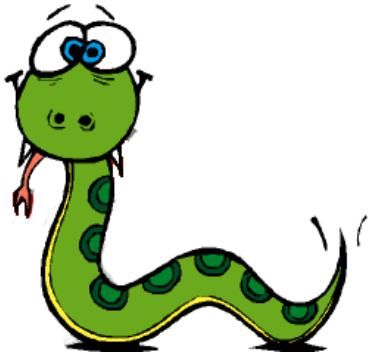
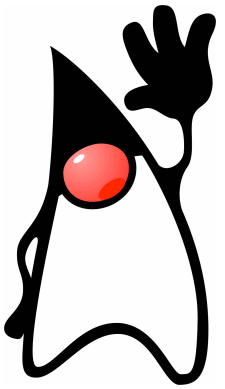
Freie Universität Berlin

Algorithmen und Programmieren II

Objektorientiertes Programmieren (Teil II)



{P} S {Q}



SS 2012

Prof. Dr. Margarita Esponda

Variante der for-Schleife

Beispiel:

```
. . .
double[] nums = { 3.4, 5.6, 1.2, 2.3, 5.6, 7.8 };
double prod = 1;

for ( double d : nums ) {
        prod = prod*d;
}
. . .
```

```
public class ForSchleifen {

      int size = 20;
      Random rand = new Random();
      Time[][] walkSpace;
   ...
   public void initWalkSpace(){

      walkSpace = new Time[size][size];

      for(int i=0; i<walkSpace.length; i++){
            for(int j=0; j<walkSpace.length; j++){
                  walkSpace[i][j] = new Time(rand.nextInt(23),rand.nextInt(59),rand.nextInt(59));
            }
      }
   }

   public void print(){
       for (Time[] line : walkSpace){
            for ( Time time : line){
                  time.print();
            }
       }
   } ...
```

Aufzählungsdatentyp

Aufzählungsdatentypen erlauben es uns, Variablen zu definieren, denen nur eine bestimmte Anzahl von konstanten Namen zugewiesen werden kann.

Der Aufzählungsdatentyp in Java ist viel mächtiger als in allen anderen Programmiersprachen.

Vorteile:

- Statische Typüberprüfung ist möglich (Compiler)
- Die Programme sind viel lesbarer
- Typ-spezifische Operationen sind definierbar
- Der Implementierungsaufwand ist sehr gering

Kleiner Datentyp, der innerhalb einer Klasse definiert wird.
Dadurch Vermeidung zu vieler kleiner Klassendefinitionen.

Aufzählungs-Datentypen

```
public class Enum_Beispiel {
        public enum Season {WINTER, SPRING, SUMMER, FALL};
        Season s = Season.FALL;

        public Season jahreszeit() {
                return s;
        }
        public static void main( String[] main ){
                Season s1 = Season.SUMMER;
                Season s2 = Season.FALL;
                System.out.println( s1 );
                System.out.println( s1.equals(s2) );
                Enum_Beispiel e = new Enum_Beispiel();
                System.out.println( e.jahreszeit() );
                //  Season s3 = 1; // Typfehler!!!
        }
}
```

Ein neuer Datentyp mit dem Namen Season wurde vereinbart, der nur die konstanten Namen Season.WINTER, Season.SPRING, Season.SUMMER und Season.FALL annehmen kann.

Ausgabe:

```
SUMMER
false
FALL
```

Beispiele:

```
public class Enum_Beispiele {

    public enum Colors{ RED, YELLOW, GREEN};
    public enum Direction { SOUTH, NORTH, EAST, WEST };
    public enum Geldschein {
                        FUENF ( 5 ),
                        ZEHN ( 10 ),
                        ZWANZIG ( 20 ),
                        FUENFZIG ( 50 ),
                        HUNDERT ( 100 );

                        private Geldschein(int w){
                            wert = w;
                        }
                         final int getWert(){
                            return wert;
                         }
                        private final int wert;
    };
}
```

```
// Der Läufer bewegt sich, wenn Platz vorhanden ist.
public void next_step() {
        if ( has_space() ) {
            int d = (int)(Math.random()*100)%4;
            switch( Direction.values()[d] ) {
                case NORTH: if ( y>0 && (ws[x][y-1]==0) )
                                ws[x][--y] = stepNumber++;  break;
                case EAST:  if ( x<(ws.length-1) && (ws[x+1][y]==0) )
                                ws[++x][y] = stepNumber++;  break;
                case SOUTH: if ( y<(ws.length-1) && (ws[x][y+1]==0) )
                                ws[x][++y] = stepNumber++;  break;
                case WEST:  if ( x>0 && (ws[x-1][y]==0) )
                                ws[--x][y] = stepNumber++;  break;
            }
        }
}// end of next_walk
```

Beispiel:

