



Designing Education  
Connecting People

## Das erwartet Sie:

- OO-Konzepte
- Java
  - Klassen/Objekte
  - Attribute
  - Methoden

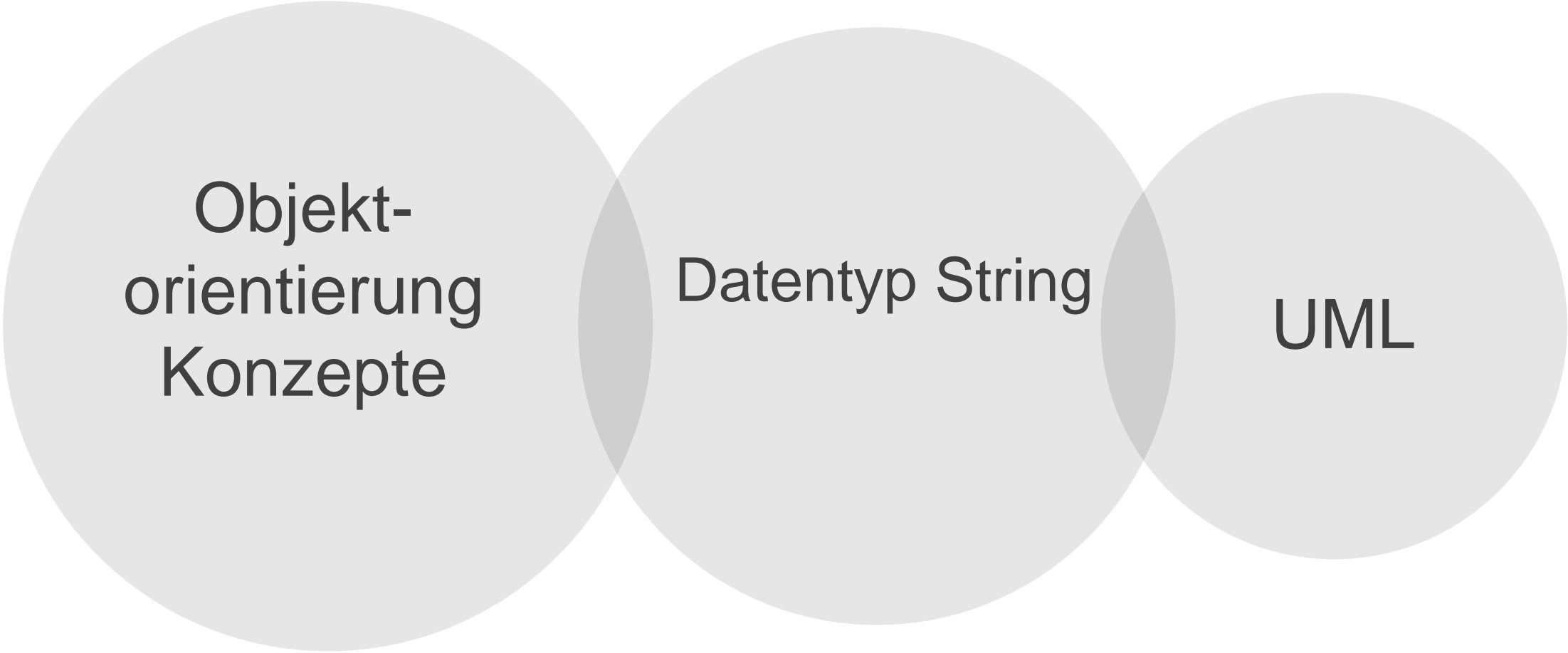


## Funktionalität in Anwendungen realisieren

Lernfeld 11a

# Überblick

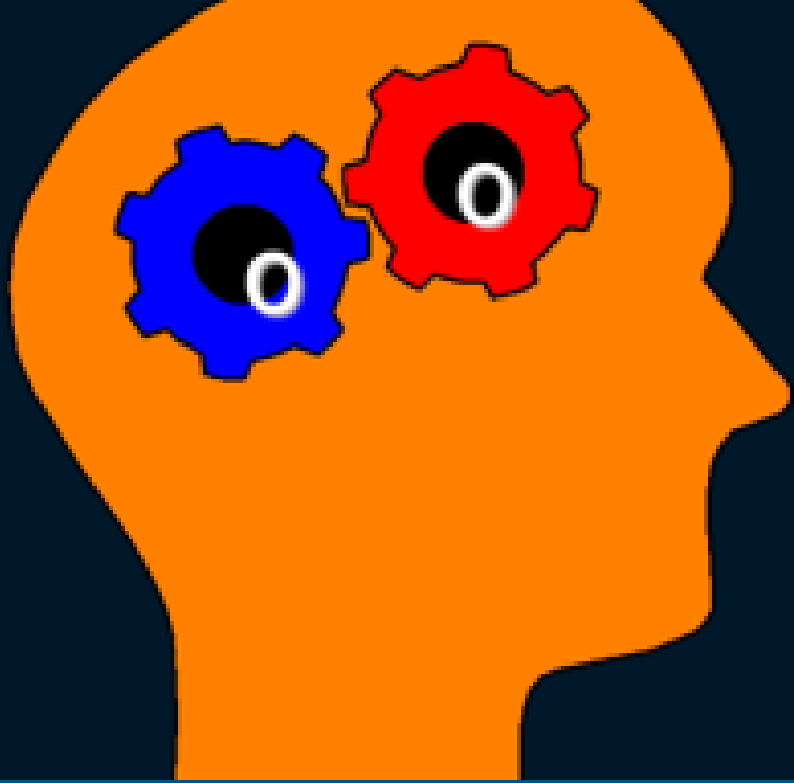
Java



Objekt-  
orientierung  
Konzepte

Datentyp String

UML

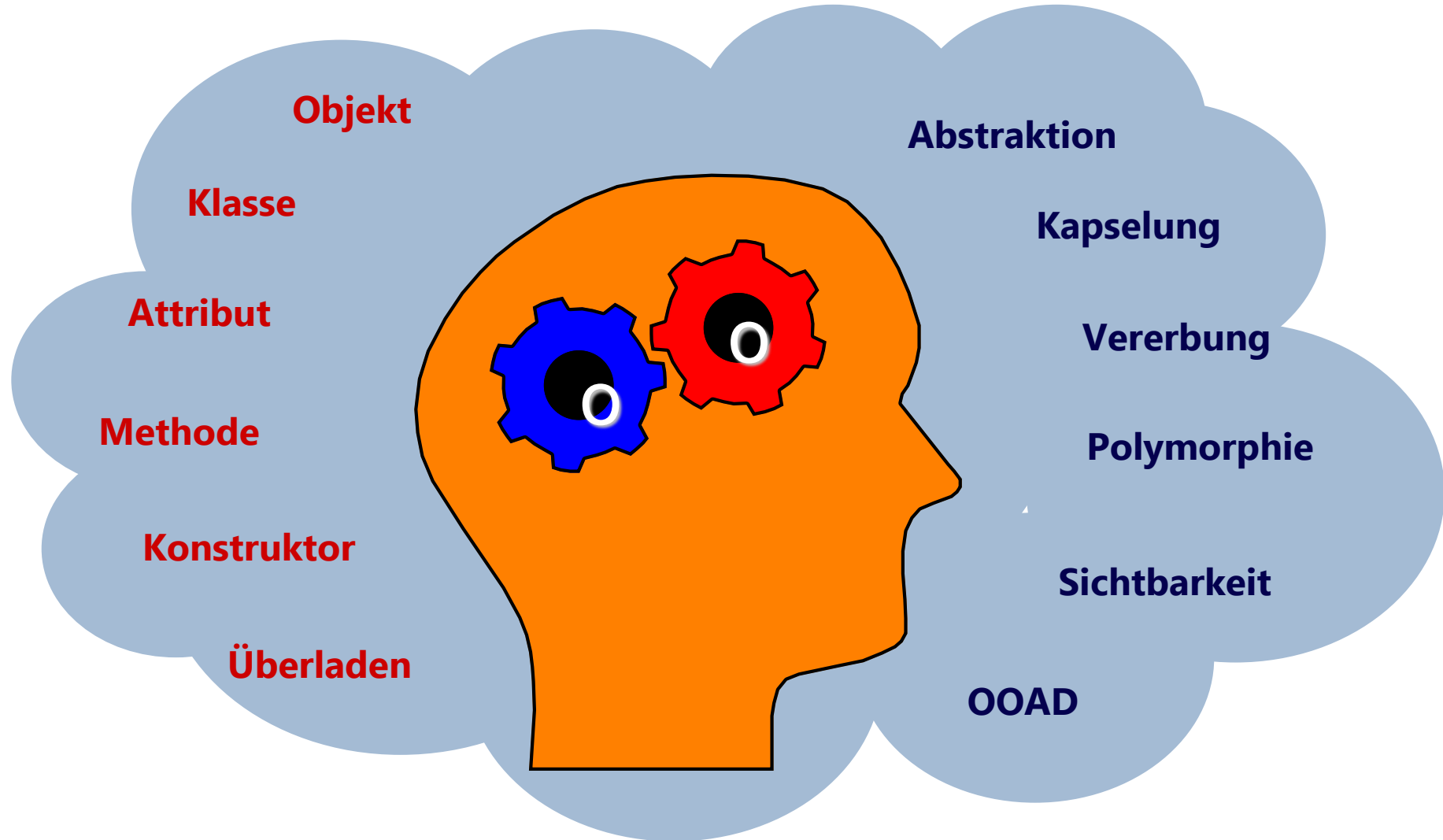


OO-Konzepte

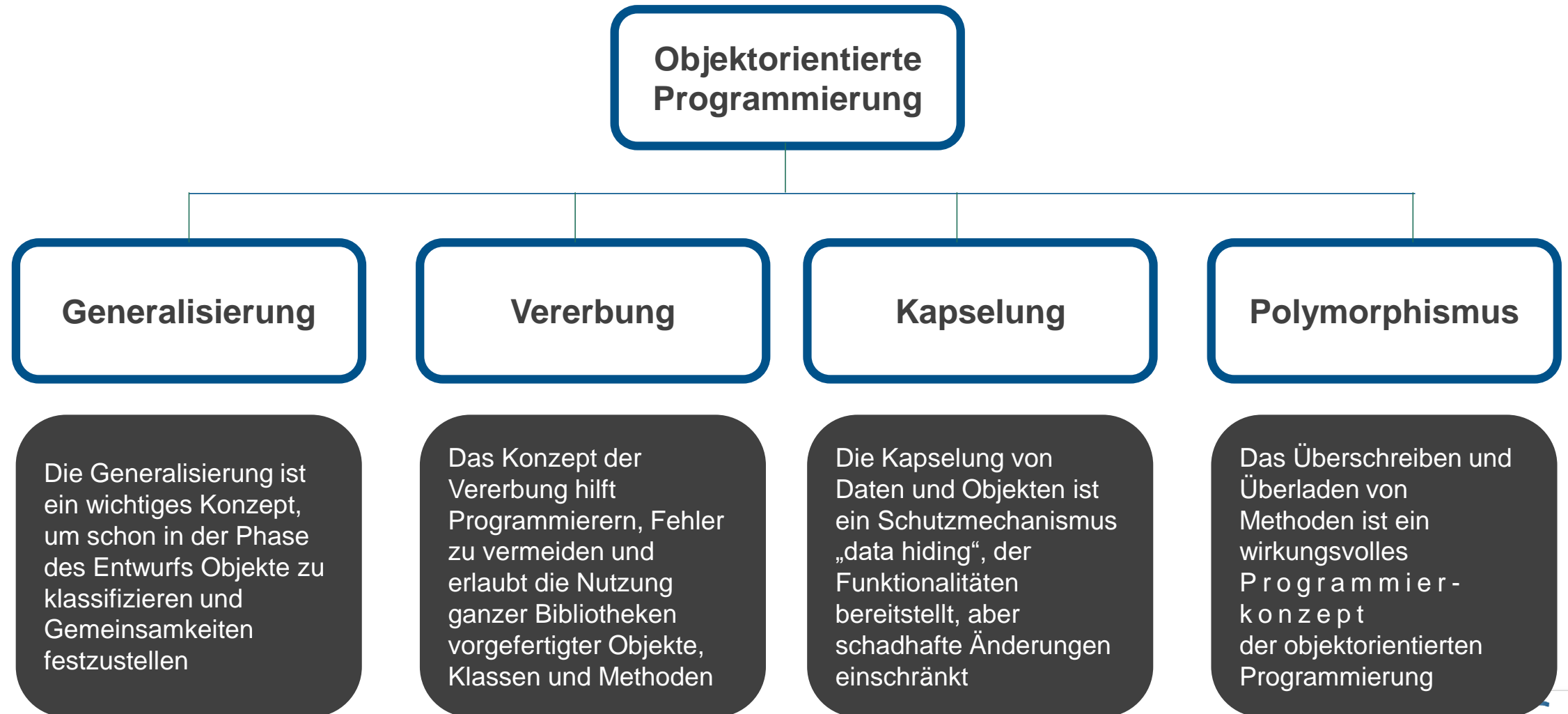
# Lernziel

Objektorientierte  
Konzepte verstehen

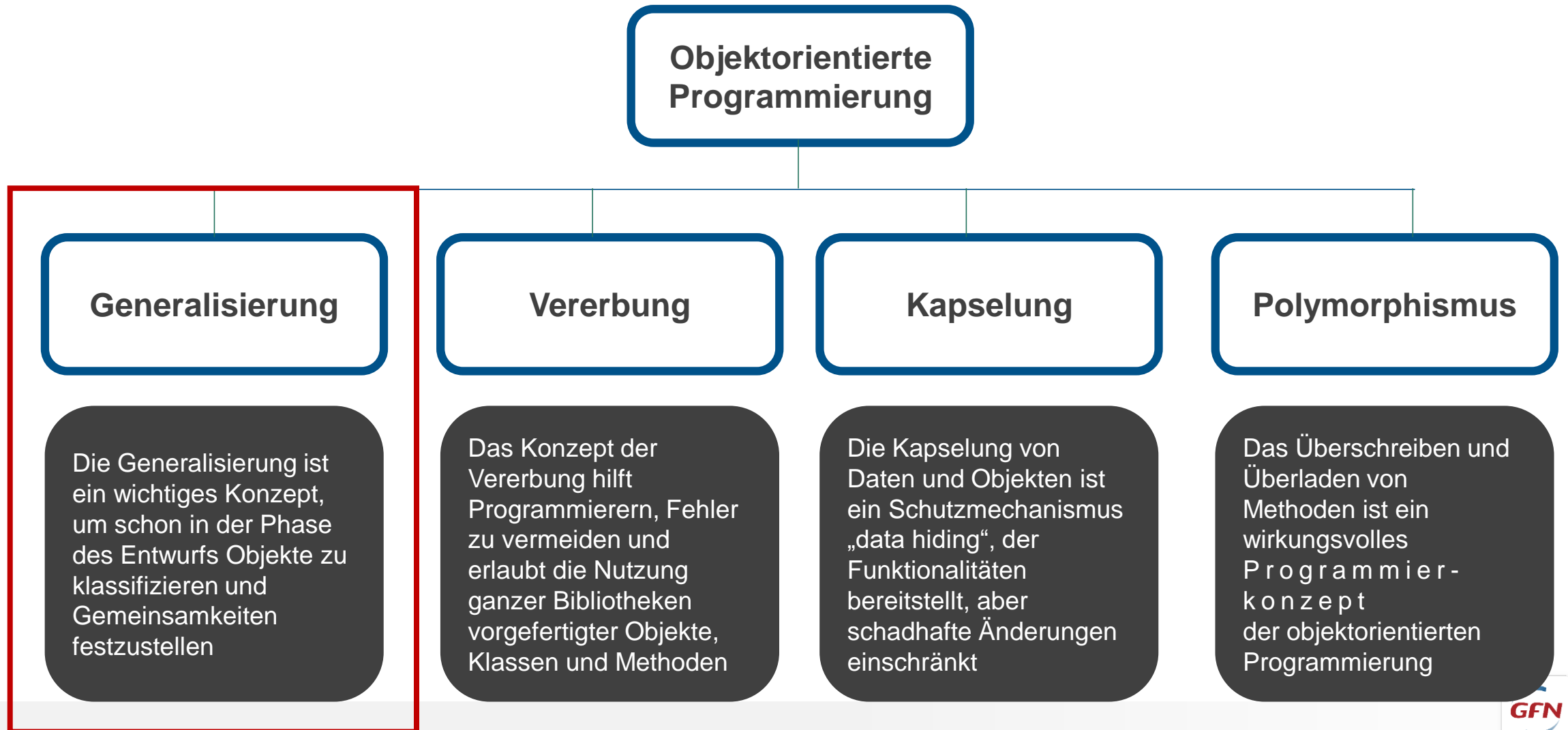
# Objektorientierung



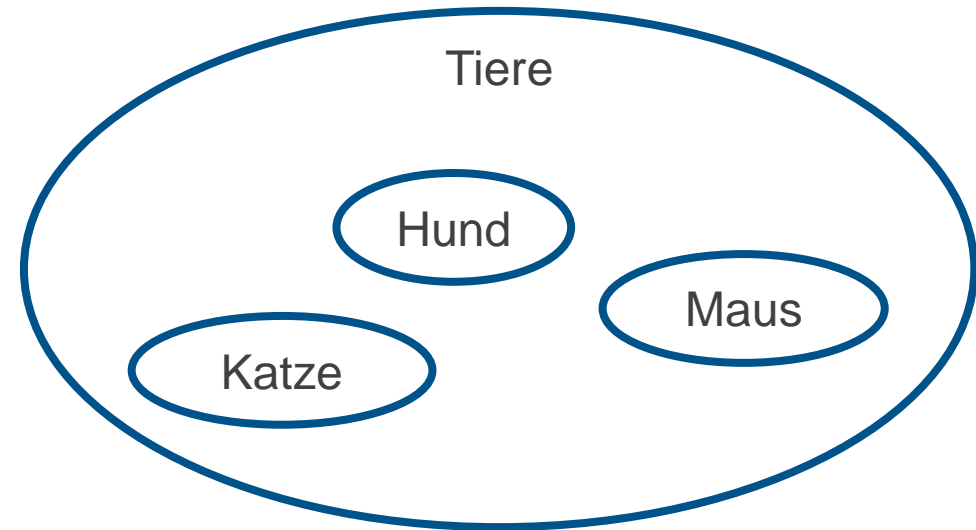
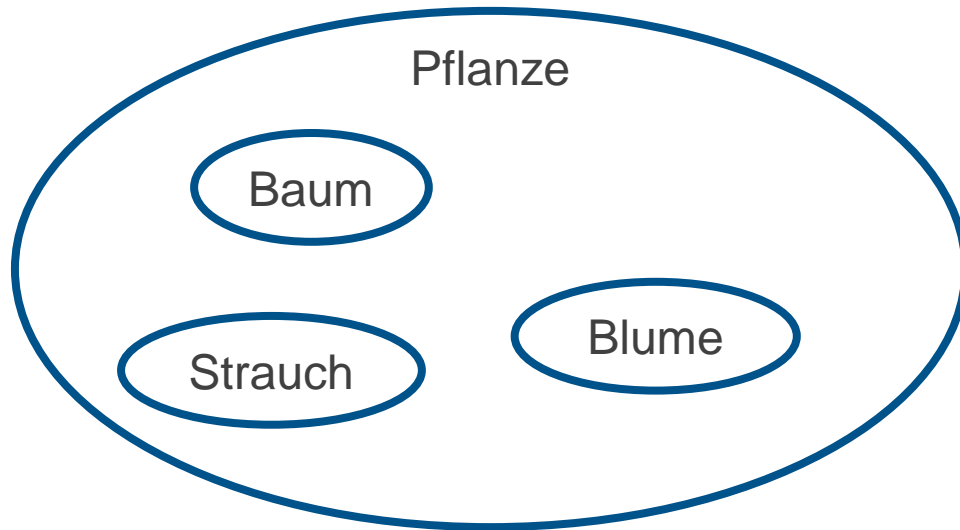
# Objektorientierung-Konzepte



# Objektorientierung-Konzepte



# Klassen - generalisieren



# ... von Objekten und Klassen

Objekte der realen Welt



Abstraktion

= Klasse

Tier
Attribute : Datentyp
Methoden()

Schablone  
Bauanleitung  
Vorlage



