

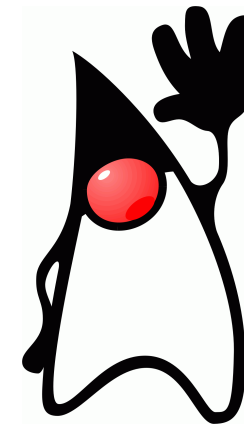
Objektorientiertes Programmieren

(Einführung in Java)

Teil 2



$\{P\} \ S \ \{Q\}$



SoSe 2020

Prof. Dr. Margarita Esponda

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.

Strings

Strings sind in Java kein Basistyp, sondern eine Bibliotheksklasse

`java.lang.String`

`String s = "hallo";` äquivalent zu `String s = new String ("hallo");`

Methoden:

`s.toUpperCase()`

.....▶ "HALLO"

`s.charAt(4)`

.....▶ 'O'

`s.length()`

.....▶ 5

`s.equals(s)`

.....▶ **true**

`s.replace('A', 'Ä')`

.....▶ "HÄLLO"

`s.substring(2, 4)`

.....▶ "LL"

Strings

Strings in Java sind immer **konstant**, d.h. unveränderlich.

Folgende Zuweisungen erzeugen jeweils neue String-Objekte und sind äquivalent.

```
String s = "hallo";
```

s.length() Wert → 5

```
s = s + "Welt!";
```

||

```
s = new String(s + "Welt!");
```

Klassenmethoden

Methoden enthalten den ablauffähigen Programmcode.

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

Methoden dürfen nicht geschachtelt werden.

Beispiel:

Modifizierer *Rückgabetyt* *Methodenname* *Parameter*

```
public static int umfang (int width, int height) {  
    return 2*(width + height);  
}
```

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:

```
public class MyMath {  
    public static long factorial( int n ) {  
        ...  
    }  
}
```

Anwendung innerhalb
anderer Klassen

```
...  
long f = MyMath.factorial( 29 );  
...
```

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.

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.

Klassenvariablen

Variablen haben den Deklarationsspezifizierer **static** oder **final**

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

Beispiel:

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

    // Klassenvariable
    public static final float PI = 3.141598f;