Die Haus-Klasse

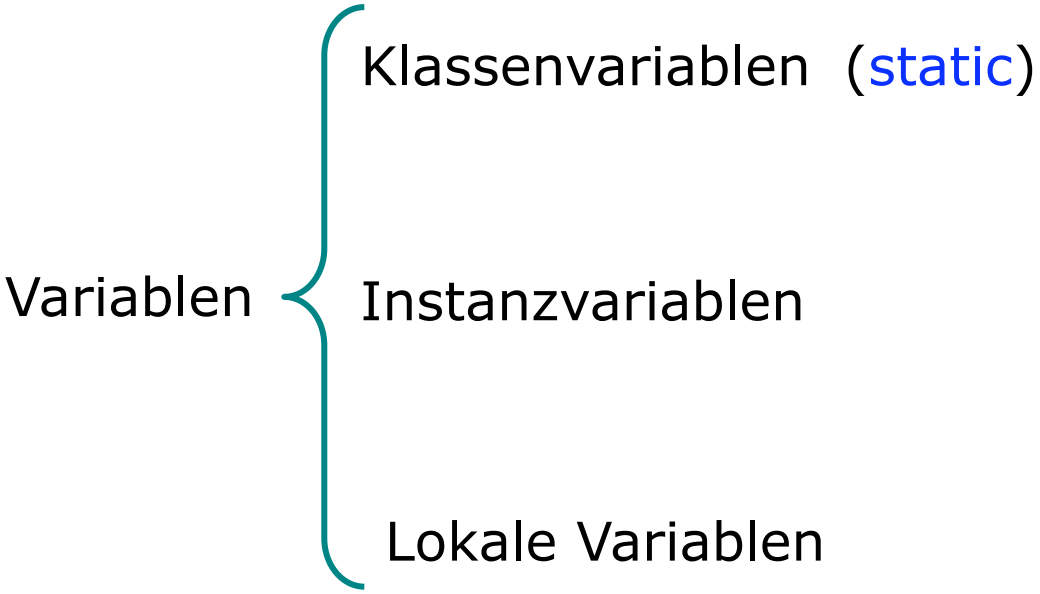
Klassendefinition

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

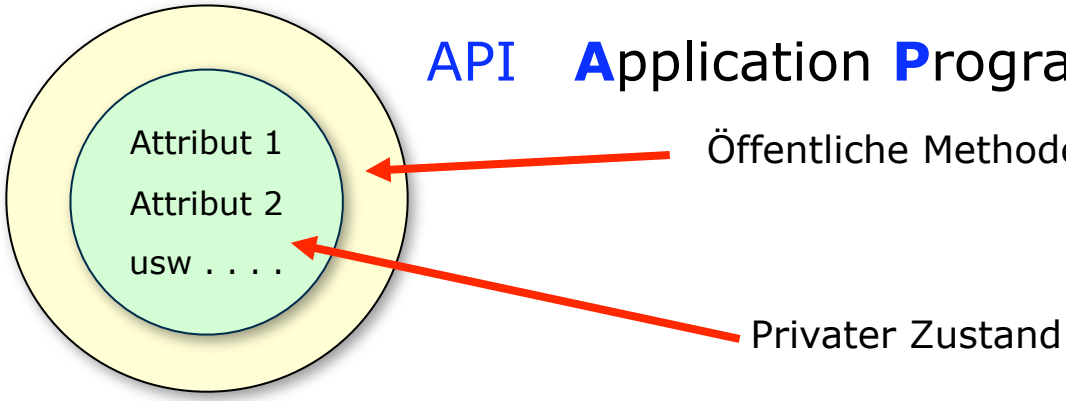
Konstruktor

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;
        }
}
```

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)

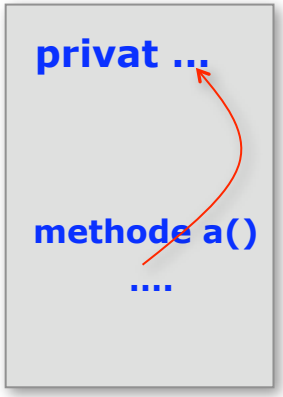
Sichtbarkeit von Java-Variablen