# Klassen

- Klassen beschreiben die **Gemeinsamkeiten** von **Objekten**  
(Beschreibung durch Abstraktionsprozess, generalisieren)
- Klassen dienen als eine Art **Schablone**, die benutzt werden kann, **tatsächliche Objekte** zu erzeugen
- Eine Klasse kann einerseits **Bauplan für konkrete Objekte** sein, die im Programmablauf je nach Bedarf erzeugt und mit der Ausführung bestimmter Methoden beauftragt werden
- Eine Klasse kann andererseits aber auch **Akteur** sein  
(Methoden einer anderen Klasse ausführen und aufrufen)

# Klassen und Objekte

- **Objekte** sind die Elemente, die ein Programmierer verwendet (vergleichbar mit **Variablen**)
- **Objekte** unterscheiden sich durch **unterschiedliche Zustände**



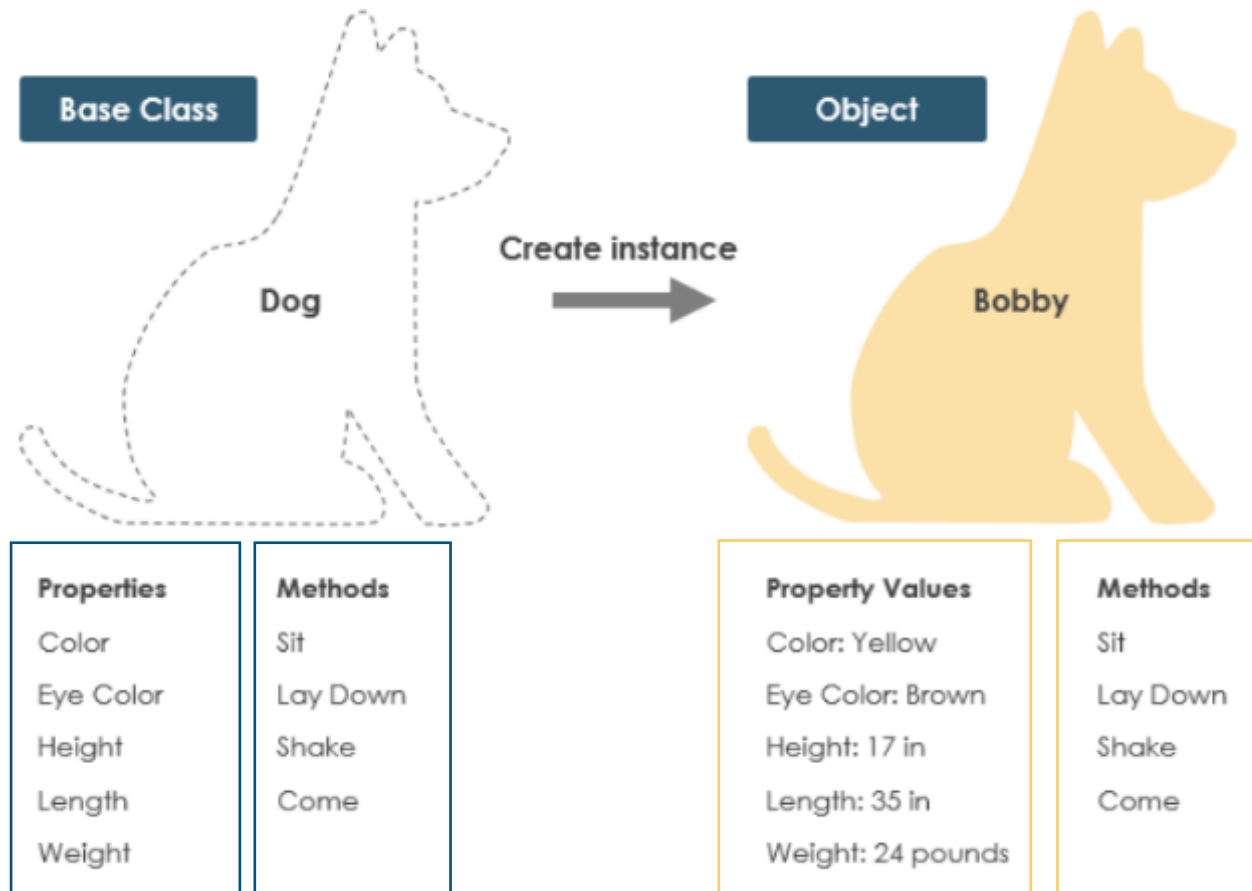
## Objekt

hat eine Menge von Eigenschaften (Felder),  
die besagen, wie das Objekt gestaltet ist

hat ein wohldefiniertes Verhalten, Handlungs-  
kompetenzen (Methoden)

