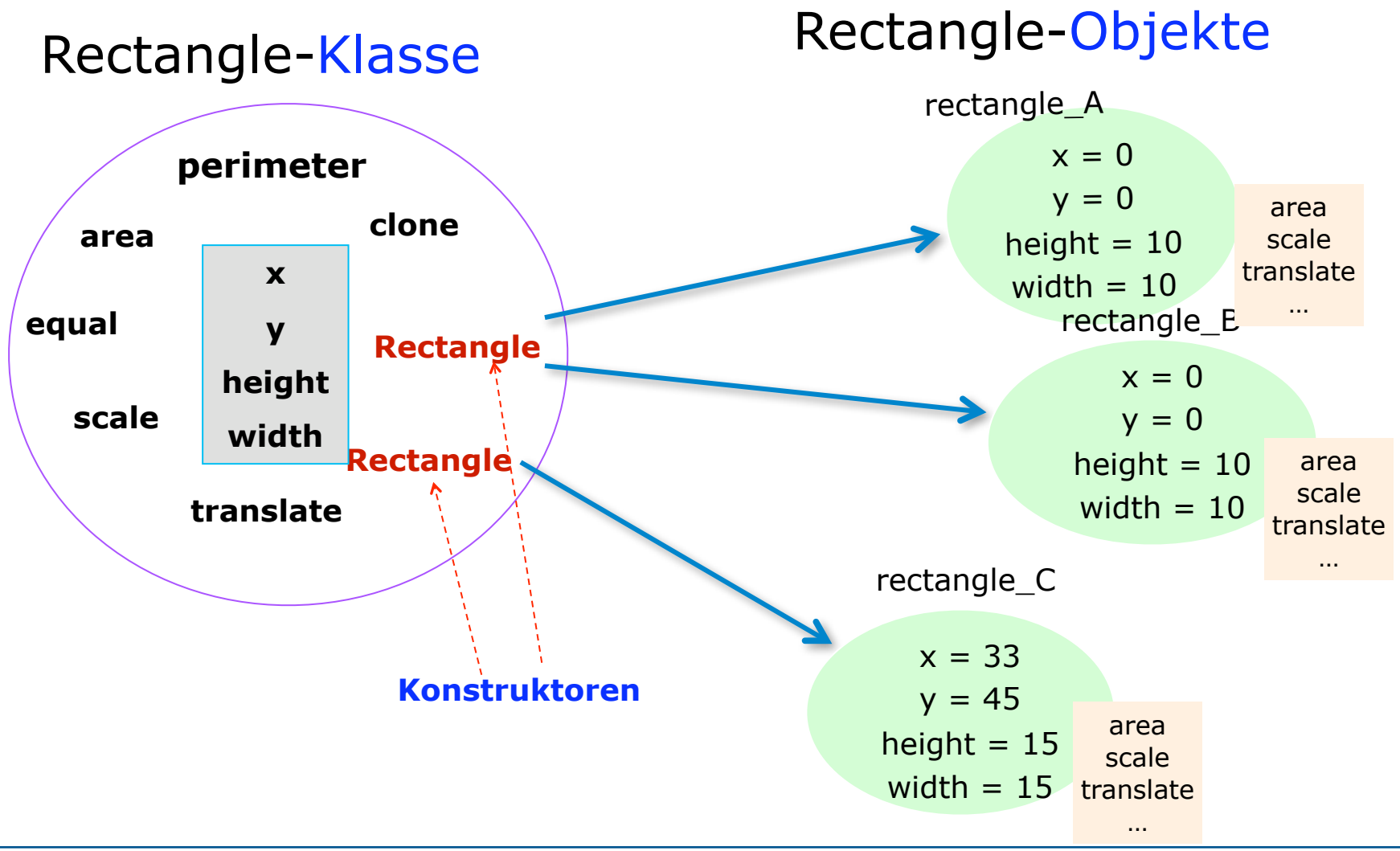
Mit dem wesentlichen Unterschied, dass Objekte immer einen zusätzlichen Zeiger auf die Tabelle der Klasse haben, nach deren Vorgabe diesen erzeugt wurden.



OO-Programmausführung

Die OOP betrachtet eine Programmausführung als ein System kooperierender Objekte.




Rectangle-Klasse
perimeter
area
clone
equal
x
y
height
width
scale
Rectangle
Rectangle
translate
Konstruktoren
Rectangle-Objekte
rectangle_A
x = 0
y = 0
height = 10
width = 10
area
scale
translate
...
rectangle_B
x = 0
y = 0
height = 10
width = 10
area
scale
translate
...
rectangle_C
x = 33
y = 45
height = 15
width = 15
area
scale
translate
...

Klasse Rectangle

in Java

```
public class TestRectangle{

// main-Methode
   public static void main( String[] args ) {
      Rectangle r1 = new Rectangle();
      Rectangle r2 = new Rectangle();
      int u = r1.area();
      int f = r2.perimeter();
  }
} // end of class TestRectangle
```

```
public class Rectangle{
  // Attribute
    int x;
    int y;
    int width;
    int height;
  // Konstruktoren
    public Rectangle() {
        x = 0;
        y = 0;
        width = 10;
        height = 10;
    }
  // Methoden
    public int perimeter() {
         return 2*(width + height);
    }
    public int area() {
          return (width * height);
    }
} // end of class Rectangle
```

OOP: Das Grundmodell

- Objekte haben einen lokalen Zustand.

- Objekte empfangen und bearbeiten Nachrichten.

  Ein Objekt kann

  - seinen Zustand ändern,
  - Nachrichten an andere Objekte verschicken,
  - neue Objekte erzeugen oder existierende Objekte löschen.

- Objekte sind grundsätzlich selbständige Ausführungseinheiten, die unabhängig voneinander und parallel arbeiten können.

Variante der for-Schleife

for-Schleifen besuchen oft jede Position eines Arrays oder einer Datenstruktur, und aus diesem Grund haben die Java-Entwickler ab Java 1.5 eine abgekürzte Form der **for**-Schleife eingeführt.

```
for ( Typ Element : Datenstrukturname )
    {   Anweisungen   }
```

Jedes Element der Datenstruktur wird der Variablen "Element" zugewiesen und innerhalb der Anweisungsblöcke benutzt.

Klassenmethoden

$$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

```
public class ForStatements {

        public static double pi_leibnitz( int n_max ){
                double sum = 0;
                for (int n = 0; n<=n_max; n++ ){
                        if ( n%2 == 0 )
                                sum = sum + (1.0)/(2*n+1);
                        else
                                sum = sum - (1.0)/(2*n+1);
                }
                return 4*sum;
        }
}
```

Java-Operatoren

Bezeichnung	Operator	Priorität
Komponentenzugriff bei Klassen	.	15
Komponentenzugriff bei Feldern	[ ]	15
Methodenaufruf	( )	15
Unäre Operatoren	++,--,+,-,~,!	14
Explizite Typkonvertierung	( )	13
Multiplikative Operatoren	*, /, %	12
Additive Operatoren	+, -	11
Schiebeoperatoren	<<, >>,>>>	10
Vergleichsoperatoren	<, >, <=, >=	9
Vergleichsoperatoren (Gleichheit, Ungleichheit)	==, !=	8
bitweise UND	&	7
bitweise exklusives ODER	^	6
bitweise inklusives ODER	|	5
logisches UND	&&	4
logisches ODER	||	3
Bedingungsoperator	? :	2
Zuweisungsoperatoren	=,*=,-=, usw.	1

Variablendeklarationen

Im Unterschied zu Python müssen in Java alle Variablen vor der ersten Anwendung deklariert werden.