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

Kein Modifikator ist äquivalent zur package

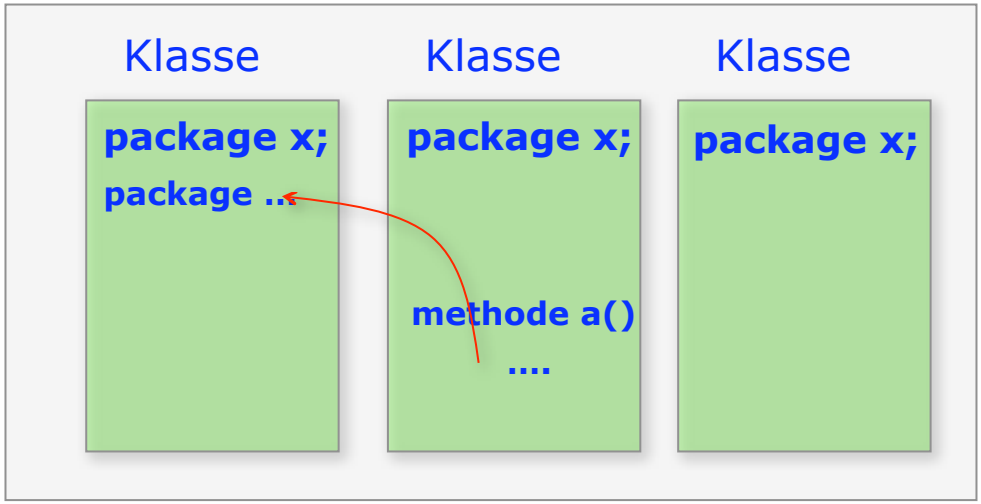
Zugriffsangabe oder Sichtbarkeit von Variablen

Klasse



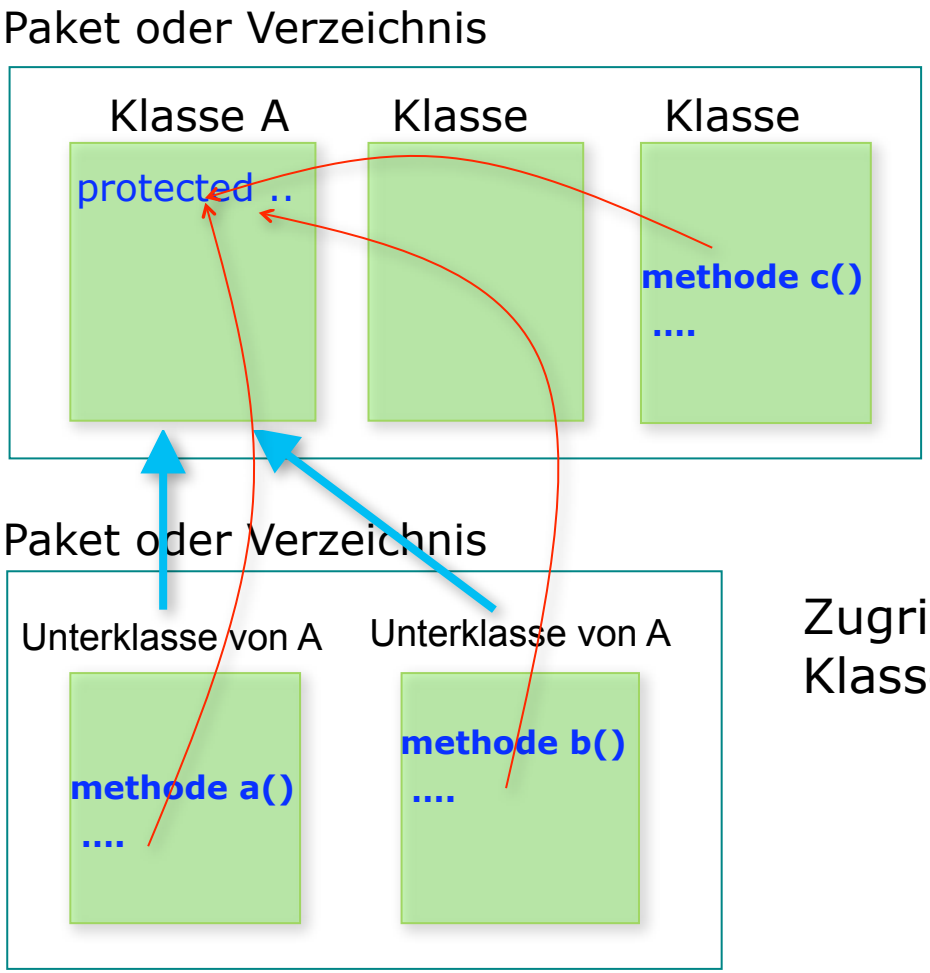
Nur das Objekt selbst kann den Inhalt einer privaten Variablen mittels seiner Methoden modifizieren.

Paket- oder Verzeichnis-x



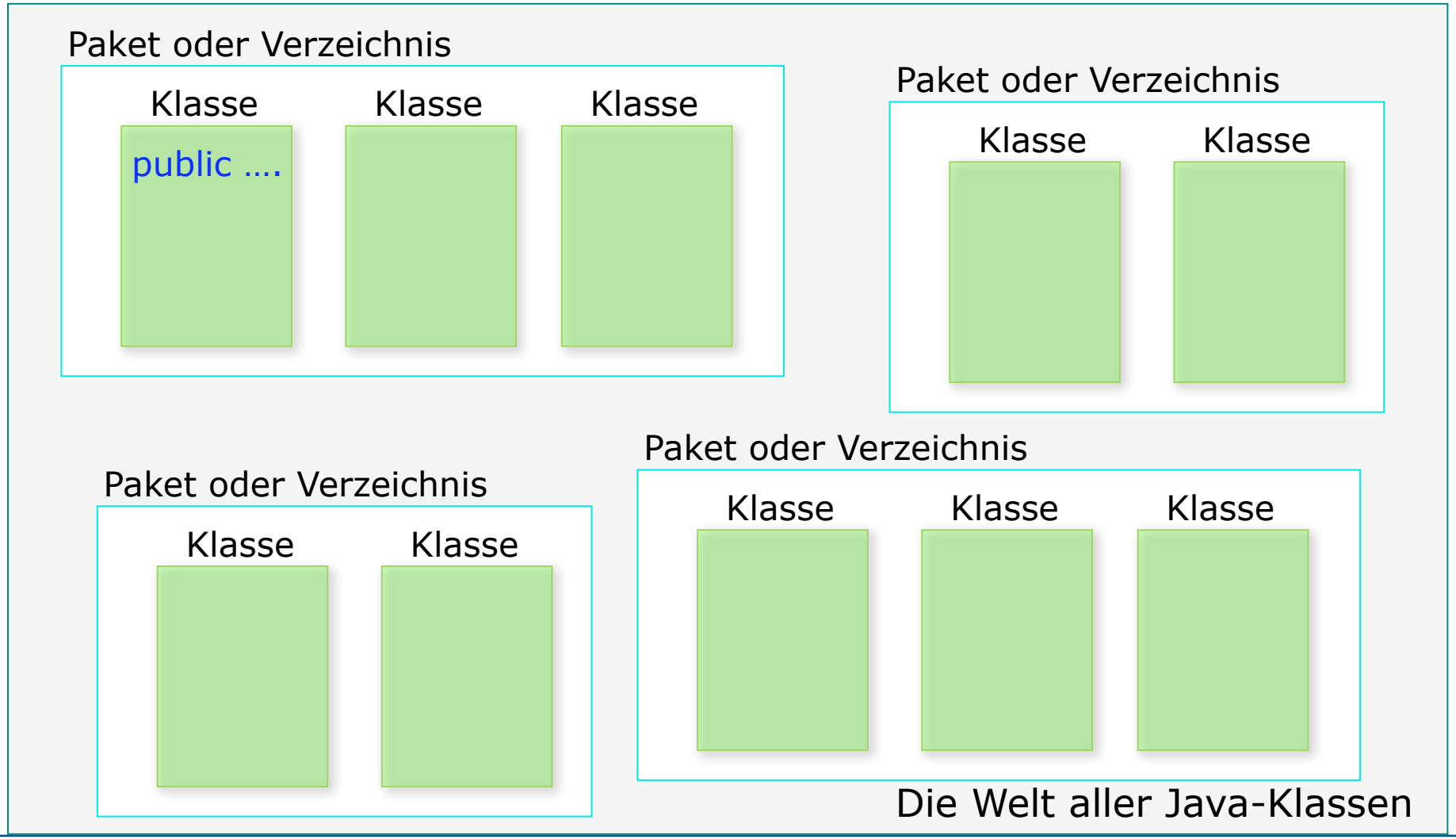