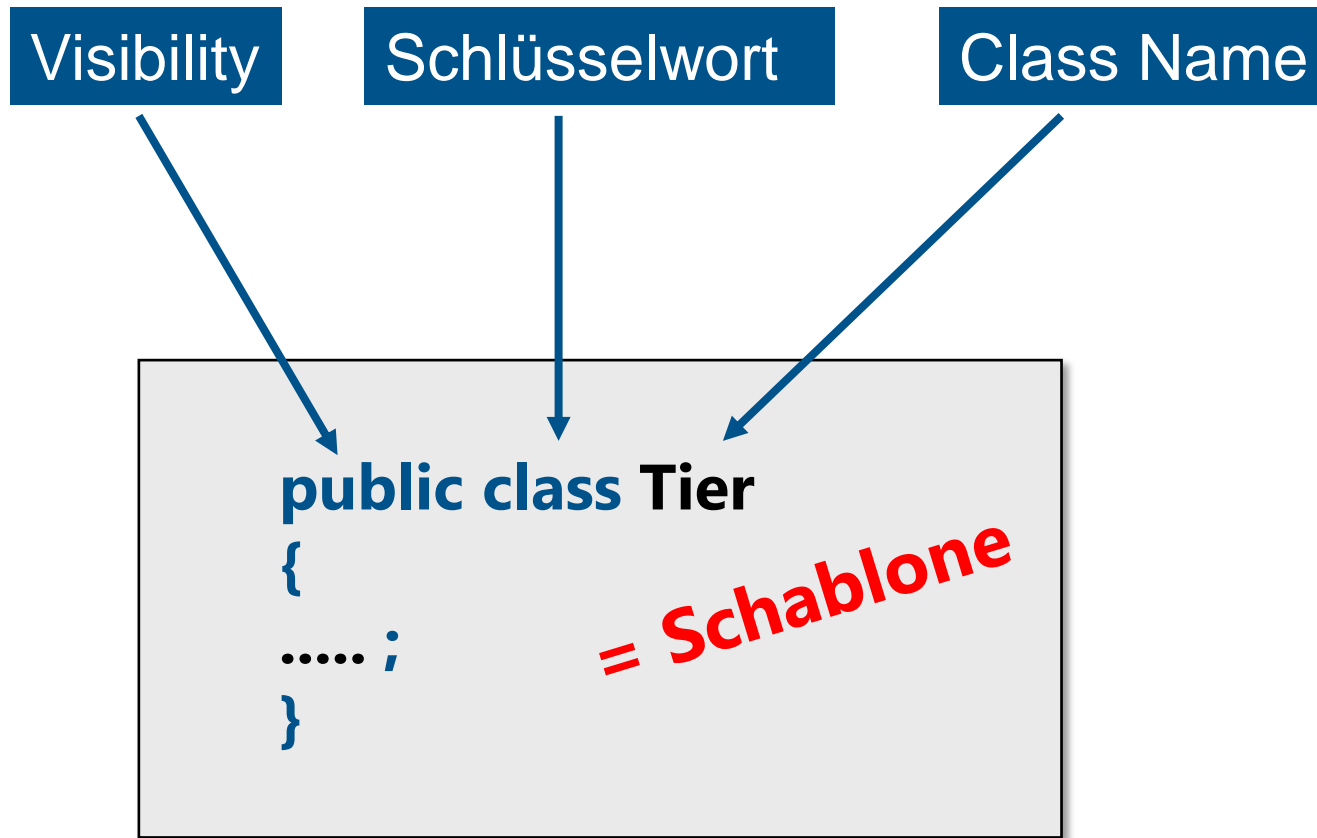
# Klassen und Objekte

Ein Hund hat Zustände (Felder) – Farbe, Name, Rasse  
sowie Verhaltensweisen (Methoden) – Wedeln, Bellen, Essen



Quelle: [www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/](http://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/)

# Definieren von Klassen



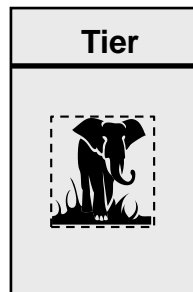
# Klassen und Objekte (Instanzen)

## Schablone

Felder

```
public class Tier
{
    double groesse;
    double gewicht;
    String name;
    String art;
}
```

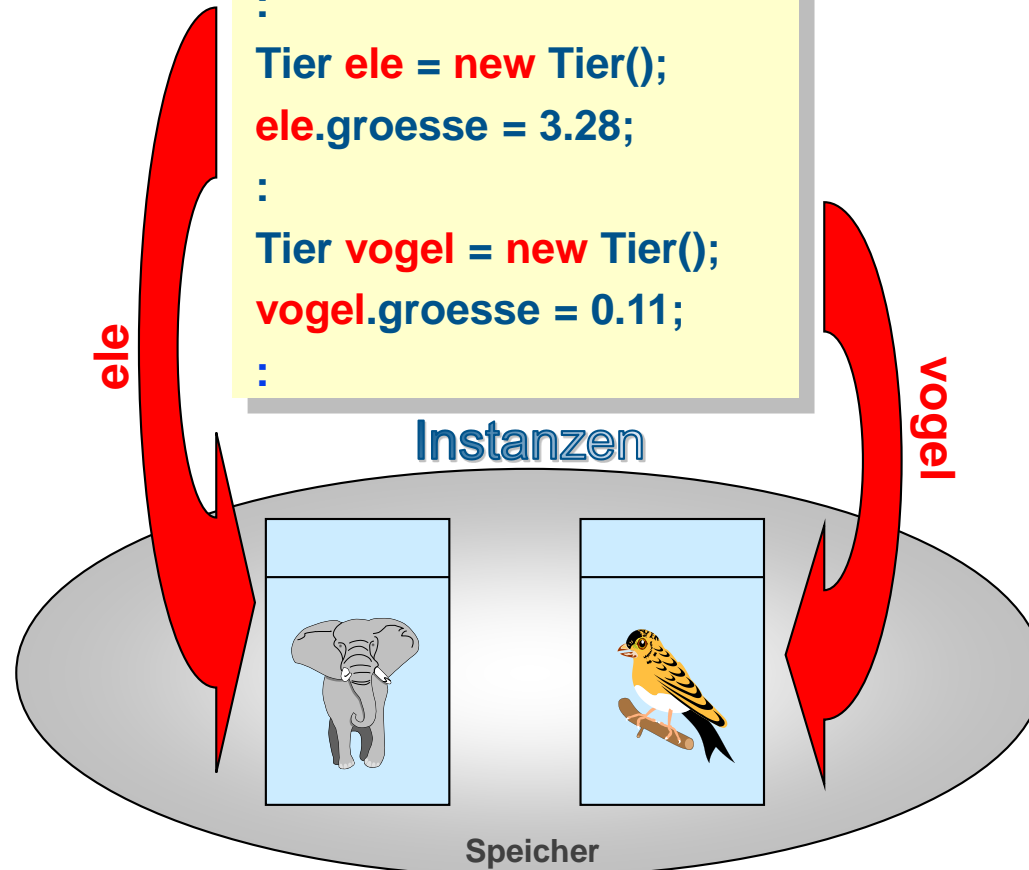
## Schablone



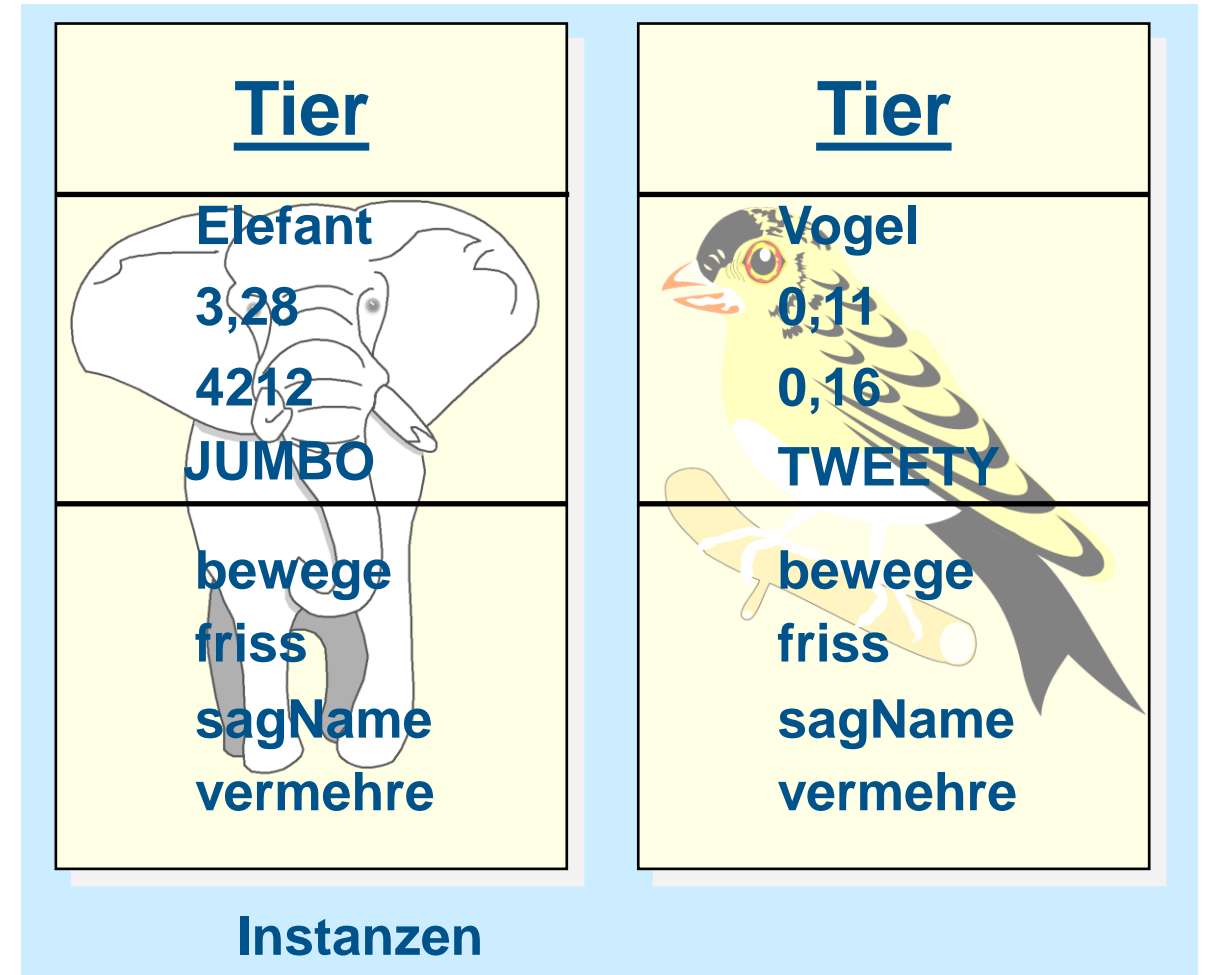
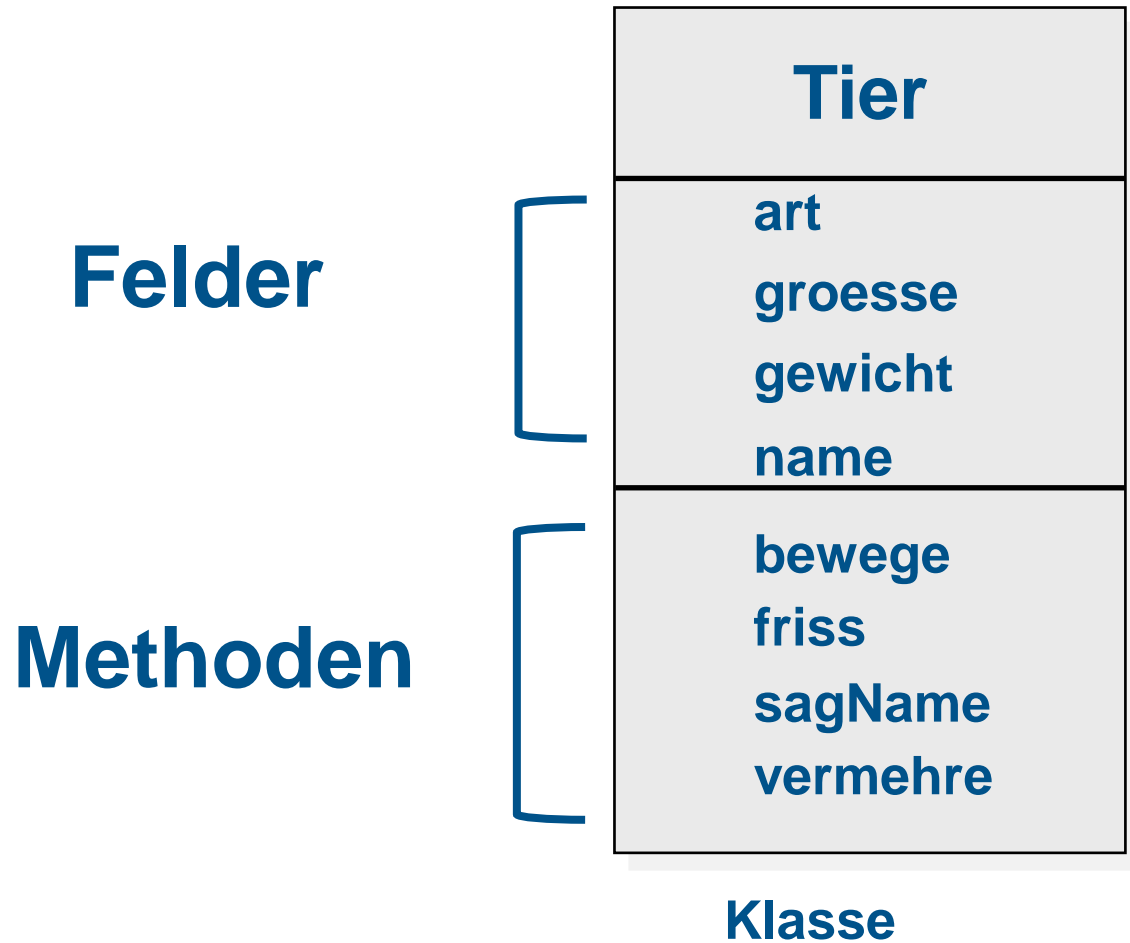
## Anwendung

```
:
Tier ele = new Tier();
ele.groesse = 3.28;
:
Tier vogel = new Tier();
vogel.groesse = 0.11;
:
```