Modifizierer	Typ	Name	Wert
	int	breite ;	
	int	hoehe =	10 ;
public	float	radius =	0.0 ;
private	float	radius =	0.0 ;

Der Modifizierer bestimmt die Zugriffsrechte, die andere Objekte auf eine Variable (Attribut) haben.

Java-Typsystem

- Java ist ***streng typisiert***, d.h. jeder Ausdruck hat einen wohldefinierten Typ.

- Die Einhaltung aller durch das Typsystem definierten Regeln wird

  - zuerst statisch vom **Übersetzer** überprüft.
  - und dann dynamisch vom **Interpreter**.

- Neue Objekttypen werden durch Klassen definiert.

  Variablen solcher Typen sind ***Referenzen****:*

Klasse Rectangle

```
public class Rectangle{
  // Attribute
    int x;
    int y;
    int width;
    int height;

  // Konstruktoren
    public Rectangle() {
        x = 0;
        y = 0;
        width = 10;
        height = 10;
    }

  // Methoden
    public int perimeter() {
         return 2*(width + height);
    }

    public int area() {
         return (width * height);
    }
} // end of class Rectangle
```

```
public class TestRectangle{

// main-Methode
   public static void main( String[] args ) {
       Rectangle r1 = new Rectangle();
       Rectangle r2 = new Rectangle();
       int u = r1.area();
       int f = r2.perimeter();
   }
} // end of class TestRectangle
```

Variablendeklarationen

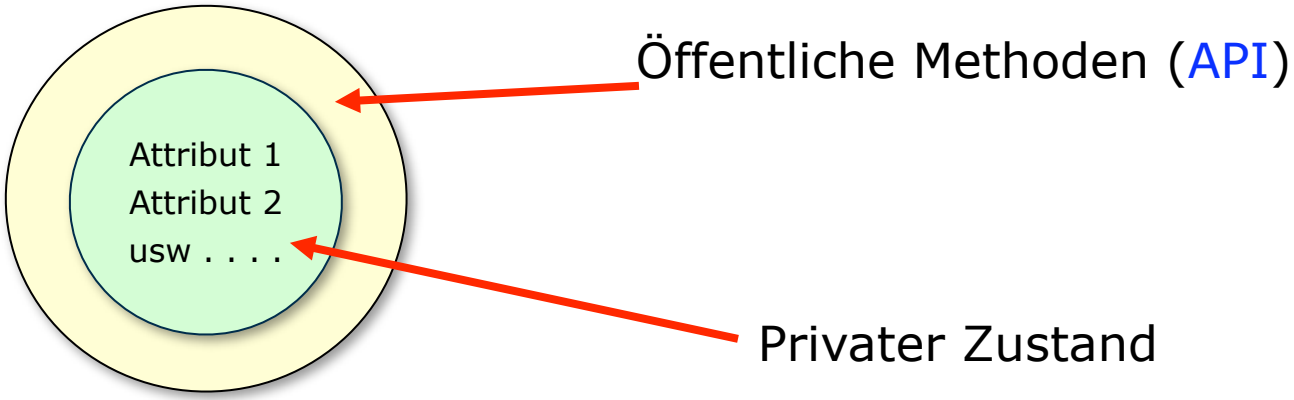
Modifizierer	Typ	Name	Wert
	int	breite ;	
	int	hoehe = 10 ;	
public	float	radius = 0.0 ;	
private	float	radius = 0.0 ;	

Der Modifizierer bestimmt die Zugriffsrechte, die andere Objekte auf eine Variable (Attribut) haben.

Was ist Kapselung?

Kapselung ist die Einschränkung des Zugriffs auf die Instanzvariablen eines Objektes durch Objekte anderer Klassen.
Man spricht von einer **Kapselung** des Objektzustands.

API **A**pplication **P**rogramming **I**nterface.



Kapselung schützt so den Objektzustand vor "unsachgemäßer" Änderung und unterstützt **Datenabstraktion**.

Kapselung erfolgt durch die Zugriffsmodifizierer (public, private, protected und package)

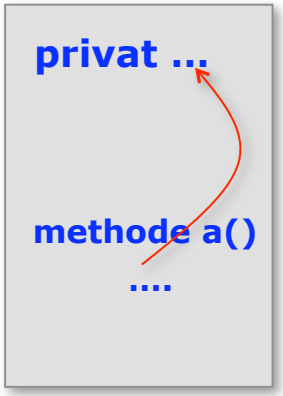
Sichtbarkeit von Java-Variablen

Zugriffsangabe oder Sichtbarkeit	Klasse	Unterklassen	Paket	Welt
privat	✓			
package	✓		✓	
protected	✓	✓	✓	
public	✓	✓	✓	✓

Kein Modifikator ist äquivalent zur package

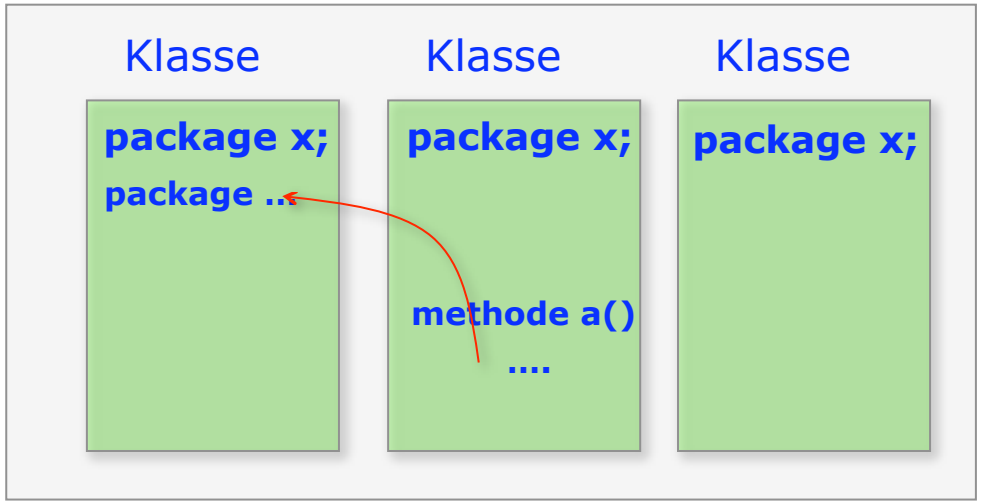
Zugriffsangabe oder Sichtbarkeit von Variablen