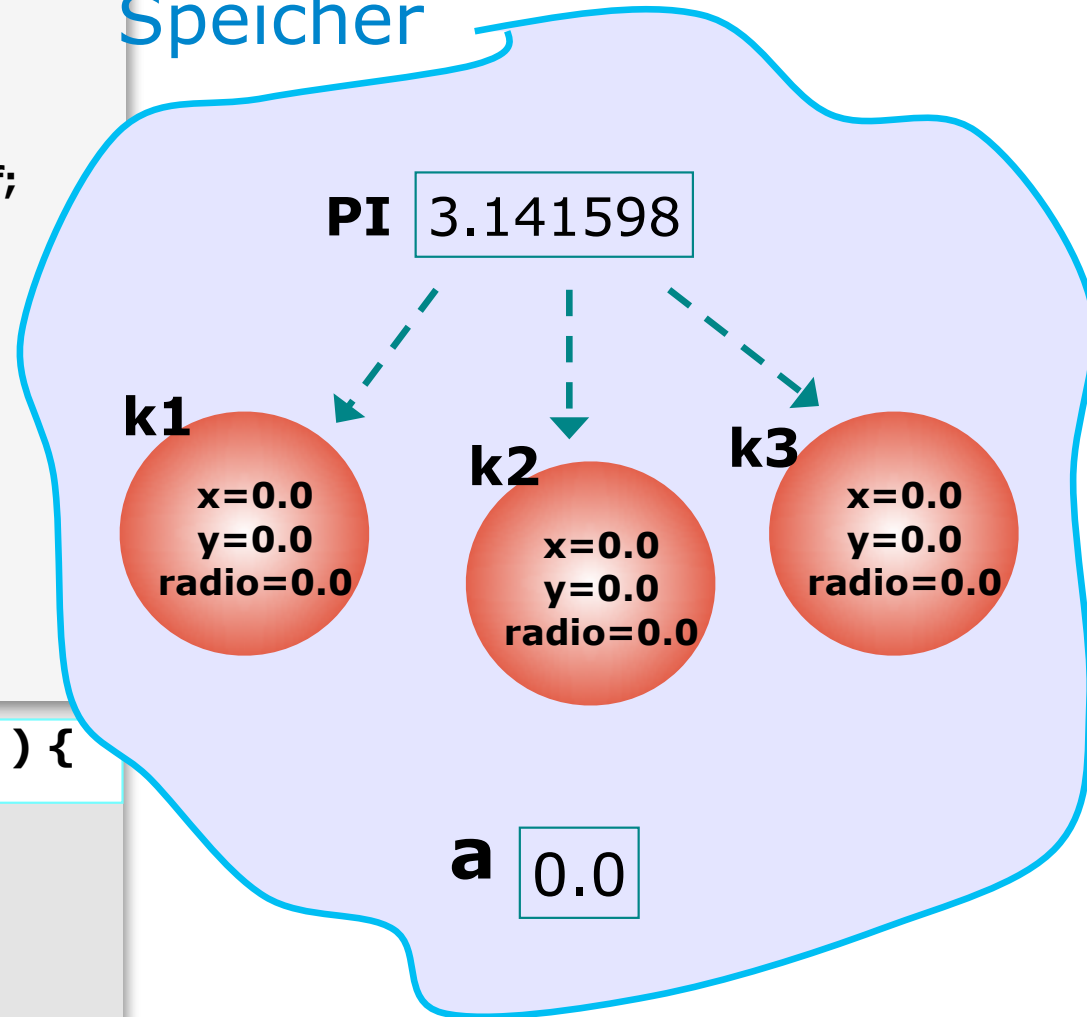
    // Methoden
    public float area() {
        // Lokale Variable
        float a;
        a = PI*radio*radio;
        return a;
    } ...
}
```

Beispiel:

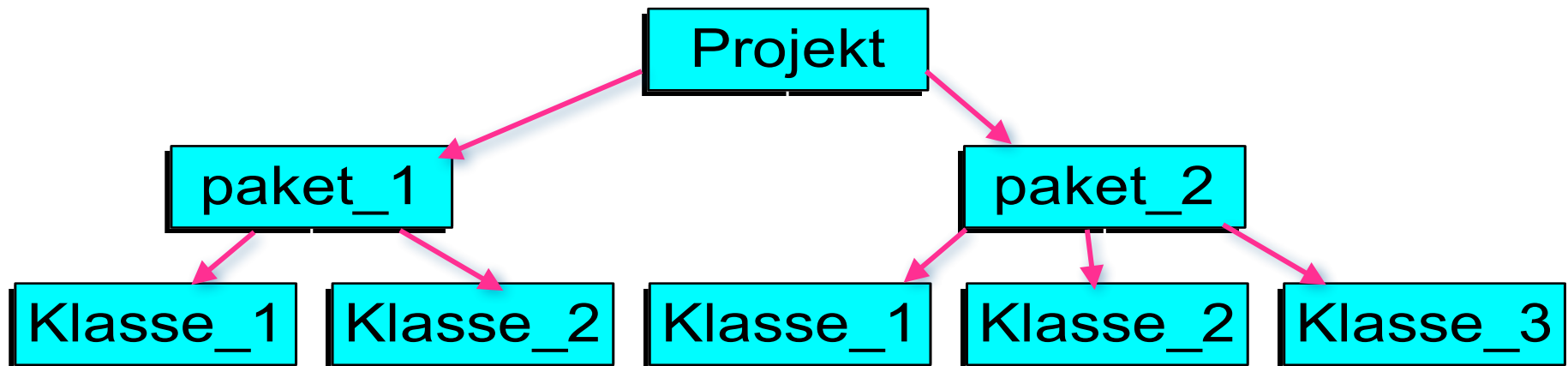
```
public class Kreis {  
    // Instanzvariablen  
    float x;  
    float y;  
    float radio;  
  
    // Klassenvariable  
    public static final float PI = 3.141598f;  
  
    // Methoden  
    public float area() {  
        // Lokale Variable  
        float a;  
        a = PI*radio*radio;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

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

Speicher



java-Anwendungen



```
packet packet_1 ;  
public class Klasse_1 {  
    . . .  
}
```


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

```
package beispiele;  
import java.lang.*; // Standard  
public class Person { ... }
```



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


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

schreibe ich nur → **Button** button;

```
import java.awt.*;
```

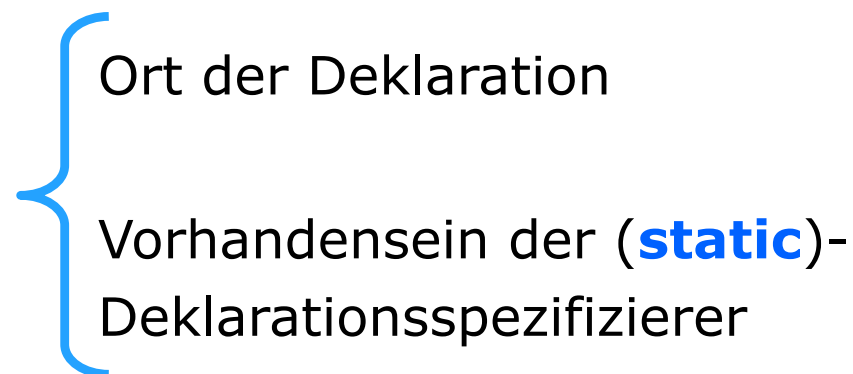
```
. . .
```

```
Button button;
```

Variablen in Java



Diese Klassifikation richtet sich nach folgenden zwei Aspekten:



Variablen in Java

Lokale Variablen

Hilfsvariable für Berechnungen

Sie werden innerhalb von Methoden deklariert.