## Instanzen



# Klassen und Objekte (Instanzen)



# Fachklasse und Startklasse

## Fachklasse

```
public class Tier
{
    double groesse;
    double gewicht;
    String name;
    String art;
    public void sagName()
    {
        System.out.println(" Ich bin ein "
            + getArt() + " und heie " + getName());
    }
    public double friss(double pMenge) {
        setGewicht(getGewicht() + pMenge;
        return getGewicht();
    }
    public static void main(String[ ] args)
    {
        :
    }
}
```

Methode zum Testen

## Startklasse

```
public class Tierpark
{
    public static void main(String[ ] args)
    {
        Tier ele = new Tier();
        ele.groesse=3.28;
        ele.gewicht=4212.0;
        ele.name="JUMBO";
        ele.art="Elefant";
        ele.friss(50.0);
        System.out.println
            ("Groesse : " + ele.groesse
            + "\nGewicht  " + ele.gewicht
            + "\nName :   " + ele.name
            + "\nArt :     " + ele.art);
    }
}
```

Methode zum Starten

# Startklasse

- In jedem Java - Programm muss es eine Startklasse geben, die eine Methode mit dem Namen `main()` in ihrem klassenbezogenen Handlungsrepertoire besitzt
- Beim Programmstart wird die Startklasse vom Laufzeitsystem aufgefordert, die Methode `main()` auszuführen.
- Aufruf in der Kommandozeile: `java <Klassenname>`  
Groß- Kleinschreibung beachten!
- `public` → Zugriffsmodifikator, damit die JVM die Klasse starten kann
- `static` → `main` ist eine statische Methode, d. h., diese Methode wird von der Klasse selbst ausgeführt
- `void` → Rückgabetyt, d. h., diese Methode liefert keinen Rückgabewert



# Primitive Datentypen

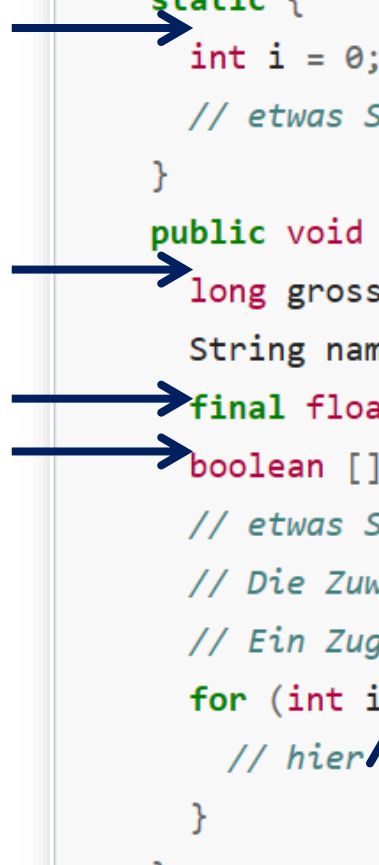
- Variablen zur Aufnahme von
  - Zahlen,
  - Zeichen oder
  - Wahrheitswerten
- Felder von Klassen
- Lokale Variable in einer Methode
- Sie unterscheiden sich hinsichtlich
  - Speichertechnik,
  - Wertebereich und
  - Platzbedarf
- Man nennt sie auch Wertvariablen

# Datentypen

Datentyp	Größe	Werte	Bemerkungen
<b>boolean</b>	1 Byte	<b>true, false</b>	
<b>char</b>	2 Byte	'\u0000' bis '\uFFFF'	Unicode, $2^{16}$ Zeichen, <u>stets in Apostrophs (!)</u>
<b>byte</b>	1 Byte	-128 bis +127	$-2^7, -2^7 + 1, \dots, 2^7 - 1$
<b>short</b>	2 Byte	-32 768 bis +32 767	$-2^{15}, -2^{15} + 1, \dots, 2^{15} - 1$
<b>int</b>	4 Byte	-2 147 483 648 bis +2 147 483 647	$-2^{31}, -2^{31} + 1, \dots, 2^{31} - 1$ Beispiele: 123, -23, 0x1A, 0b110
<b>long</b>	8 Byte	-9 223 372 036 854 775 808 bis +9 223 372 036 854 775 807	$-2^{63}, -2^{63} + 1, \dots, 2^{63} - 1$ Beispiele: 123L, -23L, 0x1AL, 0b110L
<b>float</b>	4 Byte	$\pm 1.4\text{E-}45$ bis $\pm 3.4028235\text{E+}38$	$[\approx \pm 2^{-149}, \approx \pm 2^{128}]$ Beispiele: 1.0f, 1.F, .05F, 3.14E-5F
<b>double</b>	8 Byte	$\pm 4.9\text{E-}324$ bis $\pm 1.7976931348623157\text{E+}308$	$[\pm 2^{-1074}, \approx \pm 2^{1024}]$ Beispiele: 1.0, 1., .05, 3.14E-5

# Primitive Datentypen

```
public class Beispiel {  
    static {  
        int i = 0;  
        // etwas Sinnvolles mit i anstellen  
    }  
    public void machWas() {  
        long grosseGanzeZahl;  
        String name;  
        final float PI = 3.1415;  
        boolean [] wahrheitswerte = new boolean[10];  
        // etwas Sinnvolles mit grosseGanzeZahl, name und PI anstellen  
        // Die Zuweisung PI = 1; verursacht einen Compilerfehler  
        // Ein Zugriff auf i verursacht einen Compilerfehler  
        for (int i=0; i < wahrheitswerte.length; i++) {  
            // hier existiert i (ein anderes als obiges im static Block)  
        }  
    }  
}
```



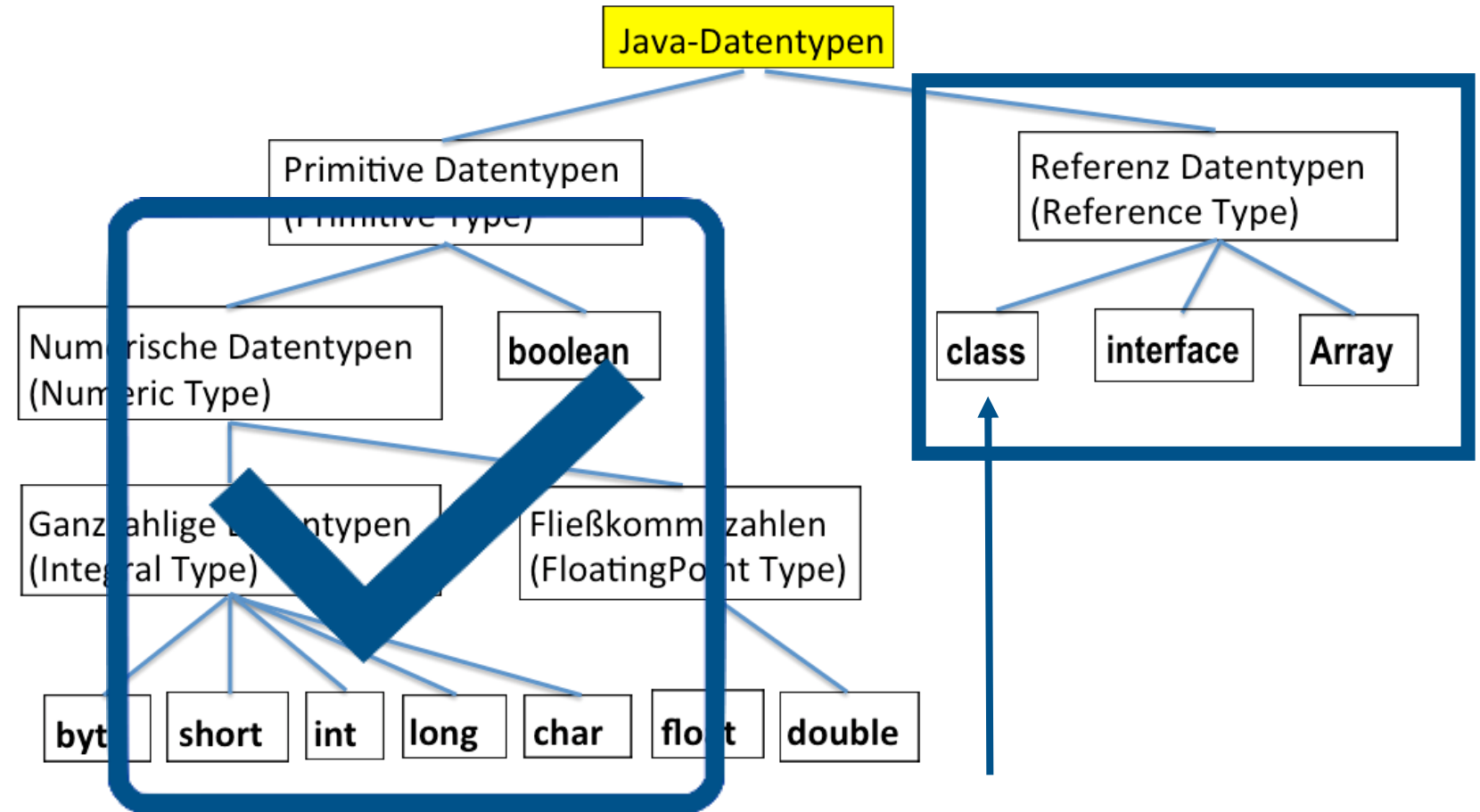
# Datentypen

In Java wird bei **primitiven Datentypen** bei der Anweisung

`zahlA = zahlB;`  
der Wert von Variable `zahlB` kopiert und in Variable `zahlA` gespeichert.