Klasse



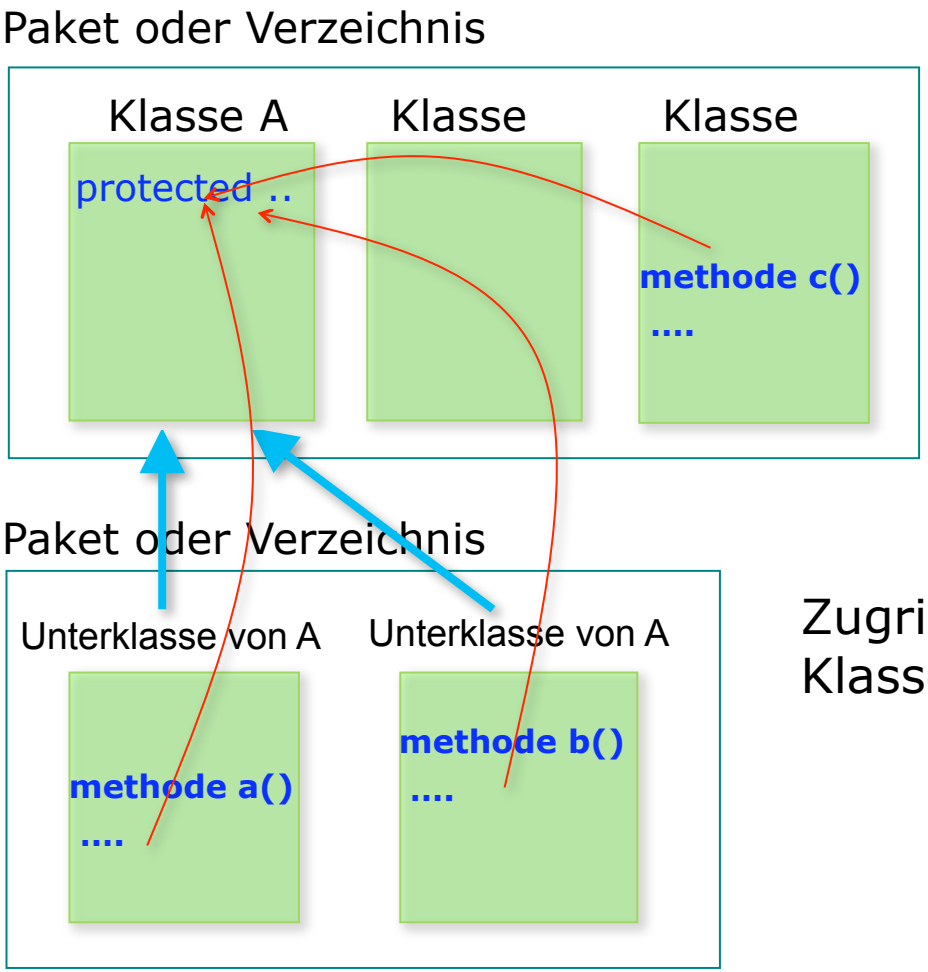
Nur das Objekt selbst kann den Inhalt einer privaten Variablen mittels seiner Methoden modifizieren.

Paket- oder Verzeichnis-x



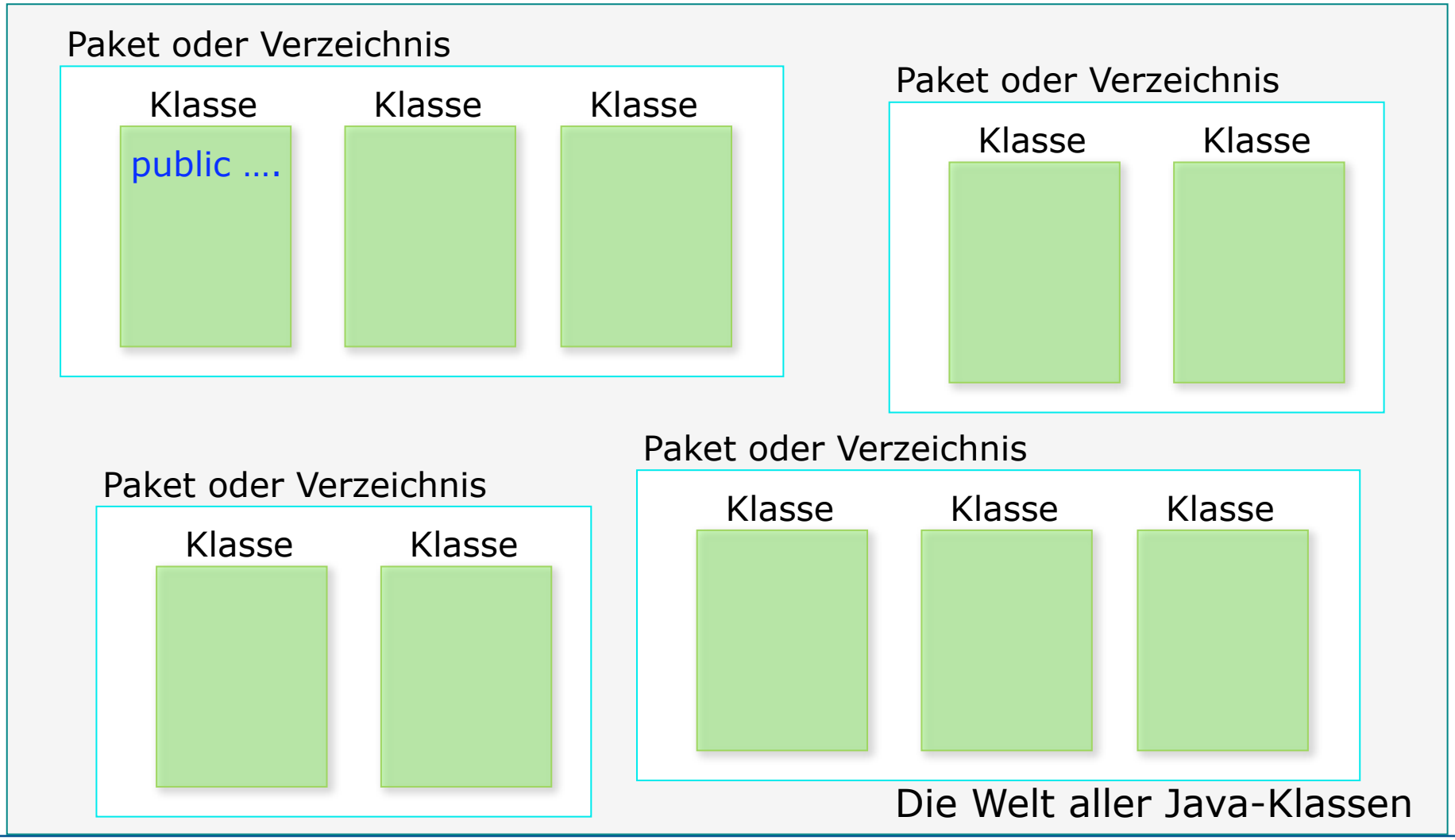
Alle Objekte innerhalb eines Verzeichnisses haben direkten Zugriff auf eine package-Variable.