Lebenszeit: nur solange die Methode ausgeführt wird.

Instanzvariablen

(Feldvariablen,
Attribute)

Variablen, in denen die Eigenschaften
von Objekten gespeichert werden

Lebenszeit: nur solange das Objekt existiert.

Klassenvariablen

Variablen, die zu einer Klasse gehören

Lebenszeit: solange das Programm ausgeführt wird.

Was ist eine Referenz?

Point p1;
Point p2;
p1 = new Point (10, 25);
p2 = new Point (0, 0);

Speicher für
Variablen mit
fester Größe

p1

1000

p2

1024

1000

(10, 25)

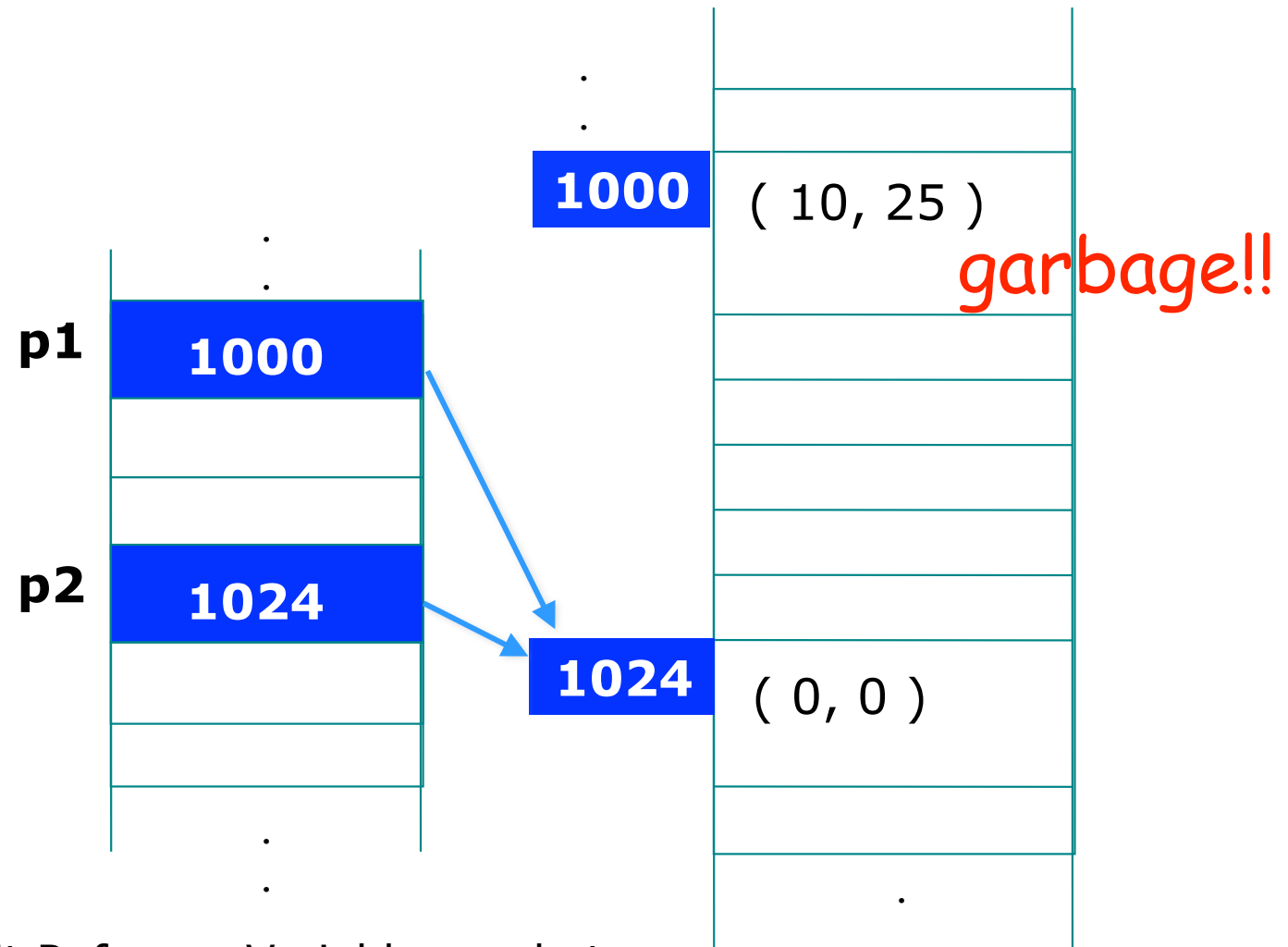
1024

(0, 0)