Bei **Objekten** wird bei der Anweisung

`objektA = objektB;`  
nicht das **Objekt** selbst kopiert und übergeben, sondern lediglich die Objektreferenz, also die Speicheradresse



# Referenz

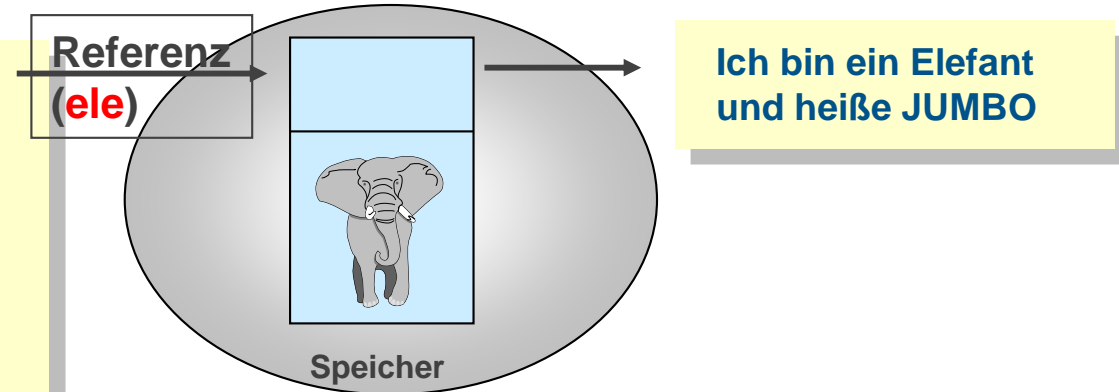
```
public class Tier
{
    double groesse;
    double gewicht;
    String name;
    String art;

    public void sagName()
    {
        System.out.println ("Ich bin ein "
            + art + " und heiße "
            + name );
    }
}
```

Fachklasse

```
:
Tier ele = new Tier();
ele.groesse=3.28;
ele.gewicht=4212.0;
ele.art="Elefant";
ele.name="JUMBO");
:
ele.sagName( );
:
```

Startklasse

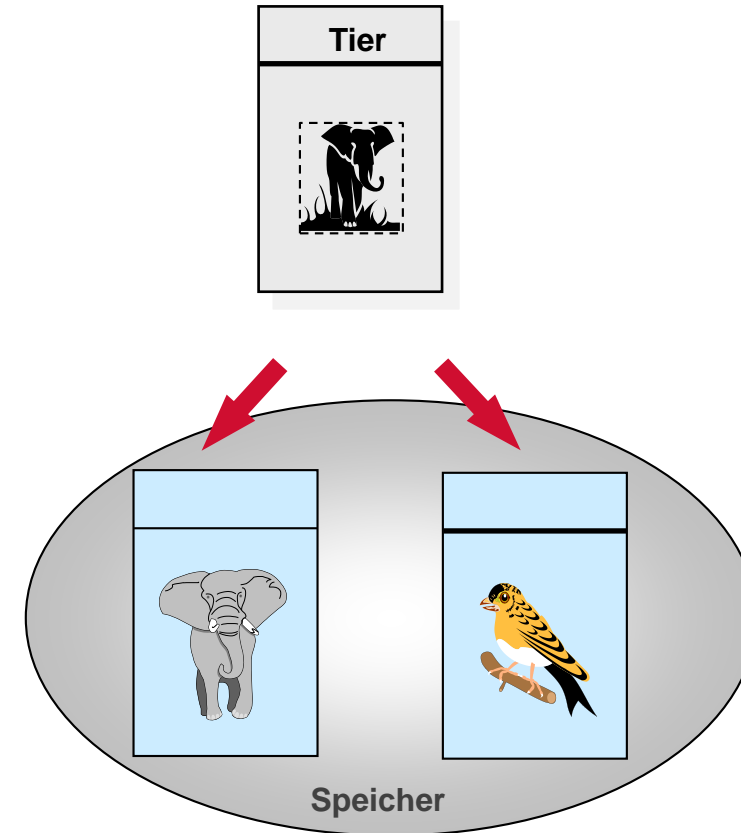


# Weitere Instanzen

:

```
Tier ele = new Tier();  
ele.art="Elefant";  
ele.gewicht=4212.0;  
ele.friss (50.0);  
System.out.println  
    ("Art: " + ele.art +  
     "\nGewicht" + ele.gewicht );  
Tier tweety = new Tier();  
Tweety.art="Vogel";
```

"Anwendungsprogramm"



# Referenz

- Eine Referenz ist ein Zeiger, der auf ein Objekt verweist
- Deshalb besitzt eine Referenz einen viel engeren Rahmen, meist innerhalb eines Programmes (Speicherbereich)
- Objektorientierte Programmiersprachen benutzen Referenzen, weil sich dadurch unterschiedliche Programmteile ein Objekt teilen können: es entstehen Aliase
- Die Anwendung von Referenzen unterstützt Datenkonsistenz und Anwendungsperformanz

# Beispiel Klasse Bruch

Name der Klasse: **Bruch**

Zentrale Eigenschaften eines Bruch-Objekts: Zähler und Nenner  
Sie werden durch Felder zum Speichern von ganzen Zahlen (int) dargestellt

Felder:

- **zaehler**
- **nenner**

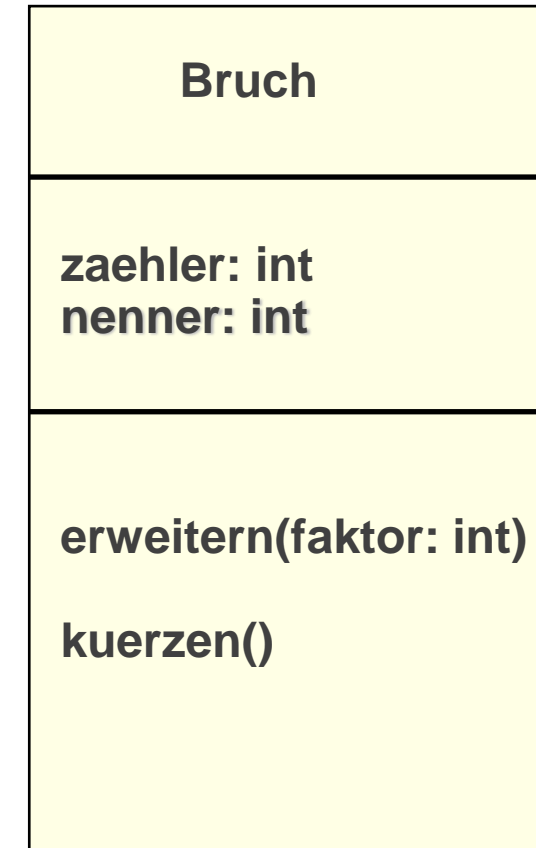
Die Klasse Bruch sollte den Zähler und den Nenner um einen Faktor erweitern können

Die Klasse Bruch sollte Zähler und Nenner kürzen können

Methoden:

- **erweitern**
- **kuerzen**

Die Klasse soll getestet werden.





**Aufgabe**



# Aufgabe

Programmieren von Klassen

Bruch

(gemeinsam mit Trainer)

# Deklarieren – Initialisieren – Instanziieren

## ○ Deklarieren

- Erstes Erwähnen einer Variablen

```
int zaehler;  
int nenner;
```

## ○ Initialisieren

- Erstes Zuweisen eines Wertes

```
double dblUmsatz, dblProvsatz;  
dblUmsatz = 35000;
```

```
int a = 4;  
int b = 7;
```

## ○ Instanziieren

- Erzeugen eines Objekts/Instanz

```
String text;  
text = "Beispieltext";  
text = new String("Beispieltext");  
System.out.println("Text: " + text);  
  
Car meinAuto;  
meinAuto = new Car();
```

# Nullwert-Konzept



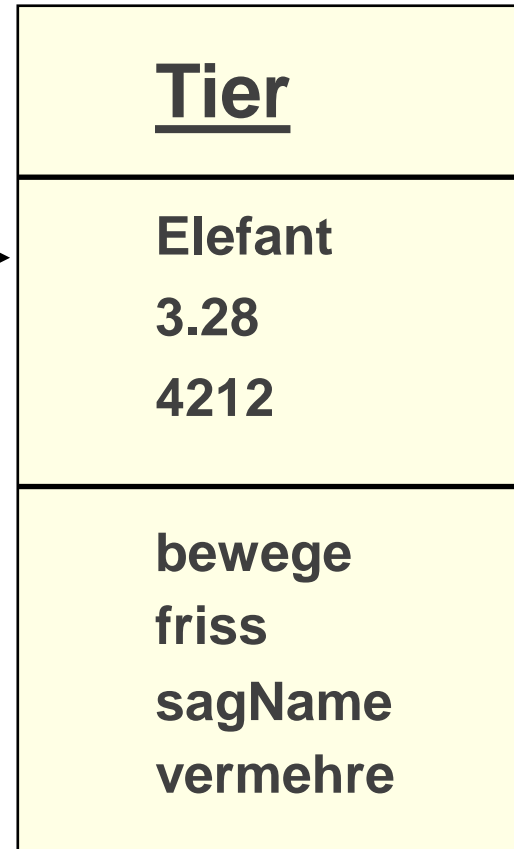
```
Tier ele;  
ele = new Tier();
```



```
ele = null;
```



```
if (ele == null )
```



Objekt

# Initialisieren

- Instanzvariablen im **Konstruktor** initialisieren, um Fehler durch ungültige Werte zu vermeiden (bspw. Bruchrechnen, Zähler > 0 und Nenner = 0, Teilen durch 0 ist nicht möglich)
- Weitere Attribute, die notwendig sind, aber nichts über das Objekt aussagen, an passender Stelle initialisieren

Datentyp	Standardwert
byte	(byte) 0
short	(short) 0
int	0
long	0L
float	0.0f
double	0.0d
char	(char) 0
boolean	false
Referenzdatentyp	null

Defaultwerte von Instanzvariablen

# Java Konstruktor Typen

Ein Konstruktor

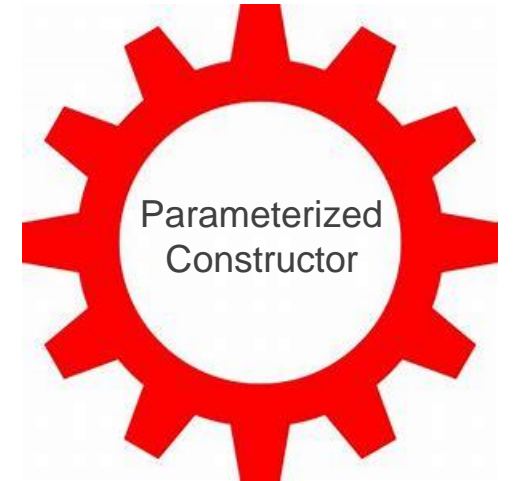
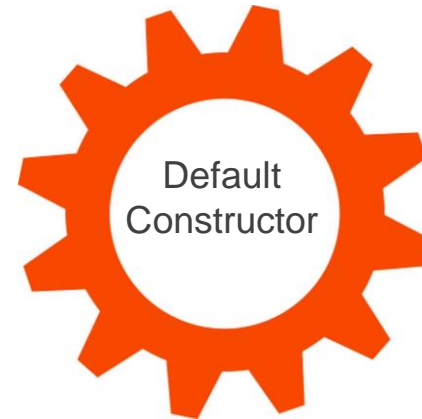
- dient zum Erzeugen neuer valider Objekte
- ist für die Initialisierung eines Objektes zuständig
- besitzt den gleichen Namen wie die Klasse
- besitzt keinen Rückgabewert