Zugriffsangabe oder Sichtbarkeit von Variablen



Zugriff innerhalb der Klassen-Hierarchie

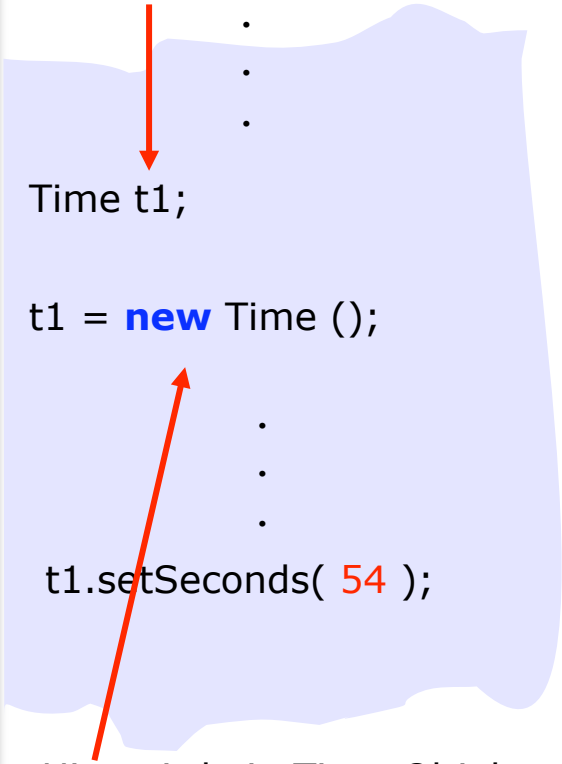
Zugriffsangabe oder Sichtbarkeit von Variablen



Kapselung

```
public class  Time {
 // Instanzvariablen oder Eigenschaften
         private int hours ;
         private int minutes ;
         private int seconds ;
 // Konstruktor
        public Time( ) {
              hours = 0;   minutes = 0;   seconds = 0;
         }
 // Methoden
         public int getHours ( ) {
              return hours;
         }
         public void setHours ( int neuHours ) {
              if  ( neuHours>=0 && neuHours<24 )
                   { hours= neuHours; }
         }
         public int toSeconds( ) {
              // Lokale Variablen
             int temp ;
             temp = seconds + minutes*60 + hours*3600;
             return temp;
         }
         . . . .
} // Ende der Klassedeklaration
```

Eine Variable t1 wird erzeugt, um die Referenz (Adresse) eines Objektes des Typs "Time" zu speichern.



Hier wird ein Time-Objekt erzeugt und eine Referenz zum Objekt in der Variablen t1 gespeichert.

Beispiel:

Definition der setTime-Methode

```
. . .
public void setTime( int h,  int m,  int s )  {
      if  (  (s>59) || (s<0) || (m>59)  || (m<0) || (h>23)  || (h<0) ){
            System.out.println("Falsche Zeitangabe:"+h+":"+m+":"+s);
      } else {
            hours = h;     minutes = m;     seconds = s;
      }
 }
. . .
```

Beispiel:

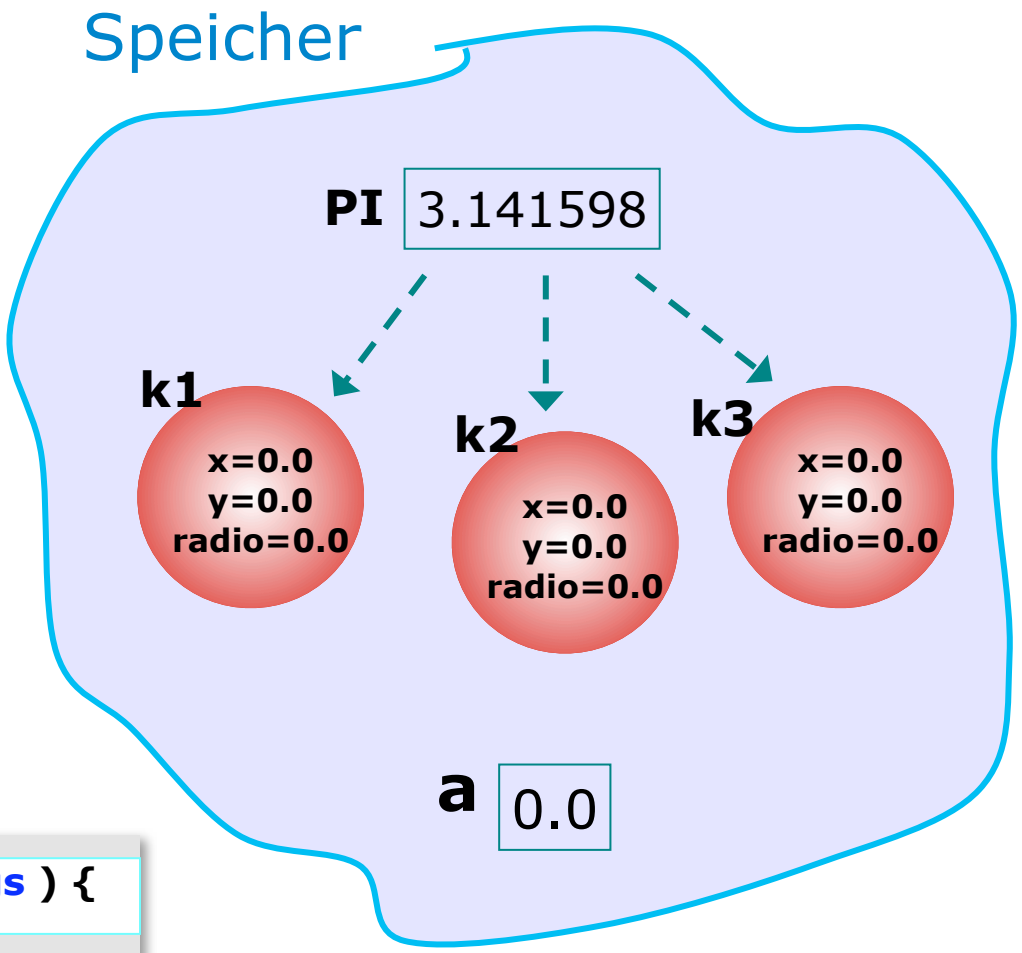
```
public class Kreis {
    // Instanzvariablen
     float x;
     float y;
     float radio;

    // Klassenvariable
     public final float PI = 3.141598f;

   // Methoden
     public float area() {
              // Lokale Variable
          float a;
           a = PI*radio*radio;
           return a;
      }  ...
}
```

ab hier!

Beispiel:

```
public class Kreis {
   //Instanzvariablen
      float x;
      float y;
      float radio;
   // Klassenvariable
      public final float PI = 3.141598f;
   // Methoden
      public float area() {
       // Lokale Variable
          float a;
          a = PI*radio*radio;
          return a;
      }
}
```

```
public static void main ( String[] args ) {
      Kreis k1 = new Kreis();
      Kreis k2 = new Kreis();
      Kreis k3 = new Kreis();
      float flaeche = k1.area();
}
```



Objekterzeugung

Objekte werden durch den Aufruf von Konstruktoren erzeugt.