sind Variablen, wo die Adresse eines Objekts gespeichert wird

Referenz-Variablen

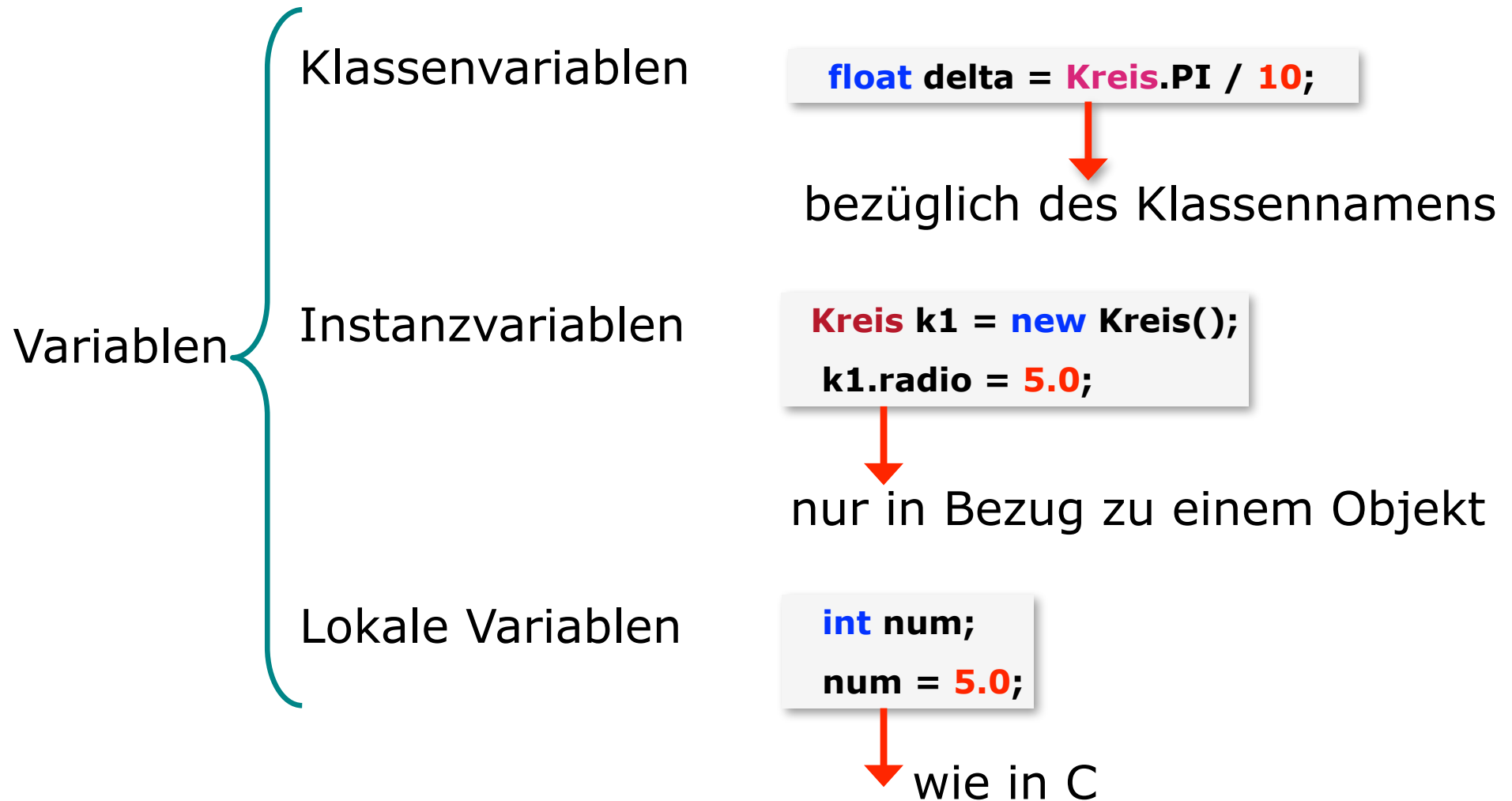
`p1 = p2 ;`



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

Variablen in Java

Zugriff



Sichtbarkeit von Java-Variablen

Zugriffsangabe
oder
Sichtbarkeit

	Klasse	Unterklassen	Paket	Welt
privat	✓			
package	✓		✓	
protected	✓	✓	✓	
public	✓	✓	✓	✓