**Default constructor** (no-arg constructor):

Wenn kein Konstruktor definiert wurde, wird der Default-Konstruktor verwendet

**Parameterized constructor** (arg constructor)

Diesem Konstruktor können Parameter (Argumente) mitgegeben werden, die dann in der Methode verwendet werden



```
new <Klasse> ()
```

```
new <Klasse> (1, "A")
```

# Unterschied Konstruktor/Methode

## Konstruktor

- ist eine spezielle Methode in einer Klasse, die Objekte dieser Klasse erstellt und initialisiert
- Aufruf mit dem Schlüsselwort „new“
- hat keinen Returntype
- wird implizit aufgerufen
- ist kein Konstruktor vorhanden, wird ein Standard (default)-Konstruktor aufgerufen
- heißt wie die Klasse
- wird nicht vererbt

## Methode

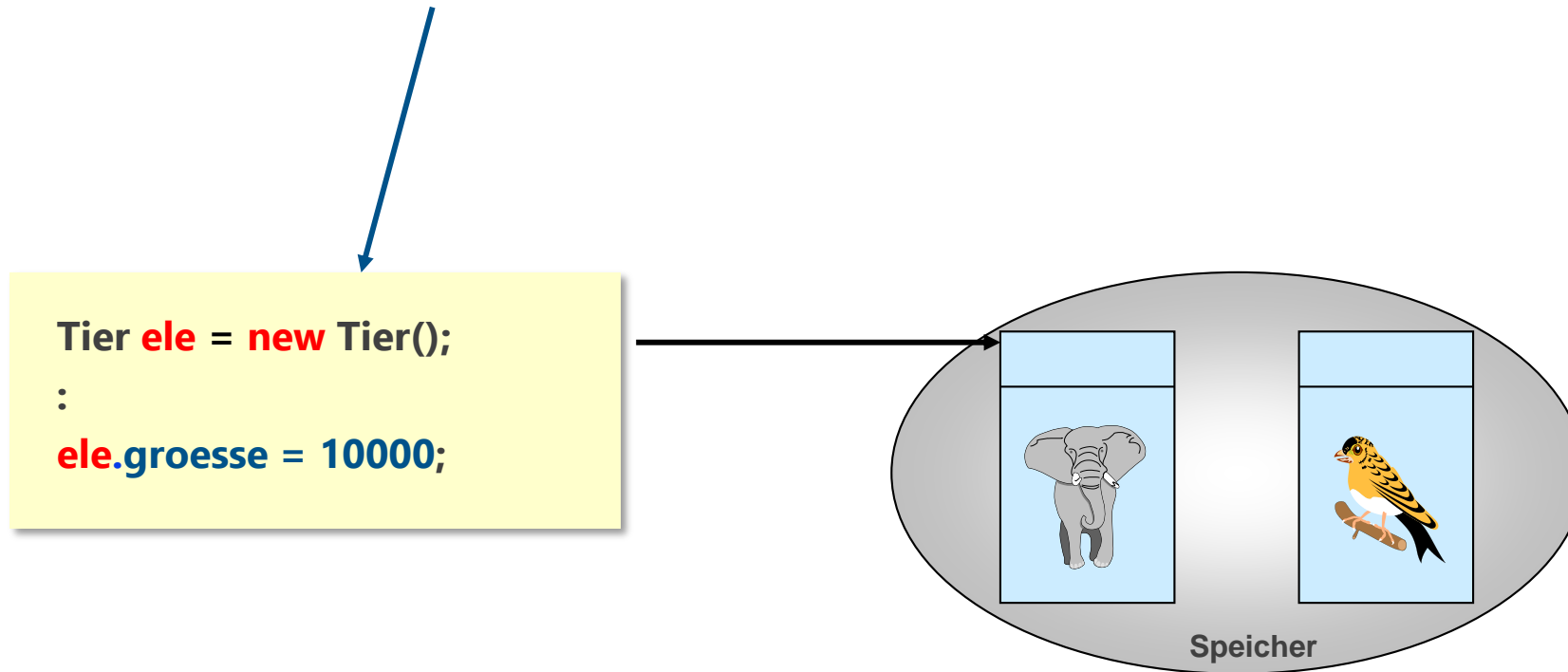
- Ist eine Prozedur oder eine Funktion, die einen Satz von Anweisungen (Aufgaben) ausführt, die einer Klasse zugeordnet sind
- muss einen Returntype haben (z. B. int oder void)
- wird explizit aufgerufen
- es gibt keine default-Methode
- der Name unterscheidet sich vom Klassennamen
- können vererbt werden

# Konstruktor

- Der Konstruktor ist eine Methode, die bei Instanziierung (new) einer Klasse aufgerufen wird
- Der Standardkonstruktor hat keine Parameter
- Im Konstruktor findet die Initialisierung statt, also alle Vorgänge, die genau einmal zum Start erledigt werden sollten, bevor die Variable benutzt wird. Da die Variable sich quasi selbst vorbereitet, werden Initialisierungsfehler vermieden und der Code wird stabiler
- Ein Konstruktor trägt den Namen der Klasse und hat keinen Rückgabewert
- Es kann weitere (überladene) Konstruktoren mit Parametern geben

# Default (Standard) Konstruktor

- Wird durch den Compiler zur Verfügung gestellt
- Wird mit dem Schlüsselwort `new`, dem Klassennamen und zwei runden Klammern `()` aufgerufen

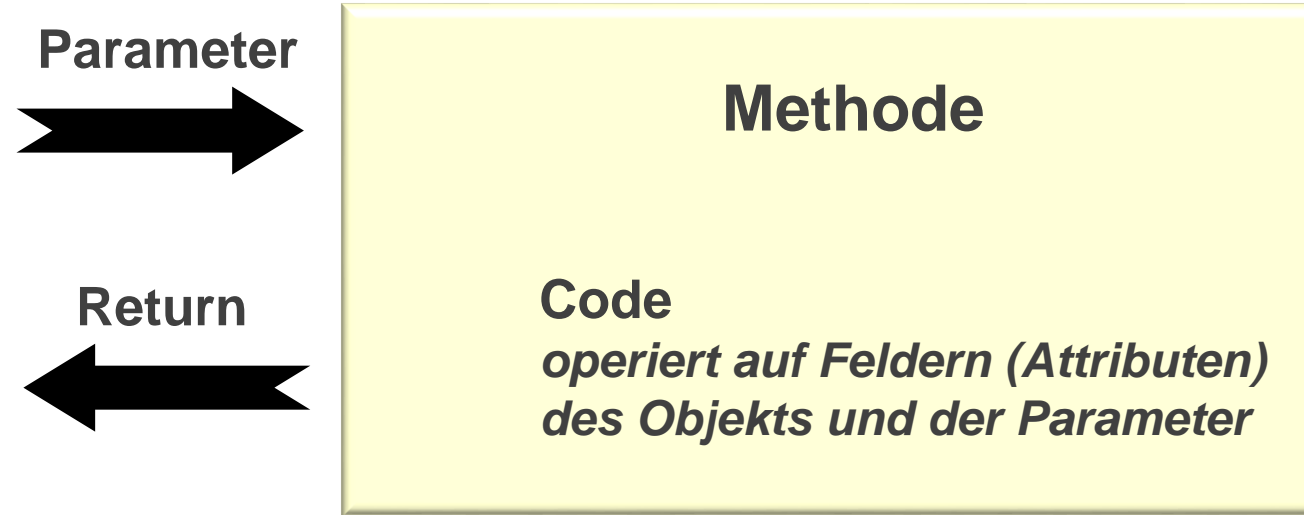




# Methoden

- Eine Methode fasst mehrere Anweisungen zusammen
- Besitzt einen Namen, mit dessen Aufruf diese Anweisungen ausgeführt werden
- Fasst häufig benutzten Code in einer Einheit zusammen
- Methoden können eine Referenz zurückgeben
- Methoden bestehen aus:
  - Kopf (Deklaration)
  - Rumpf (Definition), innerhalb geschweifter Klammern