Ein Konstruktor wird mit Hilfe der **new**-Operatoren aufgerufen.

```
Kreis k1 = new Kreis();
```

Eine Klassendefinition kann mehrere Konstruktoren haben mit verschiedenen Initialisierungen der Objekteigenschaften.

Wenn in einer Klasse keine Konstruktoren definiert worden sind, werden die Eigenschaften von Objekten mit Defaultwerten initialisiert.

Variablen in Java

Zugriff

Variablen

- Klassenvariablen

```
float delta = Kreis.PI / 10;
```

bezüglich des Klassennamens

- Instanzvariablen

```
Kreis k1 = new Kreis();
k1.radio = 5.0;
```

nur in Bezug zu einem Objekt

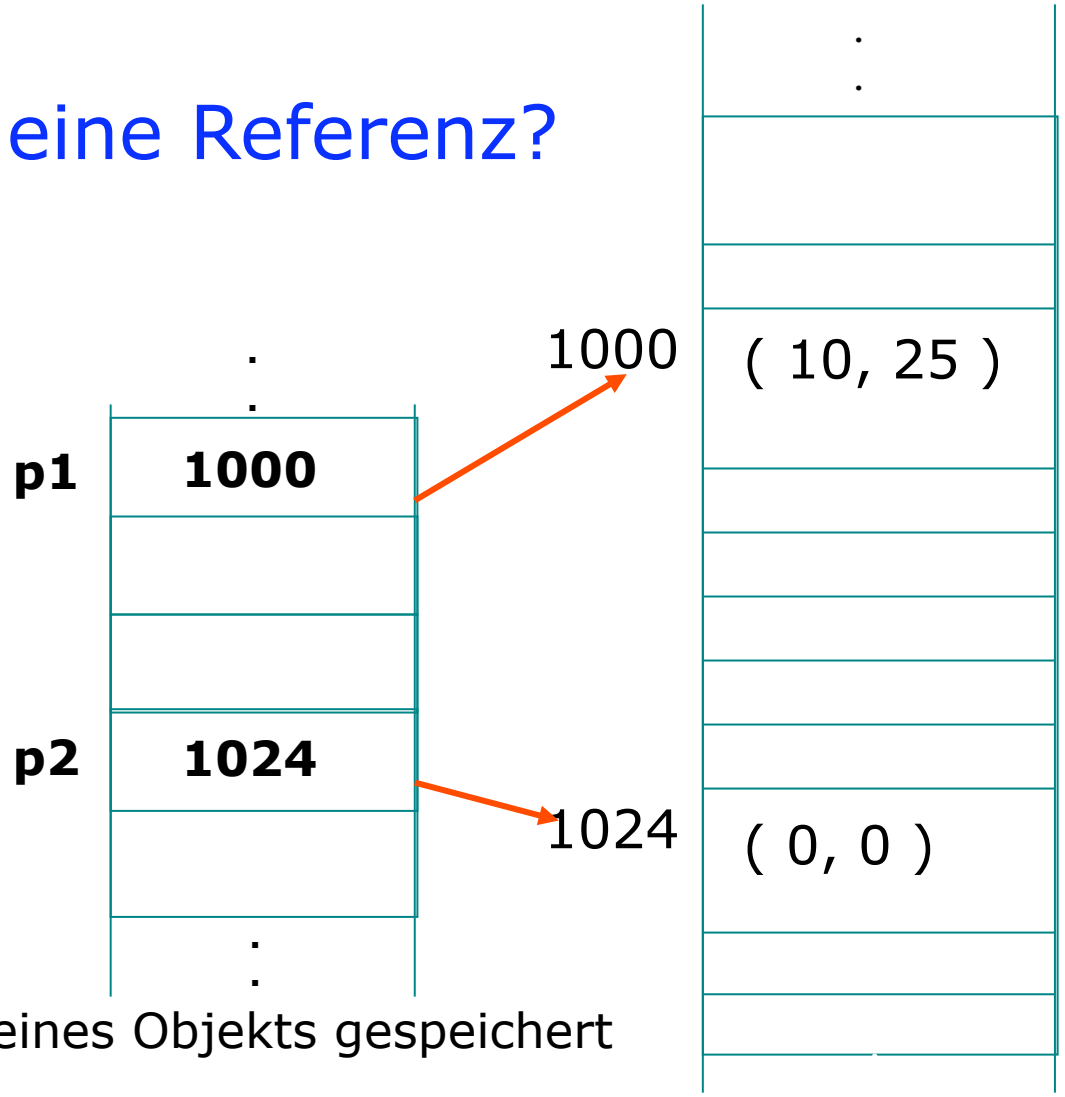
- Lokale Variablen

```
int num;
num = 5.0;
```

wie in C

Was ist eine Referenz?

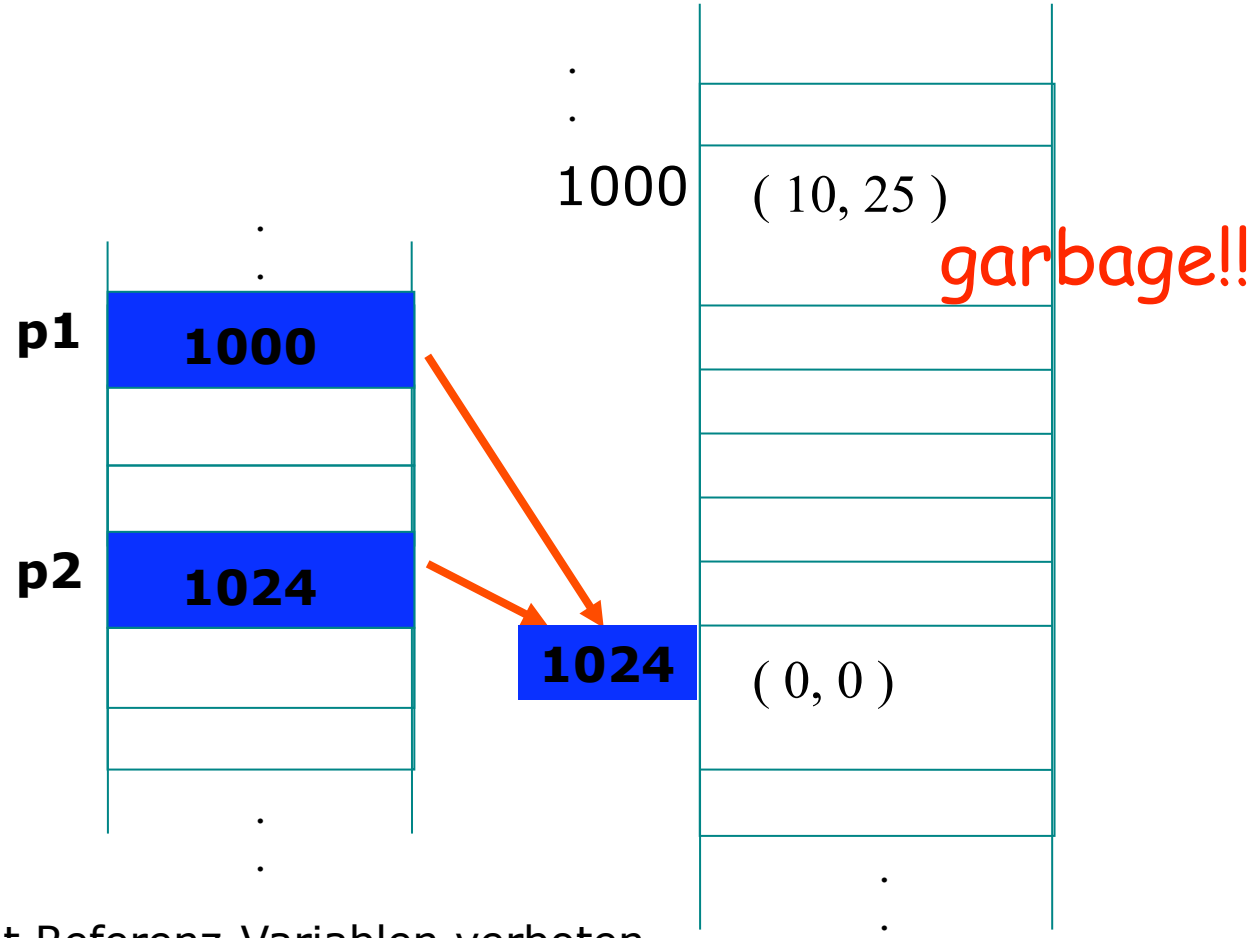
```
Point p1;

Point p2;

p1 = new Point ( 10, 25 );

p2 = new Point ( 0, 0 );
```