Kein Modifikator ist äquivalent zur **package**

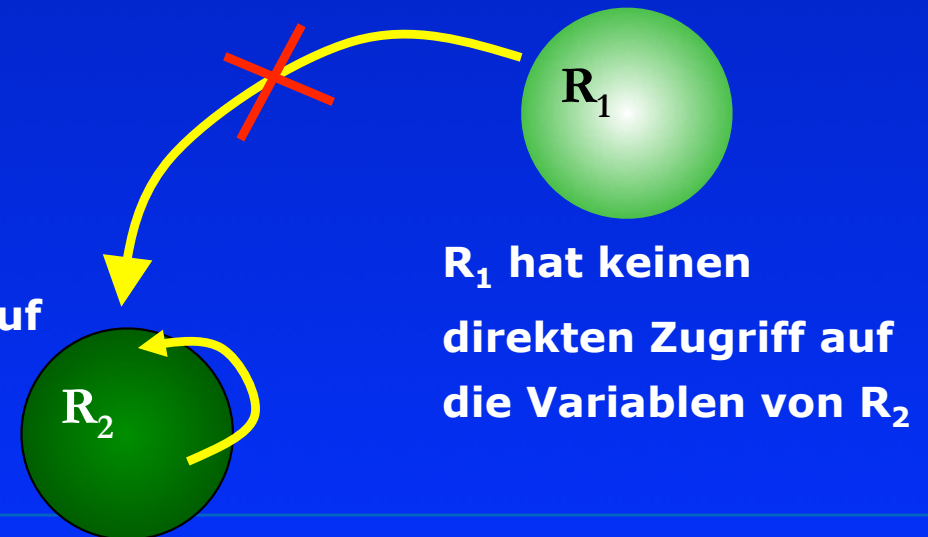
Sichtbarkeit von **private**-Variablen

Rechteck-Klasse

```
public class Rectangle {  
  
    private int x;  
    private int y;  
    private int width;  
    private int height;  
  
    /* Methoden */  
    public int area() {  
        return width*height;  
    }  
}
```

Zugriff nur innerhalb
der Rechteck-Klasse

R₂ hat Zugriff auf
seine eigenen
Variablen.



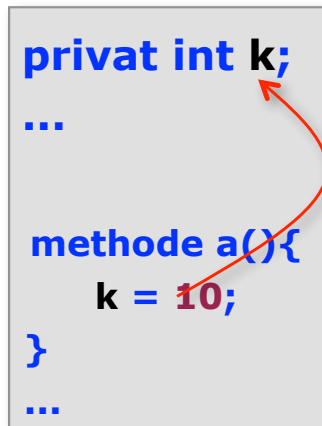
Zugriffsangabe oder Sichtbarkeit von Variablen

Klasse

```

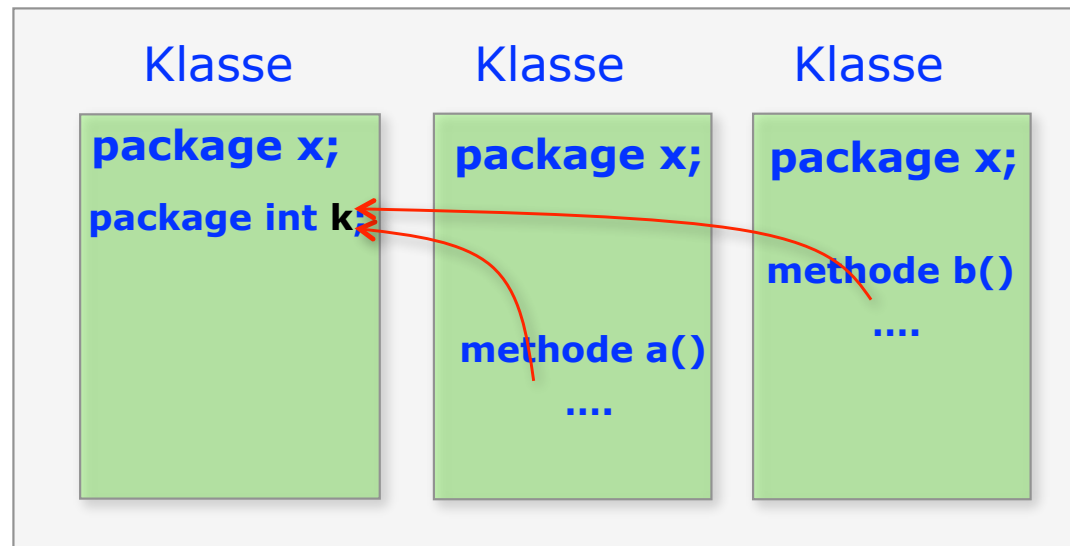
privat int k;
...

methode a(){
    k = 10;
}
...
    
```