# Methoden

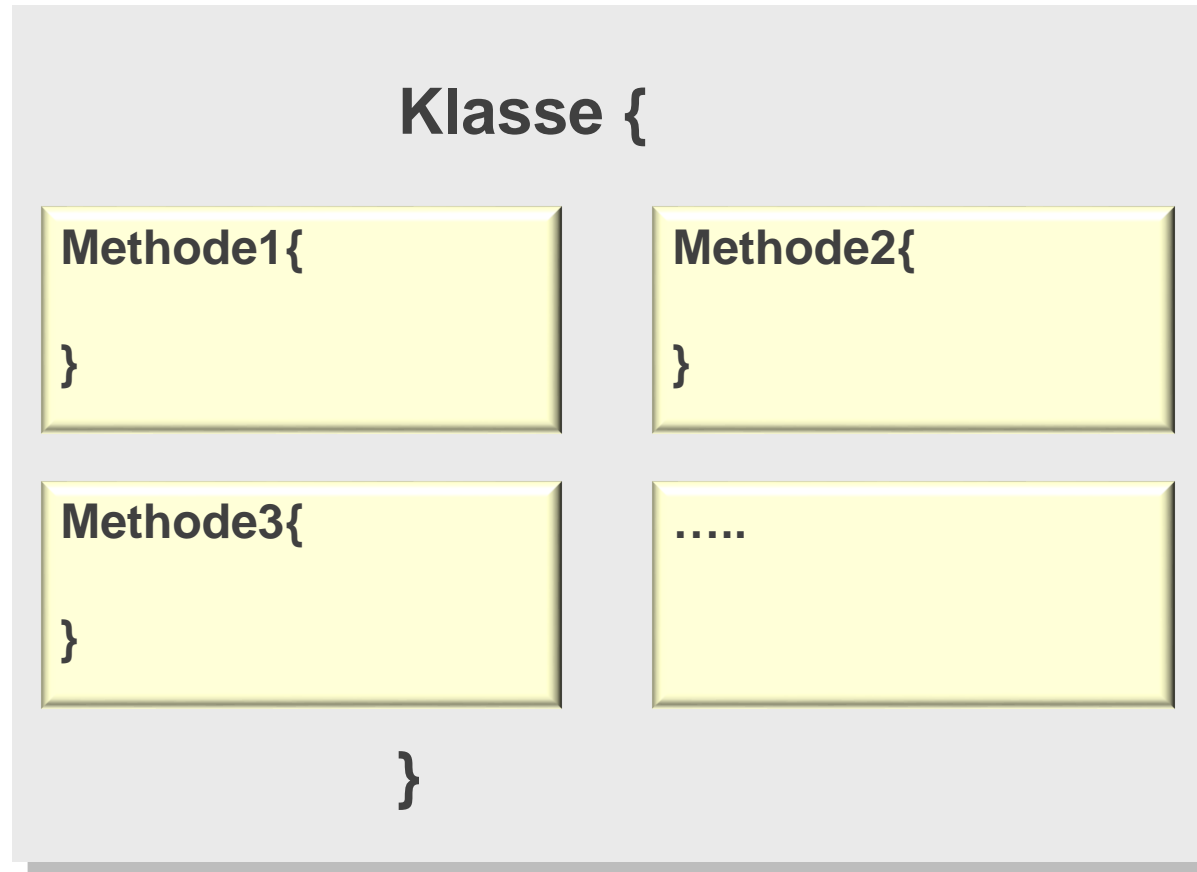


## Variablen

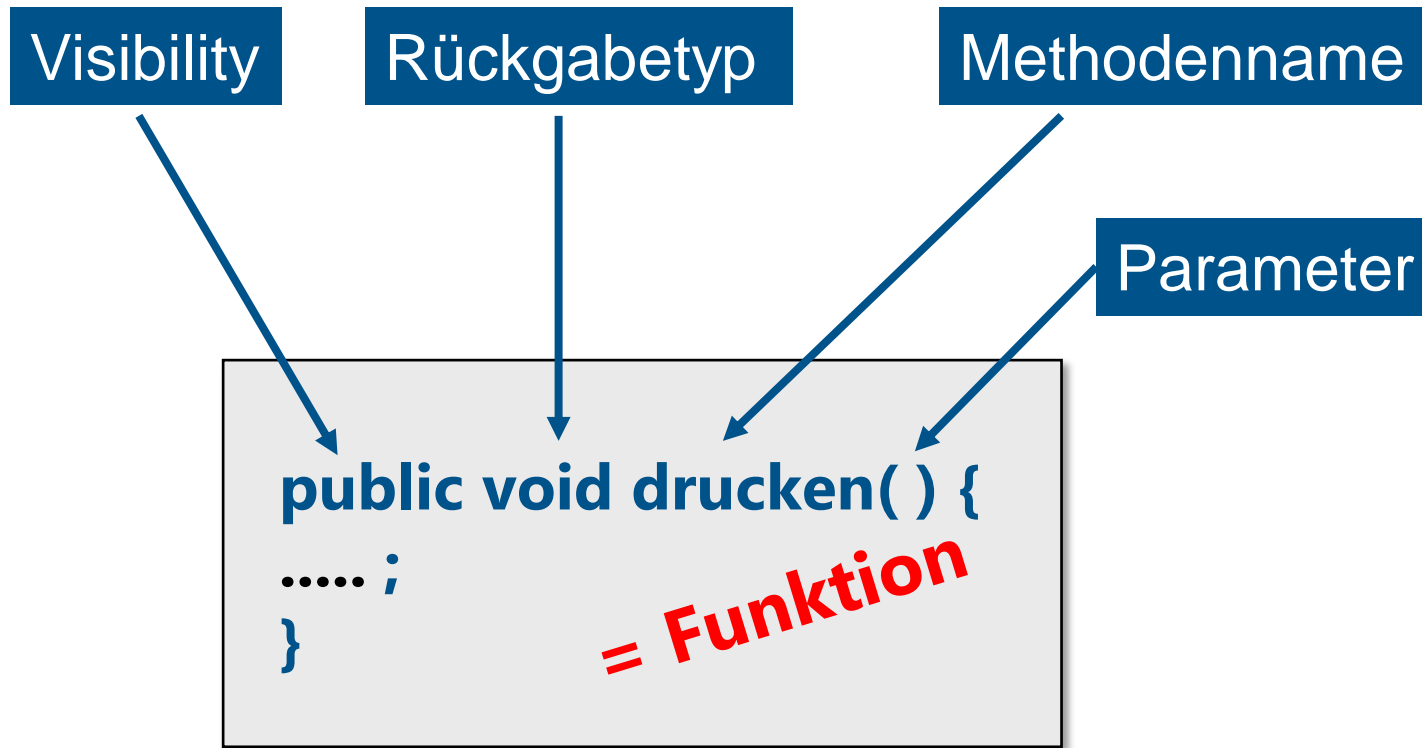
Instanzvariablen → Definition in einer Klasse

Lokale Variablen → Definition in einer Methode

# Methoden

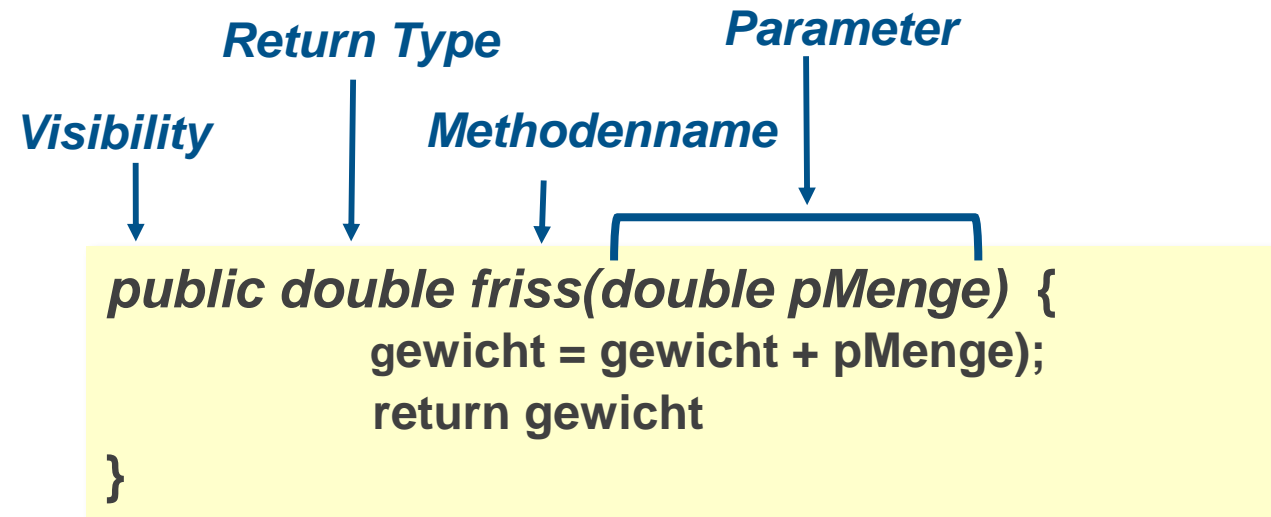
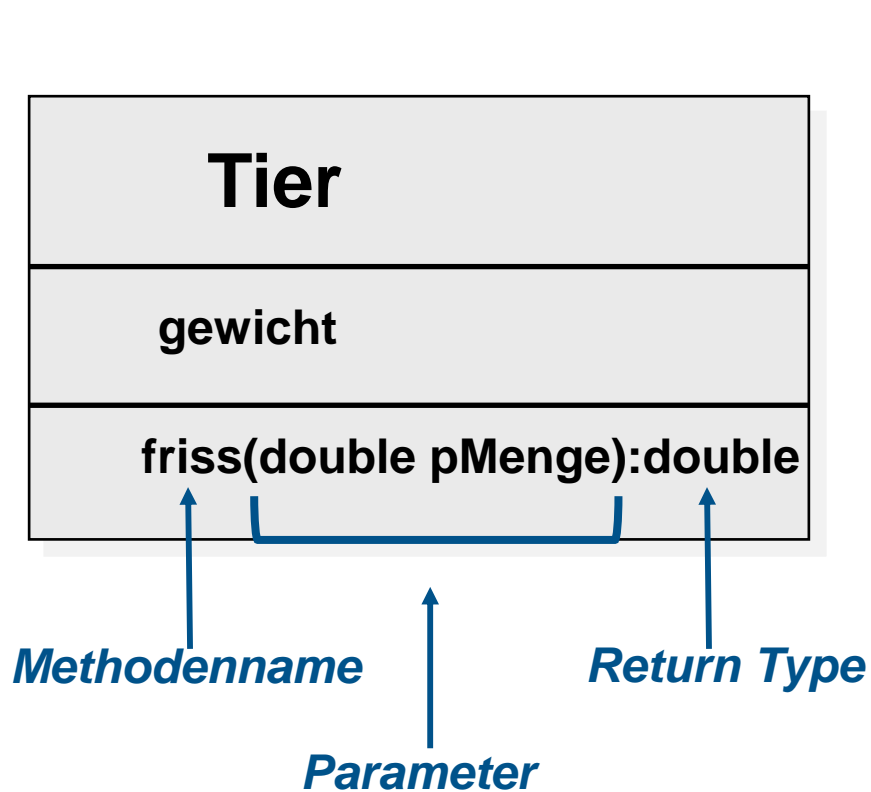


# Definieren von Methoden



Der Methodenrumpf ist mit geschweiften Klammern versehen

# ... mit Parameter und Rückgabewert



# Aufbau einer Klasse

```
public class Tier {
```

```
// Felder
```

```
String art;
```

```
double gewicht;
```

```
// Konstruktor
```

```
public Tier(){
```

```
    art= " ";
```

```
    gewicht = 4212.0;
```

```
} // schließende Konstruktorklammer
```

```
//Methode
```

```
public double friss(double pMenge) {
```

```
    gewicht = gewicht + pMenge;
```

```
    return gewicht;
```

```
} // schließende Methodenklammer
```

```
} // schließende Klassenklammer
```

# Ursprung aller Klassen

- Alle Java-Klassen erben automatisch von **Object**, keine explizite Code-Angabe notwendig
- Die Klasse **Object** stellt nützliche Methoden zur Verfügung
- Die Klasse **Object** ist in der Java-Bibliothek java.lang zu finden
  
- Um eine String-Repräsentation des aktuellen Objekts zu bewirken, steht die Methode toString() bereits zur Verfügung

# Standardverhalten der Methode toString()

Die String-Repräsentation des Objekts, die beim Aufruf der **toString()-Methode** standardmäßig zurückgeliefert wird, ist eine Zusammensetzung aus folgenden Teilen:

- Vollständiger **Klassenname** (inkl. Paketname).
- @-Zeichen
- **Adresse** des Objekts im Hauptspeicher in hexadezimaler Notation

**toString()** ist in der Klasse Object implementiert; der Methodencode sieht so aus:

```
public String toString(){  
    getClass().getName() + '@' + Integer.toHexString(hashCode());  
}
```



# Standardverhalten der Methode toString()

```
1  package tag03.oop;
2
3  public class TierEinfach {
4      // Attribute
5      private String name;
6      private int alter;
7
8      // Konstruktoren
9      public TierEinfach(String n, int a) {
10         this.name = n;
11         this.alter = a;
12     }
13     // Default-Konstruktor
14     public TierEinfach () {
15         this("Jumbo", 5);
16     }
17
18     //Methode zum testen
19     public static void main(String[] args) {
20         TierEinfach tier = new TierEinfach("Waldi",2);
21         System.out.println(tier.toString());
22     }
23 }
```

```
RUN:
tag03.oop.TierEinfach@15db9742
BUILD SUCCESSFUL (total time: 0 seconds)
```

Alternativ  
ohne toString():

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

# Überschreiben der Methode toString()

## @Override

Beim Überschreiben von toString() haben wir die Annotation @Override der Methodensignatur vorangestellt

```
@Override
public String toString() {
    return "Tier [name=" + name + ", alter=" + alter + "];"
}
```