sind Variablen, wo die Adresse eines Objekts gespeichert wird

Referenz-Variablen

p1 = p2 ;



In Java ist Arithmetik mit Referenz-Variablen verboten.
Nur die Operatoren **=, ==, !=** sind erlaubt.

Weitere vordefinierte Operationen, die mit Referenz-Variablen erlaubt sind

(Typ)_Operator

```
Object  objekt  =  null;
Button  button  =  (Button)  objekt;
```

Der (**.**) Operator

Mit dem (**.**)-Operator hat man Zugriff auf die Eigenschaften und Methoden eines Objekts.

```
Punkt  p1 = new Punkt ( 10, 35 );
int x_koord = p1.x ;
```

null

Das Schlüsselwort **null** bezeichnet immer ungültige, d.h. nicht initialisierte Referenzen.

null kann überall da verwendet werden, wo eine Referenz erwartet wird. Zugriff auf eine Referenz, die gleich null ist, erzeugt einen Laufzeitfehler.

(NullPointerException)

```
Rechteck r1;
Rechteck r2 = null;
...
r1.gleich( r2 );
```

Verursacht einen Laufzeitfehler!

Typ-Operatoren

unär

Operator	Zeichen	Rtg.	Beispiel	Priorität
"cast"-Operator	(*typ*)		(int) a	13

binär

Typvergleich für Objekte	instanceof		a instanceof Button	9

```
Button knopf = new Button();
   ...
if ( knopf instanceof Button )
      knopf.setBackground(Color.yellow);
   ...
```

Java-Operatoren

Bezeichnung	Operator	Priorität
Komponentenzugriff bei Klassen	**.**	15
Komponentenzugriff bei Feldern	**[ ]**	15
Methodenaufruf	**( )**	15
Unäre Operatoren	**++,--,+,-,~,!**	14
Explizite Typkonvertierung	**( )**	13
Multiplikative Operatoren	***, /, %**	12
Additive Operatoren	**+, -**	11
Schiebeoperatoren	**<<, >>,>>>**	10
Vergleichsoperatoren	**<, >, <=, >=**	9
Vergleichsoperatoren (Gleichheit, Ungleichheit)	**==, !=**	8
bitweise UND	**&**	7
bitweise exklusives ODER	**^**	6
bitweise inklusives ODER	**|**	5
logisches UND	**&&**	4
logisches ODER	**||**	3
Bedingungsoperator	**? :**	2
Zuweisungsoperatoren	**=,*=,-=, usw.**	1

Konzepte objektorientierter Programmierung

- Objekte ✓
- Klassen ✓
- Nachrichten ✓
- Kapselung ✓
- Vererbung
- Polymorphismus

Imperative Grundbestandteile

1. Was ist eine Variable ? ✓
2. Primitive Datentypen in Java ✓
3. Deklaration von Variablen (OOP) ✓
6. Einfache Anweisungen in Java ✓
4. Ausdrücke in Java ✓
5. Die vielen Operatoren von Java ✓
7. Anweisungen zur Ablaufsteuerung ✓

OOP: Das Grundmodell

- Objekte haben einen lokalen Zustand

- Objekte empfangen und bearbeiten Nachrichten

  Ein Objekt kann

  - seinen Zustand ändern,
  - Nachrichten an andere Objekte verschicken,
  - neue Objekte erzeugen oder existierende Objekte löschen.

- Objekte sind grundsätzlich selbständige Ausführungseinheiten, die unabhängig voneinander und parallel arbeiten können.

Klassenmethoden

Klassenmethoden haben keinen Zugriff auf Instanzvariablen.