Alle Objekte innerhalb eines Verzeichnisses haben direkten Zugriff auf eine package-Variable.

Zugriffsangabe oder Sichtbarkeit von Variablen



Zugriff innerhalb der Klassen-Hierarchie

Zugriffsangabe oder Sichtbarkeit von Variablen



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:

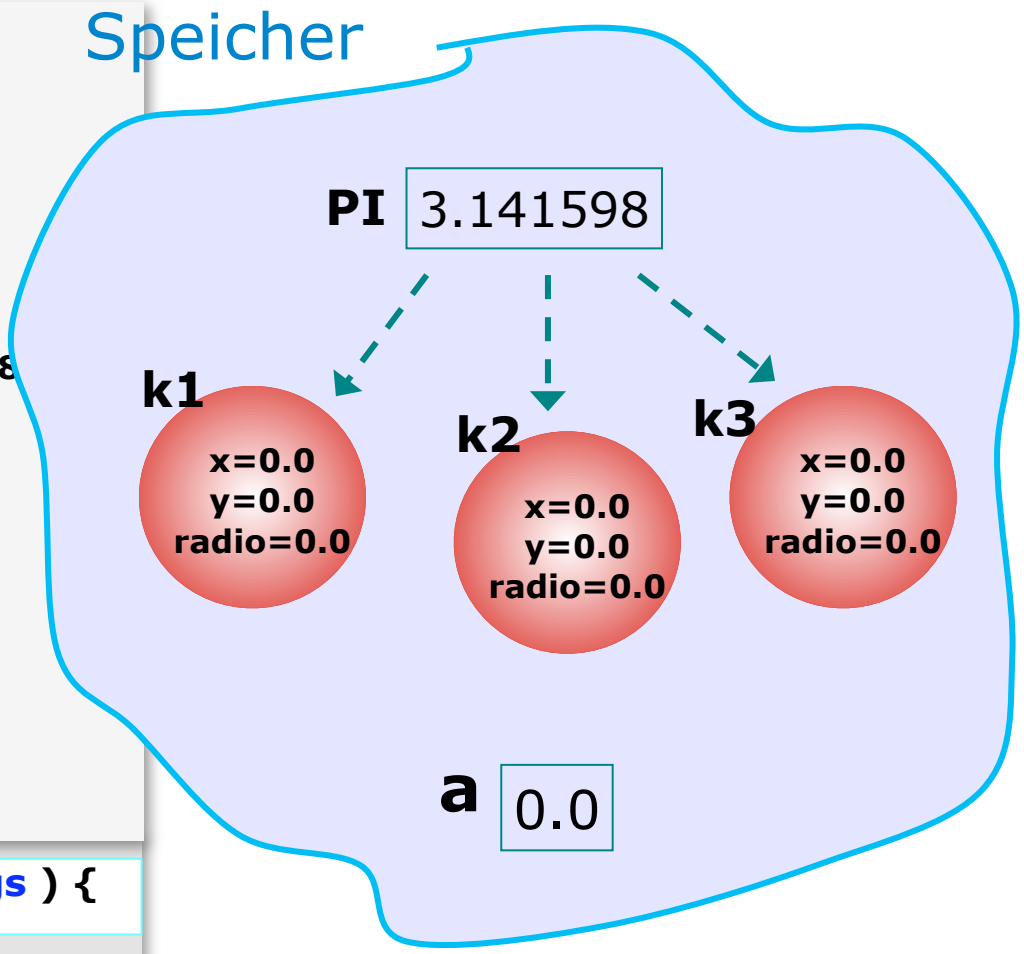
```
public class Kreis {
    // Instanzvariablen
     float x;
     float y;
     float radio;