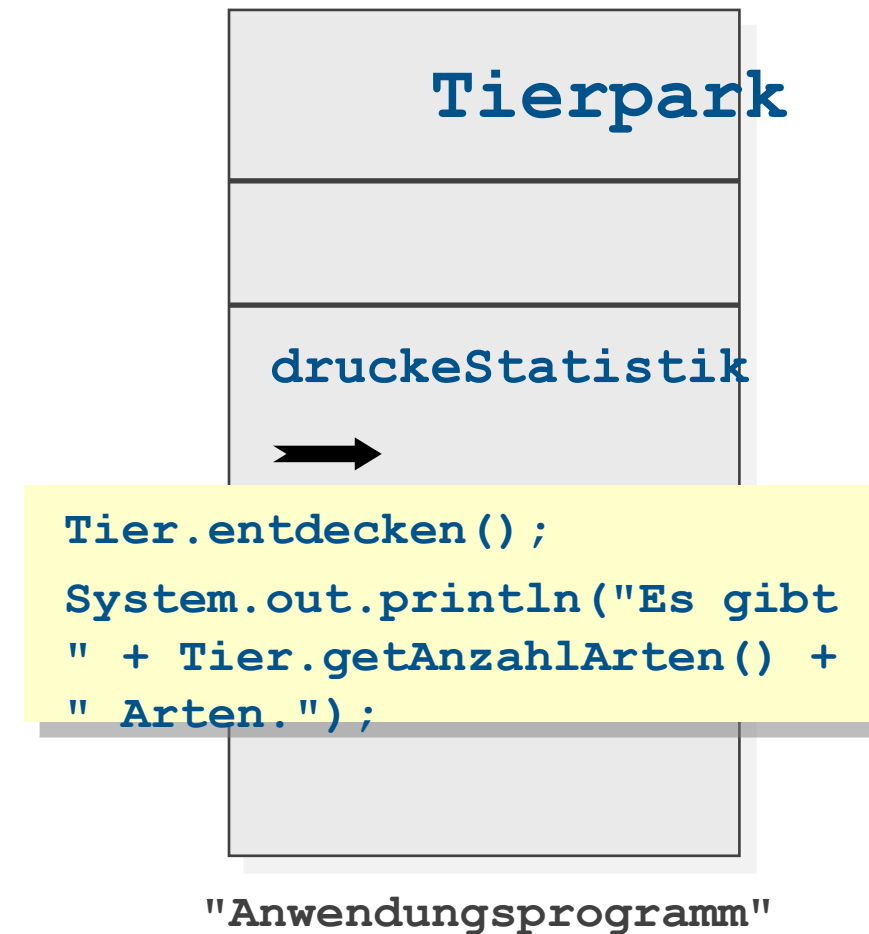
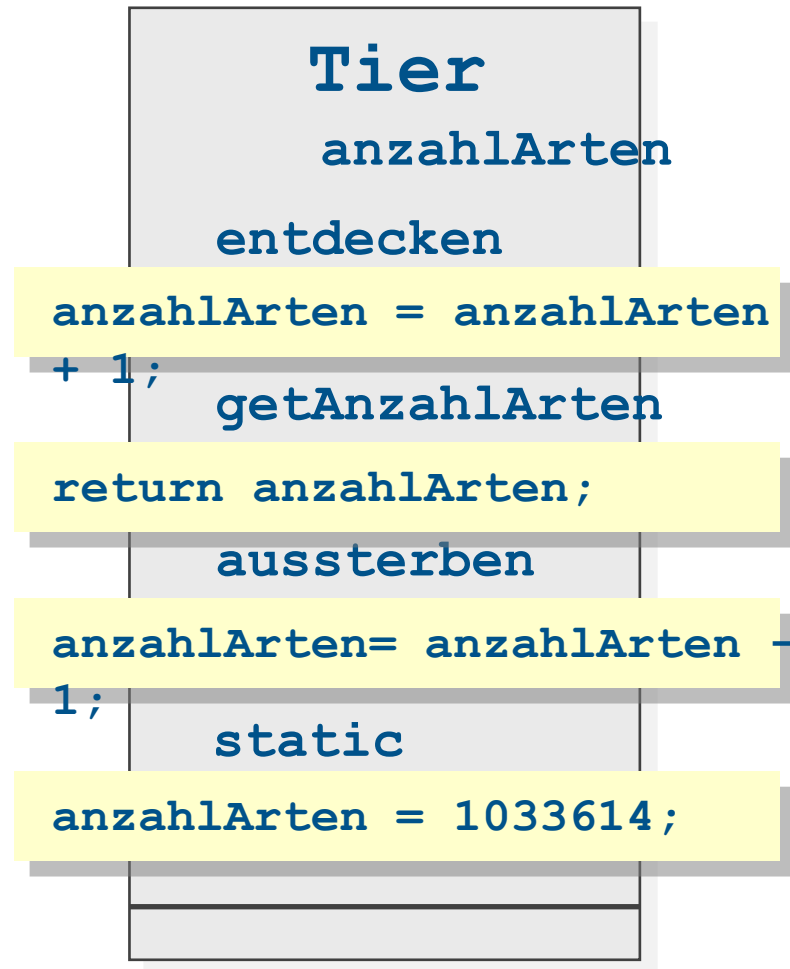
Diese Anmerkung ist *freiwillig*, gehört aber zum guten Stil und bietet zwei Vorteile:

- Bessere **Lesbarkeit** des Codes
- Der **Compiler** prüft, ob tatsächlich eine Methode aus einer Oberklasse überschrieben wird.  
Ist die Methodensignatur nicht identisch mit einer Methode aus einer Superklasse, (d. h., es würde eine neue Methode hinzugefügt oder eine Methode überladen), erhalten wir praktischerweise eine Compiler-Meldung (etwa: Method does not override method from its superclass)

```
run:
Tier [name=Waldi, alter=2]
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Klassen, Felder und Methoden - ohne Instanz

- Klassenvariable
- Klassenmethoden
- Klassenkonstruktor



# Statisches

```
:  
public static int anzahlArten;  
  
public static void entdecken  
{  
    anzahlArten = anzahlArten + 1;  
}  
:  
static  
{  
    . . .  
}  
. . .
```

Klassenfeld

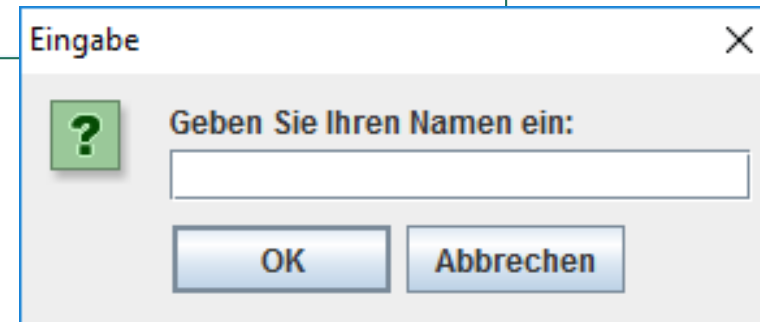
Klassenmethode

Klassenkonstruktor

# Die Klasse JOptionPane

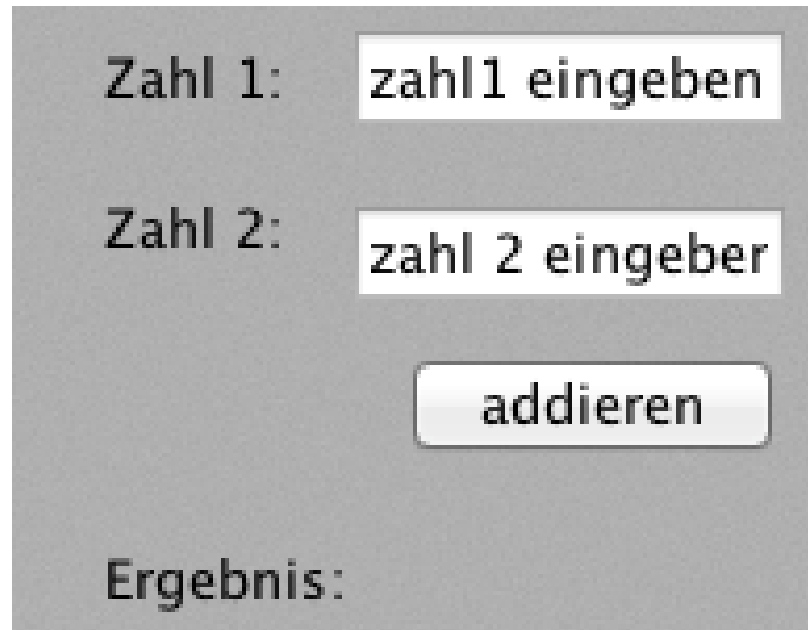
Nimmt einfache Texteingaben entgegen. Erzeugt ein kleines Fenster.

```
1 package teila;  
2  
3 import javax.swing.JOptionPane;  
4  
5 public class EingabeJOption {  
6  
7     public static void main(String[] args) {  
8         // TODO Auto-generated method stub  
9         String eingabe;  
10        eingabe = JOptionPane.showInputDialog("Geben Sie Ihren Namen ein: ");  
11        System.out.println("Ihr Name lautet: "+ eingabe);  
12    }  
13  
14 }
```



# Zeichenkette → Zahl

- Wenn in ein Textfeld Zahlen eingegeben werden, müssen diese erst mit *parseInt*, *parseFloat* o. Ä. in Zahlentyp umgewandelt werden!



The image shows a gray rectangular form with a light gray border. It contains the following elements:

- Zahl 1:** A label followed by a text input field containing the placeholder text "zahl1 eingeben".
- Zahl 2:** A label followed by a text input field containing the placeholder text "zahl 2 eingeber".
- addieren**: A button with a gradient and rounded corners.
- Ergebnis:** A label at the bottom left of the form.

# Zeichenkette → Zahl

```
int a = 17;  
// falsch: lblErgebnis.setText(a);
```

// funktioniert nicht  
// keine mathematischen Operationen mit Zeichenketten!

```
int erg = (int) (a*a);
```

// Casting nicht möglich

```
String a = "13";  
int b = Integer.parseInt(a);  
int ergebnis = b * b;
```

```
String c = "1.332";  
double d = Double.parseDouble(c);
```

<https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html>  
<https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html>

# Zahl → Zeichenkette

```
int a = 17;  
// falsch: lblErgebnis.setText(a);
```

```
Int a = 17  
String b = String.valueOf(a);  
lblErgebnis.setText(b);
```

// funktioniert nicht  
// keine numerischen Parameter bei bestimmten Methoden!

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>



# Zusammenfassung → wrappen

Zeichenkette in Zahl:

**Integer.parseInt(String)**

(z. B. nach Eingabe in Textfeld)

Zahl in Zeichenkette:

**String.valueOf(int/double/...)**

(z. B., um Zahl auf Label abzubilden)

# Wrapper Klassen

Datentyp	Größe [Bit]	Standardwert	Wertebereich	Wrapper-Klasse
boolean	1	<i>false</i>	<i>true, false</i>	java.lang.Boolean
byte	8	0	-128 ... +127	java.lang.Byte
short	16	0	$-2^{15} \dots 2^{15}-1$	java.lang.Short
int	32	0	$-2^{31} \dots 2^{31}-1$	java.lang.Integer
long	64	0	$-2^{63} \dots 2^{63}-1$	java.lang.Long
float	32	0.0	$\pm 1.2 \times 10^{-38} \dots \pm 3.4 \times 10^{38}$	java.lang.Float
double	64	0.0	$\pm 2,2 \times 10^{-308} \dots \pm 1.8 \times 10^{308}$	java.lang.Double
char	16	\u0000	‚\u0000‘ ... ‚\uFFFF‘	java.lang.Character
void	–	–	–	java.lang.Void

Eine *Wrapper*-Klasse kapselt die jeweilige primitive Variable in einem sehr einfachen Objekt. Dabei stellt die *Wrapper*-Klasse einige Methoden für den Zugriff auf die primitive Variable und nützliche Funktionen zur Verfügung

# Wrapper Klassen

```
int a = 1;  
int b = 2;  
int c;  
Integer cc;
```

```
c = a + b;  
System.out.println("Ausgabe : " + c);
```

```
// wie wird aus einem int ein Integer ?  
cc = new Integer(c);  
System.out.println("\nAusgabe : " + cc);
```

```
// wie wird aus einem Integer ein int ??  
c = cc.intValue();  
System.out.println("\nAusgabe : " + c);
```

# Kompetenzcheck



Welche Aussagen sind richtig?

- a. Unter einer Variablendeklaration ordnet man den Typ zu und legt den Wertebereich fest
- b. Primitiver Datentyp enthält direkt den Wert
- c. Ein String ist ein Referenzdatentyp
- d. Der default-Konstruktor muss selbst implementiert werden
- e. Methoden können sich gegenseitig aufrufen
- f. Von der Klasse Object kann nur einmal abgeleitet werden
- g. Um statische Methoden aufzurufen, muss erst ein Objekt erzeugt werden
- h. Ein Java-Programm besteht aus einer oder mehreren Klassen

**Aufgabe**



# Aufgabe

Programmieren von Klassen

Kreis, Rechteck, Kreistabelle, flächengleicher Kreis