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

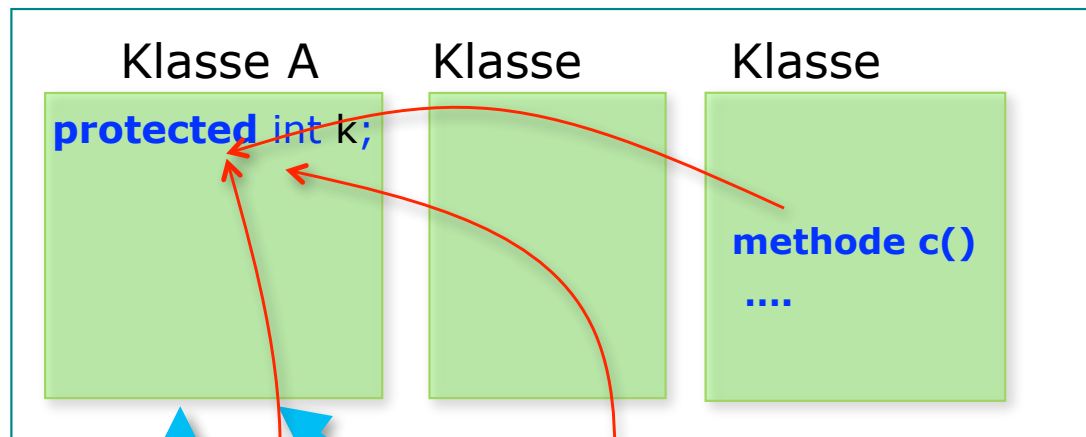
Paket- oder Verzeichnis-x



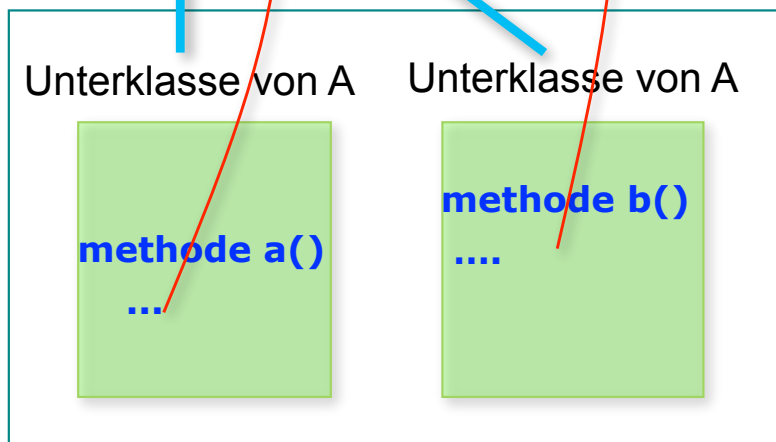
Alle Objekte innerhalb eines Verzeichnisses haben direkten Zugriff auf eine `package`-Variable.