    // 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 static final float PI = 3.141598
   // 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

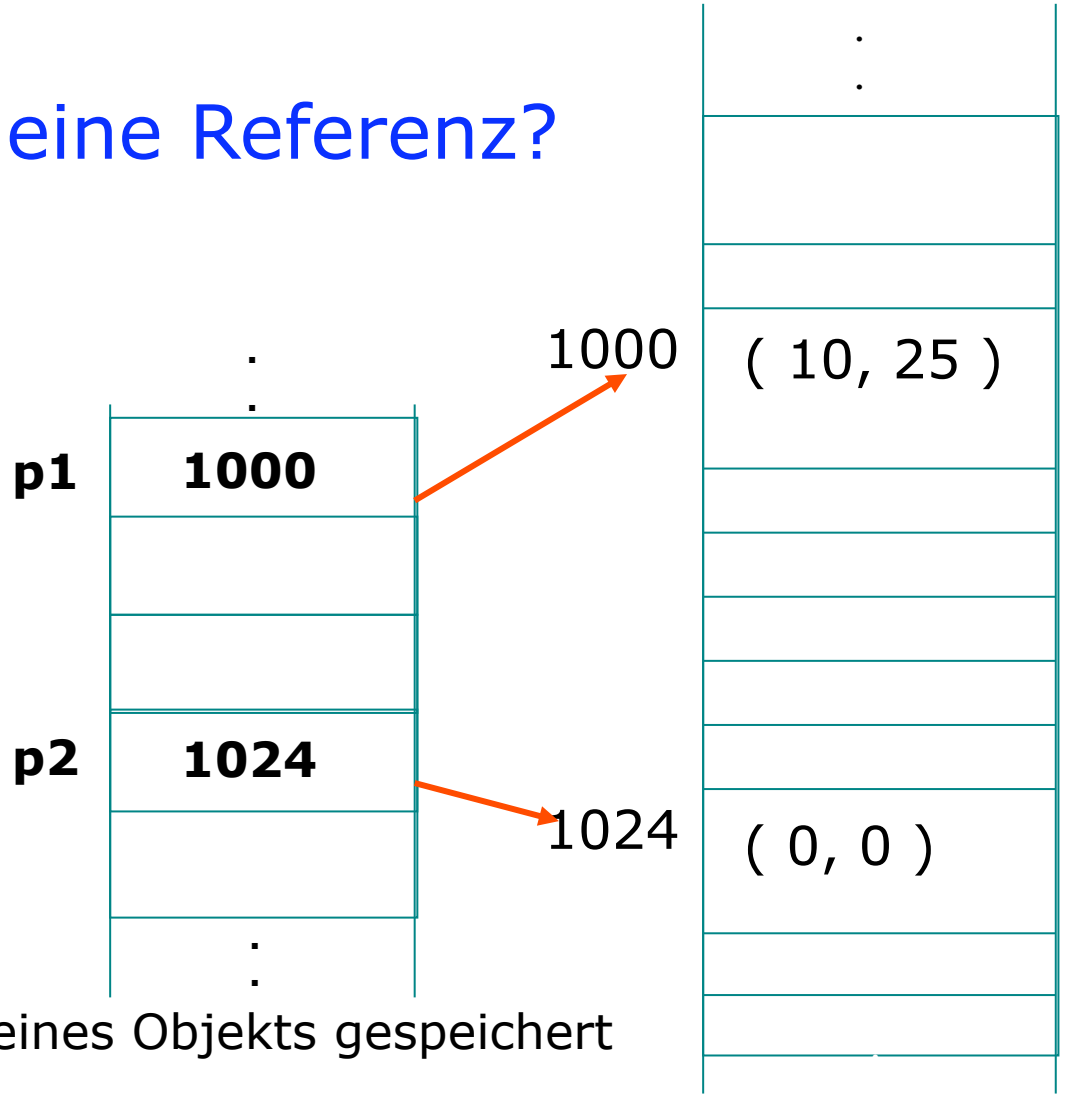
- Lokale Variablen

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

wie in C

Was ist eine Referenz?