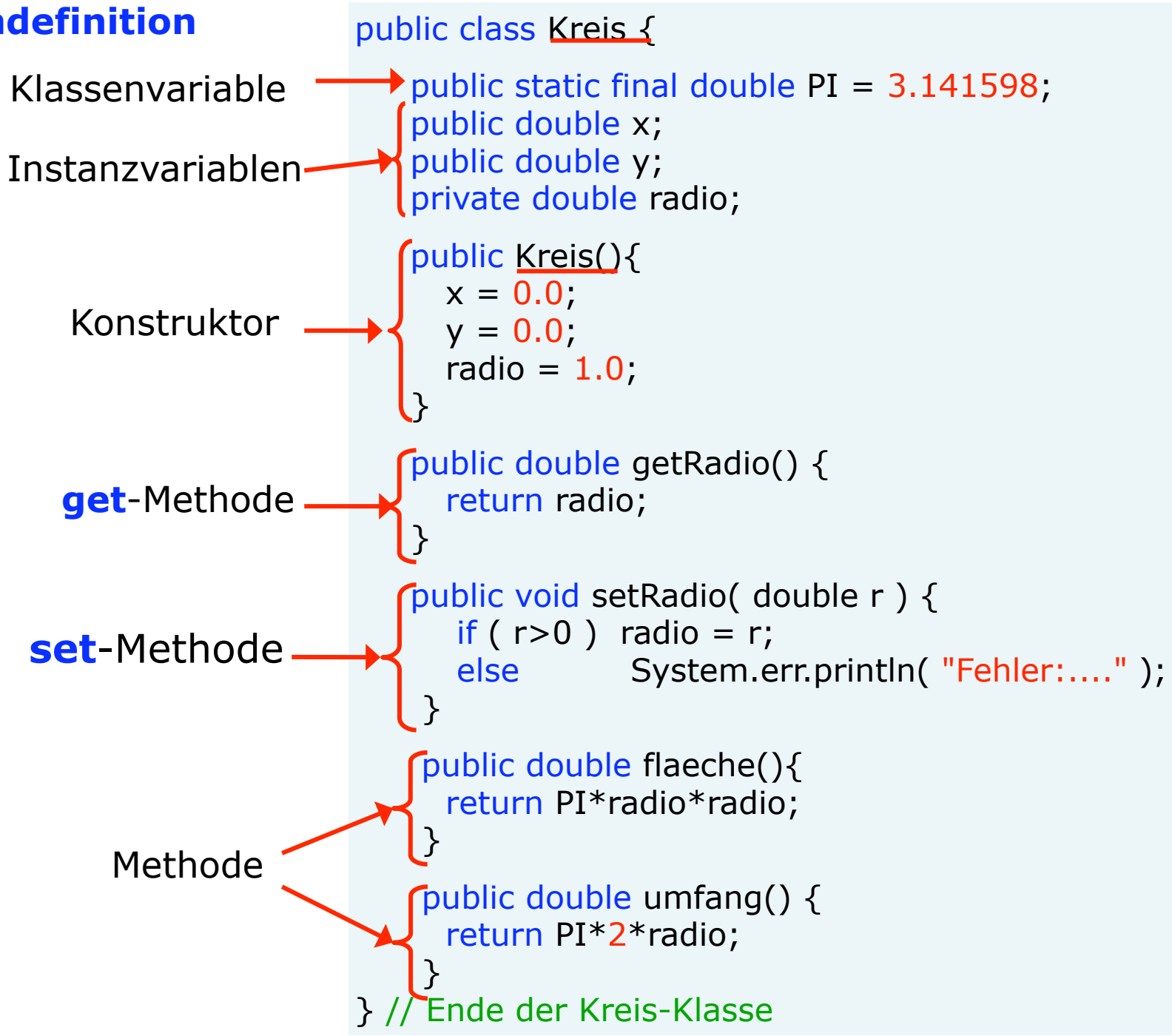
Klassenmethoden dürfen nur andere Klassenvariablen oder lokale Variablen verwenden.

Innerhalb einer als static deklarierten Methode (Klassenmethode) dürfen nur andere statische Methoden aufgerufen werden.

Weitere Konventionen

Variablennamen	beginnen mit Kleinbuchstaben myName
Klassennamen	beginnen mit Großbuchstaben Rechteck
Konstante Klassenvariablen	nur Großbuchstaben BLAU
Methoden	beginnen mit Kleinbuchstaben setColor (BLAU)

Klassendefinition




```
public class Kreis {
   public static final double PI = 3.141598;
   public double x;
   public double y;
   private double radio;

   public Kreis(){
      x = 0.0;
      y = 0.0;
      radio = 1.0;
   }

   public double getRadio() {
      return radio;
   }

   public void setRadio( double r ) {
       if ( r>0 )  radio = r;
       else        System.err.println( "Fehler:...." );
   }

   public double flaeche(){
      return PI*radio*radio;
   }

   public double umfang() {
      return PI*2*radio;
   }
} // Ende der Kreis-Klasse
```

Instanzvariablen

Die Klasse Kreis vereinbart drei *Instanzvariablen* mit jeweils einem *Typ*, einem *Namen* und einem *Wert*.

```
...
Kreis k = new Kreis();
k.x  = 0;
k.y  = 0;
...
```

Zugriff nach dem Muster <Referenz> . <Feldname>

Instanzvariablen werden beim Erzeugen des Objekts entweder mit dem im Konstruktor angegebenen Wert initialisiert oder mit einem Standardwert:

Konstruktoren

Ein guter OOP-Stil bedeutet, geeignete *Konstruktoren* zu definieren, die Objekte initialisieren und evtl. initiale Berechnungen durchführen.

```
public class Beverage {
  String name;
  int price,
  int stock;
                  // Konstruktor
  Beverage(  String name, int price, int stock ) {
    this.name   = name;
    this.price  = price;
    this.stock  = stock;
  }
  . . .
}
```

Ist kein Konstruktor definiert, wird ein impliziter Konstruktor ohne Argumente angenommen.

```
public class Kreis {

    double x, y, radio;

...

}
```

```
public class Kreis {

    double x, y, radio;

    public Kreis(){

    }

    ...

}
```

Sobald ein expliziter Konstruktor definiert ist, fällt der implizite Konstruktor weg!

this

Das Schlüsselwort **this** bezeichnet immer eine Referenz auf das aktuelle Objekt selbst.

this kann in Methoden und in Konstruktoren verwendet werden, um durch Argumentnamen "verschattete" Variablennamen zu erreichen:

this

Referenz auf das aktuelle Objekt selbst

```
public class Kreis {

    public final double PI = 3.141598;
    public double x;
    public double y;
    private double radio;

    public Kreis( double x, double y ){
        this.x = x;
        this.y = y;
    }

    public double getRadio() {
       return this.radio;
    }

    public void setRadio( double r ) {
        if ( r>0 )
                this.radio = r;
        else
                System.err.println( "Fehler:...." );
     }
   ...
```

Konstruktoren

"gute Regel" bei mehreren Konstruktoren:

Schreibe *genau einen* Konstruktor, der alle Initialisierungen vornimmt und rufe ihn aus den anderen mit geeigneten Parametern auf. Dies vermindert die Zahl potentieller Fehler.

```
public class Kreis {
    double x, y, r;
    public Kreis ( double x, double y, double radio ) {
        this.x = x;   this.y = y;    this.radio = radio;
    }
    public Kreis ( double r ) { this ( 0.0, 0.0, radio ); }
    public Kreis ( Kreis c )  { this ( c.x, c.y, c.radio ); }
    public Kreis ()           { this ( 1.0 ); }
}
```

Objekterzeugung

```
. . .
Kreis first_circle  =  new Kreis ( 0.0, 0.0, 1.0 );
Kreis second_circle  =  new Kreis ( first_circle );
Kreis four_circle = new Kreis ( 5.0 );
Kreis third_circle  =  new Kreis ( );
. . .
```

Instanzmethoden

Instanzmethoden definieren das Verhalten von Objekten.

Sie werden innerhalb einer Klassendefinition angelegt und haben Zugriff auf alle Variablen des Objekts.

Sie haben immer den impliziten Parameter **this**

```
public class Person {
  private String name = "";
   ...
  String getName() {
        return this.name;
  }

  void setName( String name ) {
        this.name = name;
   }
}
```

Test-Beispiel für die Kreis-Klasse

```
public class TestKreis {

   public static void main(String[] args) {
      Kreis k = new Kreis();
      k.x = 1.0;
      k.y = 5.0;
      k.setRadio( 30 );
      k.setRadio( -6 );
      System.out.println( k.flaeche() );
      System.out.println( k.umfang() );
      double radio = k.getRadio();
      System.out.println(radio);
      System.out.println( k.getRadio() );
   }
}
```

```
Fehler:....
2827.4382
188.49588
30.0
30.0
```

Packages

Java bietet die Möglichkeit, miteinander in Beziehung stehende Klassen zu Paketen (packages) zusammenzufassen. Die Zugehörigkeit zu einem Paket wird über die **package**-Direktive deklariert.