Zugriffsangabe oder Sichtbarkeit von Variablen

Paket oder Verzeichnis



Paket oder Verzeichnis



Zugriff innerhalb der
Klassen-Hierarchie

Zugriffsangabe oder Sichtbarkeit von Variablen

Paket oder Verzeichnis

Klasse

public

Klasse

Klasse

Paket oder Verzeichnis

Klasse

Klasse

Paket oder Verzeichnis

Klasse

Klasse

Paket oder Verzeichnis

Klasse

Klasse

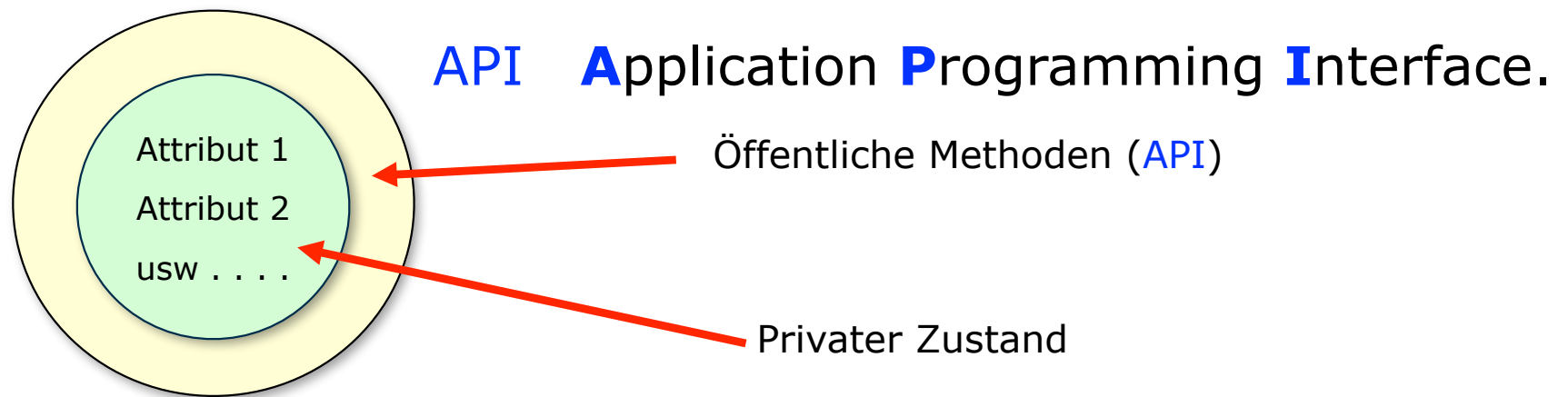
Klasse

Die Welt aller Java-Klassen

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.



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

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

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;
    }
}
. . .

```

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:

Definition der setTime-Methode

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

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

    ...
}
```


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

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!**

Weitere vordefinierte Operationen, die mit Referenz-Variablen erlaubt sind

(Typ)_Operator

```
Button button = new Button("Klick me");  
Object obj = (Object) button;
```

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 ;
```

Typ-Operatoren

unär

Operator	Zeichen	Rtg.	Beispiel	Priorität
"cast"-Operator	(<i>typ</i>)		(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

Klassendefinition

```
public class Kreis {  
    Klassenvariable → public static final double PI = 3.141598;  
    Instanzvariablen → {  
        public double x;  
        public double y;  
        private double radio;  
    }  
    Konstruktor → {  
        public Kreis() {  
            x = 0.0;  
            y = 0.0;  
            radio = 1.0;  
        }  
    }  
    get-Methode → {  
        public double getRadio() {  
            return radio;  
        }  
    }  
    set-Methode → {  
        public void setRadio( double r ) {  
            if ( r>0 ) radio = r;  
            else      System.err.println( "Fehler:...." );  
        }  
    }  
    Methode → {  
        public double flaeche(){  
            return PI*radio*radio;  
        }  
    }  
    Methode → {  
        public double umfang() {  
            return PI*2*radio;  
        }  
    }  
} // Ende der Kreis-Klasse
```

Java Konventionen

Motivation:

Java-Code-Konventionen sind aus vielen Gründen sehr wichtig.

Konventionen wurden von erfahrenen Entwicklern konzipiert, um die Kosten der Softwareentwicklung zu senken.

Konventionen verbessern deutlich die Lesbarkeit und das Verständnis von Softwaresystemen.

- a) von den **Entwicklern** selber.
- b) von anderen **Mitgliedern des Teams**.
- c) für zukünftige **Wartung**.

Offizieller Link für Code-Konventionen von Sun-Microsystems.

<http://java.sun.com/docs/codeconv/>

Namenskonventionen in Java

Klassen und Schnittstellen	Klassennamen sollen Namenswörter sein und der erste Buchstabe soll groß geschrieben werden.	<code>class</code> Circle <code>Interface</code> Shape
Methoden	Methoden sind Aktionen oder Operationen und sollen Verben sein. Der erste Buchstabe soll klein geschrieben werden.	<code>paint()</code> <code>draw()</code> <code>run()</code> <code>getColor()</code>
Instanzvariablen	Der erste Buchstabe soll klein geschrieben werden. Der Name soll aussagekräftig über ihren Inhalt sein. Wenn mehrere Worte benutzt werden, sollen Großbuchstaben dazwischen geschrieben werden.	<code>maxDistance</code> <code>width</code> <code>breite</code>
Konstanten	Alle Buchstaben sollen groß geschrieben werden. Wenn der Name aus mehreren Worten besteht, sollen '_' dazwischen stehen.	<code>MAX_DISTANCE</code> <code>WEST</code>

Weitere Konventionen

Variablennamen beginnen mit Kleinbuchstaben
myName

Klassennamen beginnen mit Großbuchstaben
Rechteck

Konstante
Klassenvariablen nur Großbuchstaben
BLAU

Methoden beginnen mit Kleinbuchstaben
setColor (BLAU)

Layout

Einrücken

Rücken Sie den Code innerhalb neuer Blöcke mindestens um 4 Leerzeichen ein.

Zeilenlänge

Vermeiden Sie Zeilen, die länger als **80** Zeichen sind. Gebrochene längere Zeilen machen gedruckte Programme schwierig zu lesen.

Vermeiden Sie lange Ausdrücke, und unterbrechen Sie diese vor einem Operator oder Komma, oder beginnen Sie den Ausdruck in einer neuen Zeile.

Sehr leicht automatisierbar in modernen Programmierumgebungen!

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:

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!

Konstruktor

"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;
    }
}
```

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.