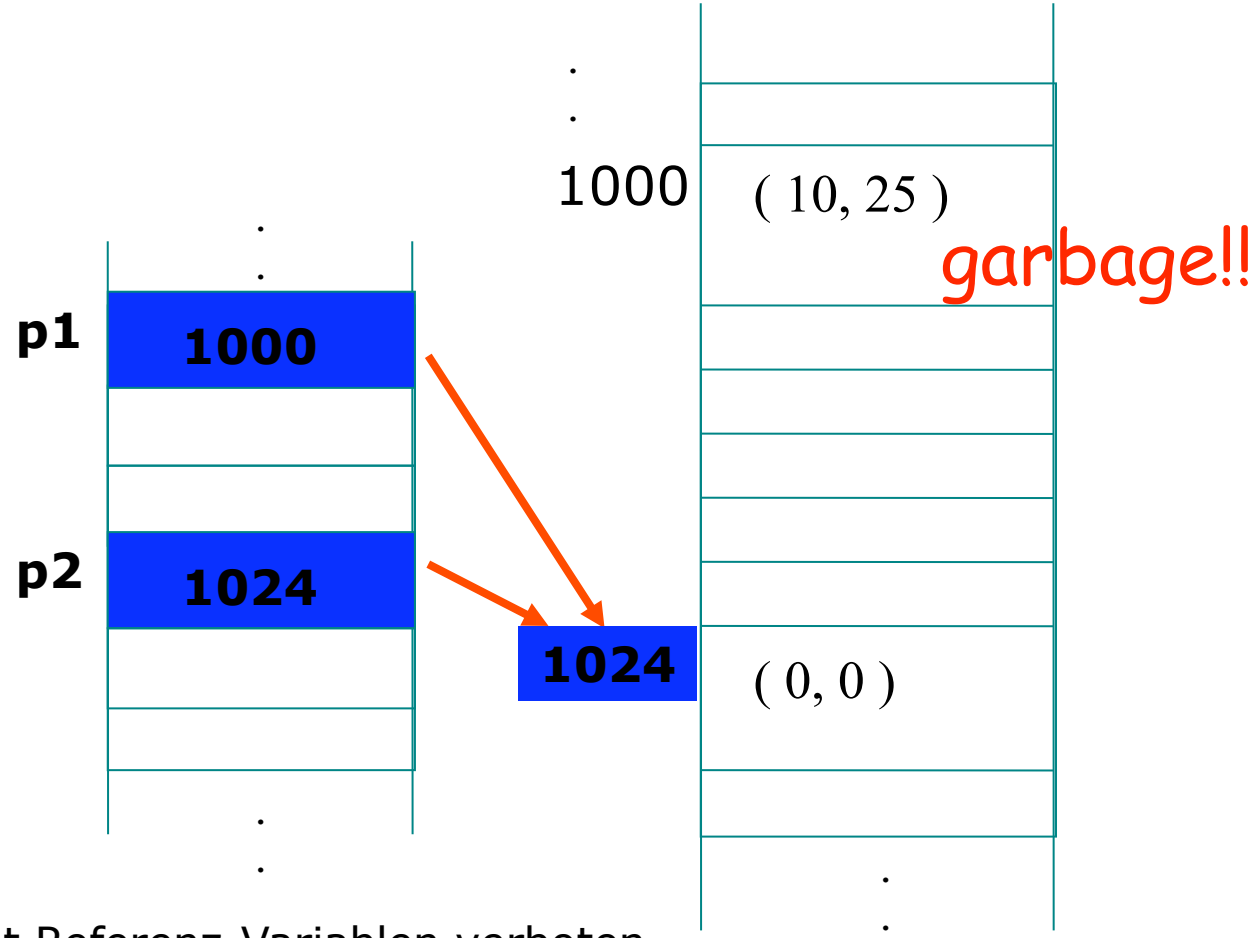
```
Point p1;

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.

Klassenvariablen

Variablen 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-Variablen

der Wert darf nur einmal zugewiesen werden

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:...." );
     }
   ...
```