Einzelne oder alle Klassen eines anderen Pakets werden mit der **import**-Direktive "sichtbar" gemacht.

Packages und Klassennamen

Wird kein Package bestimmt, so gehört die Klasse automatisch ins globale, unbenannte Package.

Der vollständig qualifizierte Name einer Klasse wird aus dem Klassennamen und allen umschließenden Package-Namen gebildet,

z.B.

```
. . .
java.awt.Button  button;
. . .
```

Wenn eine entsprechende **import**-Anweisung vorhanden ist

```
import java.awt.*;
. . .
Button button;
```

schreibe ich nur →

Packages

```
package beispiele;

import java.lang.*; // Standard

public class Person { ... }
```

```
package uebungen;
import beispiele.*;
...
    java.lang.String str = "";
    Person p1;
...
```

Zugriffskontrolle

Freie Universität Berlin

Ein guter OOP-Stil tendiert dazu, alle Instanzvariablen privat zu deklarieren und den Zugriff nur durch "set"- und "get"- Methoden zu ermöglichen.

```
public class Person {
    private  int alter;
    ...
```

andere Klasse

```
...
Person person = new Person( );
int a = person.alter;
...
```

Übersetzerfehler!!

Zugriffskontrolle auf Methoden

Der Zugriff auf Methoden kann auch durch *Modifizierer* gesteuert werden:

- **public**: überall zugänglich.
- **private**: nur innerhalb der eigenen Klasse zugänglich.
- **protected**: in anderen Klassen des selben Packages und in Unterklassen zugänglich.

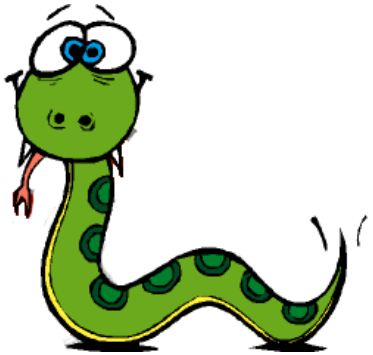
- **kein Modifizierer**: sind nur für Code im selben ***Paket*** zugänglich.

Freie Universität Berlin

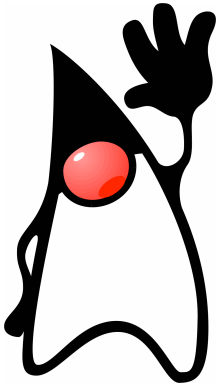
Algorithmen und Programmieren II

Objektorientiertes Programmieren

(Einführung)



{P} S {Q}



SS 2012

Prof. Dr. Margarita Esponda

Beispiel:

Die Haus-Klasse

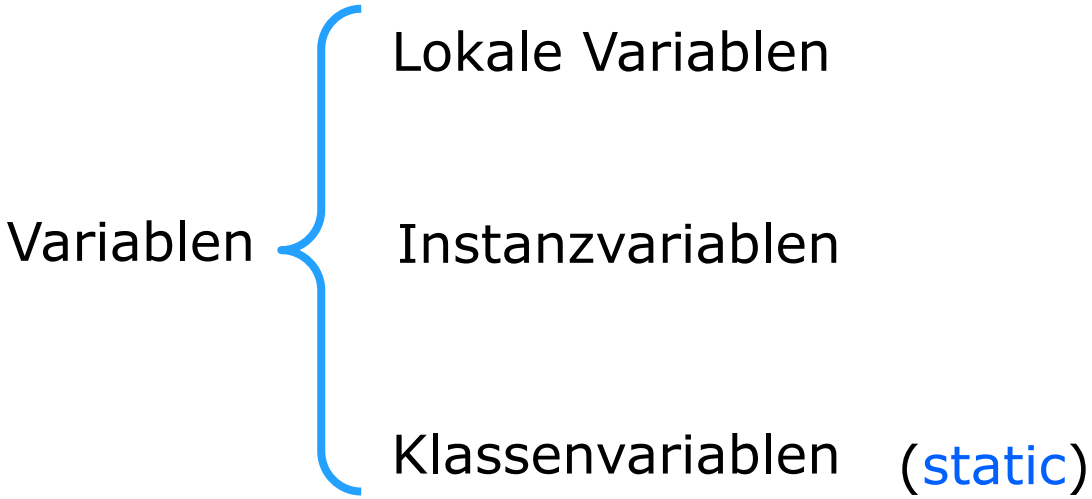
Klassendefinition

Eigenschaften:	Operationen:
Etagen	sanieren
Wohnfläche	renovieren
Nutzfläche	verkaufen
Adresse	

Ein konkretes Haus Objekt

Zustand:	Operationen:
zwei Etagen	kann saniert werden
100 m^2 Wohnfläche	kann renoviert werden
200 m^2 Nutzfläche	kann verkauft werden
Takustr. 20	

Variablen in Java



Diese Klassifikation richtet sich nach folgenden zwei Aspekten:

Ort der Deklaration

Vorhandensein der (static)-Deklarationsspezifizierer

Variablen in Java

Instanzvariablen
(Feldvariablen, Attribute)

Variablen, in denen die Eigenschaften von Objekten gespeichert werden

Lebenszeit: nur solange das Objekt existiert.

Lokale Variablen

Hilfsvariable für Berechnungen

Sie werden innerhalb von Methoden deklariert.

Lebenszeit: nur solange die Methode ausgeführt wird.

Klassenvariablen

Variablen, die zu einer Klasse gehören

Lebenszeit: solange das Programm ausgeführt wird.

Klassenvariablen

Klassenvariablen haben den Deklarationsspezifizierer static

static Variablen sind klassenbezogen

..d.h. speichern Eigenschaften, die für eine ganze Klasse gültig sind, und von denen nur ein Exemplar für alle Objekte der Klasse existiert; ihre Lebensdauer erstreckt sich über das ganze Programm.

final sind nicht modifizierbar

Die Mensch-Klasse in Java

Eigenschaften

Konstruktor

Methode

```
public class Mensch {
        int beine;
        int arme;
        int kopf;
        String name;
        String geschlecht;

        public Mensch(){
                beine = 2;
                arme = 2;
                kopf = 1;
                name = "Felix";
                geschlecht = "männlich";
        }

        public String sagDeineName(){
                return name;
        }
}
```

Beispiel:

Die Haus-Klasse

Klassendefinition

Eigenschaften:	Operationen:
Etagen	sanieren
Wohnfläche	renovieren
Nutzfläche	verkaufen
Adresse	

Ein konkretes Haus Objekt

Zustand:	Operationen:
zwei Etagen	kann saniert werden
100 m^2 Wohnfläche	kann renoviert werden
200 m^2 Nutzfläche	kann verkauft werden
Takustr. 20	