Aufzählung-Datentyp

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

Der Aufzählungsdattentyp 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;
    };
}
```

Die Human-Klasse in Java

Eigenschaften →

Konstruktor →

Methoden →

```
public class Human {
    public enum Gender { MALE, FEMALE };
    public enum Language { GERMAN, SPANISH, ENGLISH };
    public enum Mood { HAPPY, UNGRY, SAD };

    private String name;
    private Gender gender;
    public Language language;
    private Mood mood;

    public Human(String n, Gender g){
        name = n;
        gender = g;
    }

    public String getName(){
        return name;
    }

    public String getGender(){
        return gender;
    }
}
```

Parameterübergabe in Java

Alle Parameter werden in Java per Wert (**by-value**) übergeben.

Veränderungen der Parameter innerhalb der Methode bleiben **lokal**.

Die formalen Parameter einer Methode-Definition sind **Platzhalter**.

Beim Aufruf der Methode werden die **formalen Parameter** durch **reale Variablen** ersetzt, die den gleichen Typ der formalen Parameter haben müssen.

```
...  
int a = 5 ;  
Point p1 = new Point ( 1, 2 );  
g.methodeAufruf ( a, p1 );  
...
```

Nach Beendigung des Methoden-Aufrufs haben sich der Wert von **a** sowie der Referenz-Wert **p1** nicht geändert.

```
public class Parameteruebergabe {

    public static int fakultaet ( int n ) {
        int fac = 1;
        if (n>1)
            while (n>1) {
                fac = fac*n;
                n--;
            }
        return fac;
    }

    public static void main ( String[] args ) {
        int n = 20;
        System.out.println ( "n=" + n );
        fakultaet ( n );
        System.out.println( "n=" + n);
    }
}
```

Parameterübergabe mit primitiven Datentypen

Die Variable **n** wird solange verändert, bis sie gleich **0** wird.

Nur eine Kopie des Parameterwertes wird übergeben.

Nach Beendigung des Methoden-Aufrufs hat sich der Wert von **n** nicht geändert.

n=20

n=20

Parameterübergabe mit Referenz-Variablen (Objekte)

```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height );
    }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

r1: 152

152

10

20

30

30

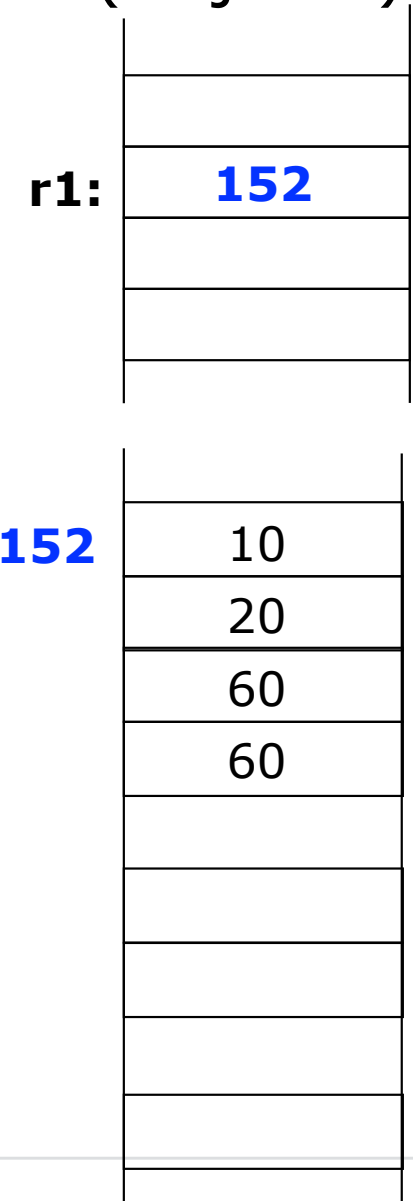
Parameterübergabe mit Referenz-Variablen (Objekte)

```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height );
    }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```



Parameterübergabe mit Referenz-Variablen (Objekte)

```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height );
    }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

r1:	152
r2:	136

152	10
	20
	60
	60
136	10
	20
	60
	60

Parameterübergabe mit Referenz-Variablen (Objekte)

```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height );
    }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

Nach Beendigung des Methoden-Aufrufs hat sich der Referenz-Wert **r1** nicht verändert.

r1:	152
r2:	136
152	10
	20
	60
	60
136	10
	20
	60
	60

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!

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;
    }
    . . .
}
```