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:

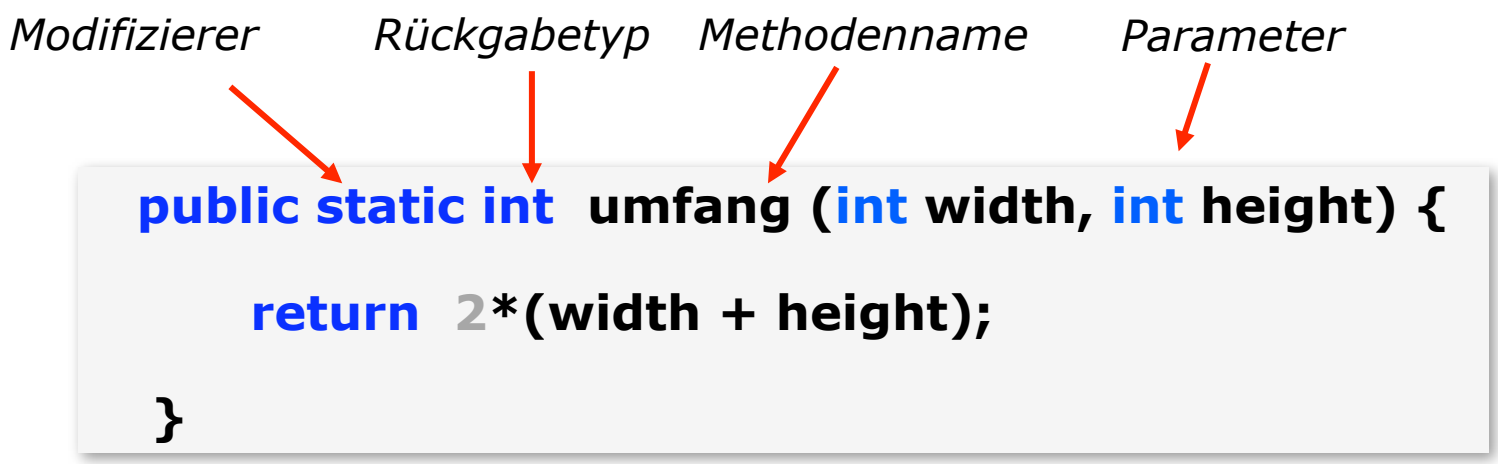
Klassenmethoden

Methoden enthalten den ablauffähigen Programmcode.

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

Methoden dürfen nicht geschachtelt werden.

Beispiel:



Klassenmethoden

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

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

Verwendung innerhalb anderer Klassen

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

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;
        }
}
```

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

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.

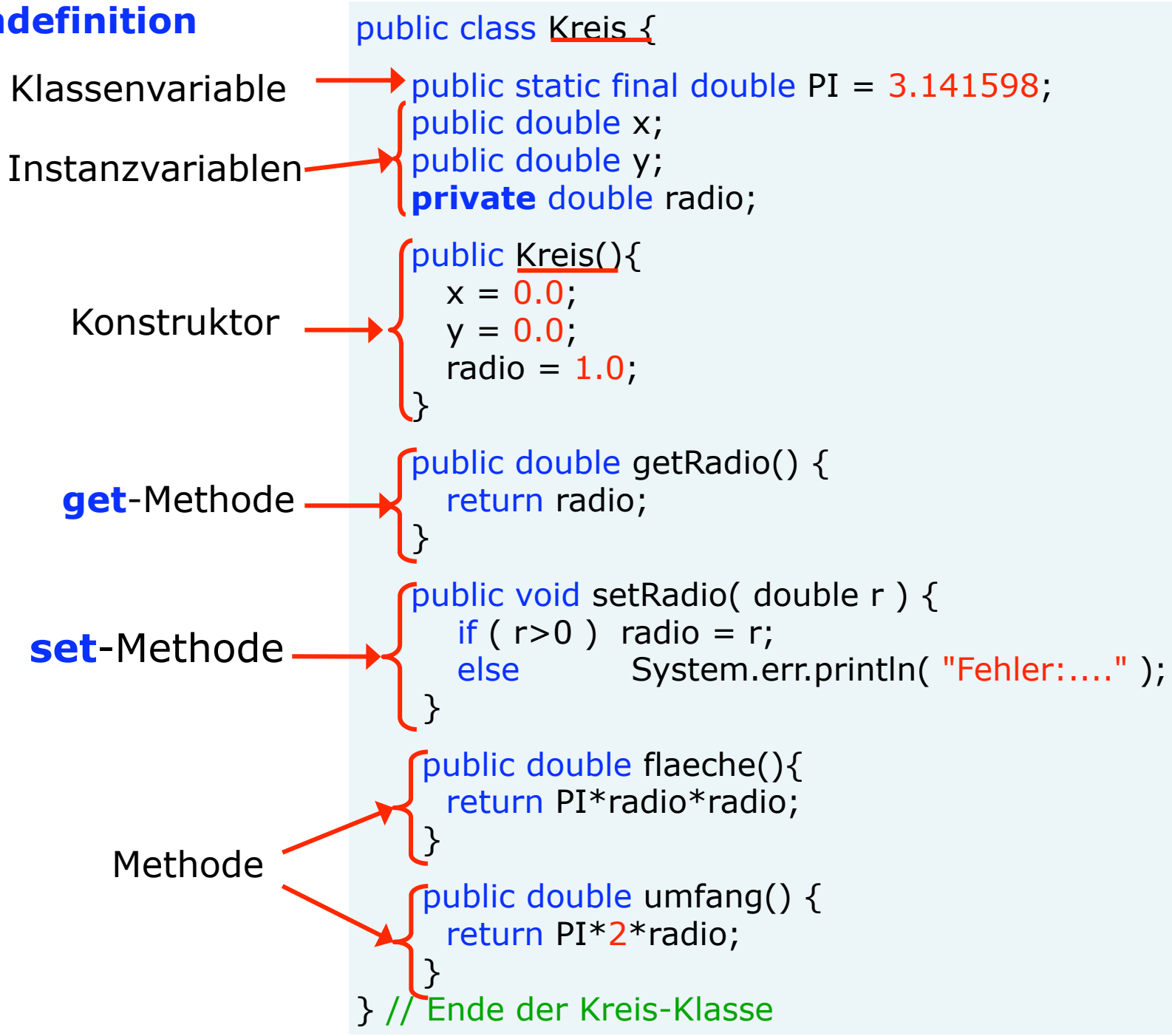
Klassenmethoden dürfen nur andere Klassenvariablen oder lokale Variablen verwenden.

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 **Rectangle**
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!

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, radio;
    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, r ); }
    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

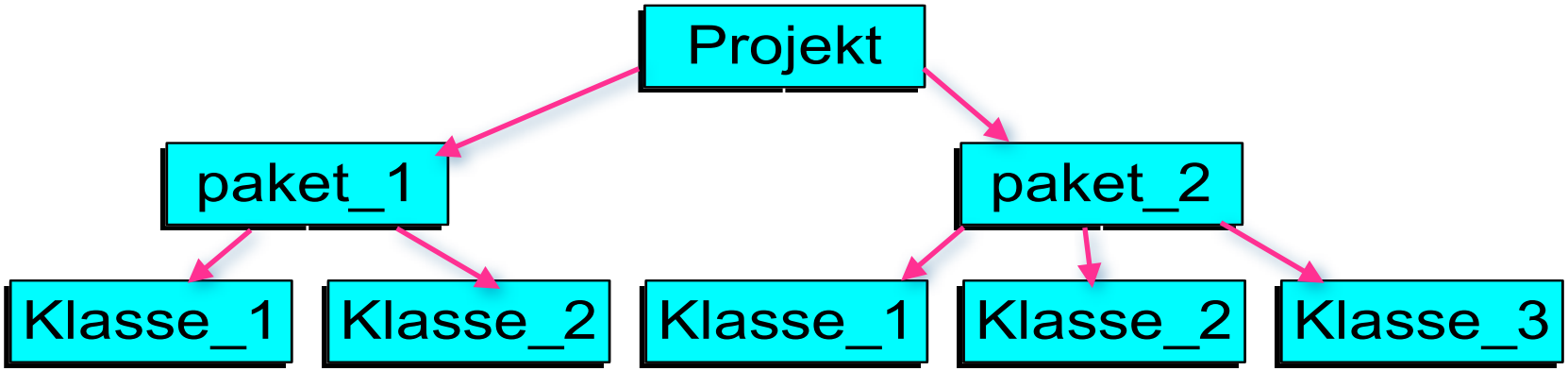
30.0

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.

java-Anwendungen




```
package   paket_1 ;

class Klasse_1 {

   . . . . .

}
```

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.*;
. . .
```

schreibe ich nur → **Button button;**

Packages

```
package beispiele;

import java.lang.*; // Standard

public class Person { ... }
```

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

Zugriffskontrolle auf Methoden

Der Zugriff auf Methoden kann genau so wie im Variablen 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*** (package) zugänglich.