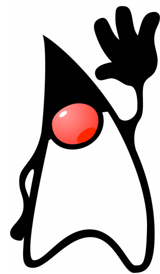
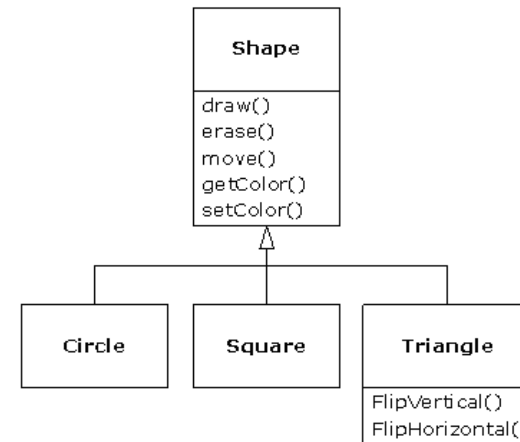
Algorithmen und Programmierung II

Vererbung





Prof. Dr. Margarita Esponda

SS 2012

Imperative Grundbestandteile

Parameterübergabe

String-Klasse

Array-Klasse

Konzepte objektorientierter Programmierung

Vererbung

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.

Parameterübergabe mit primitiven Datentypen

```
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);
   }
}
```

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

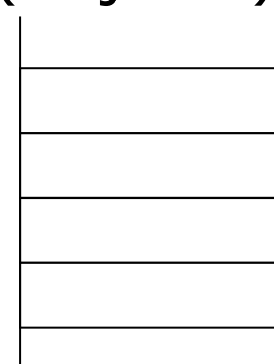
n=20

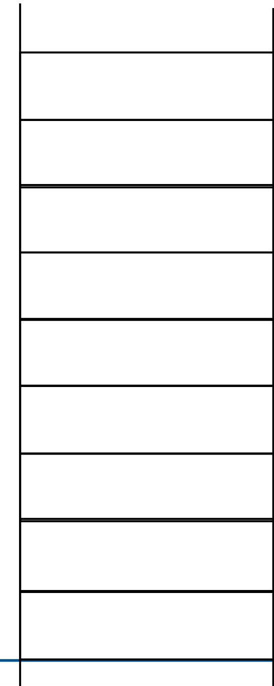
Parameterübergabe mit Referenz-Variablen (Objekte)

```
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() );
   }
}
```



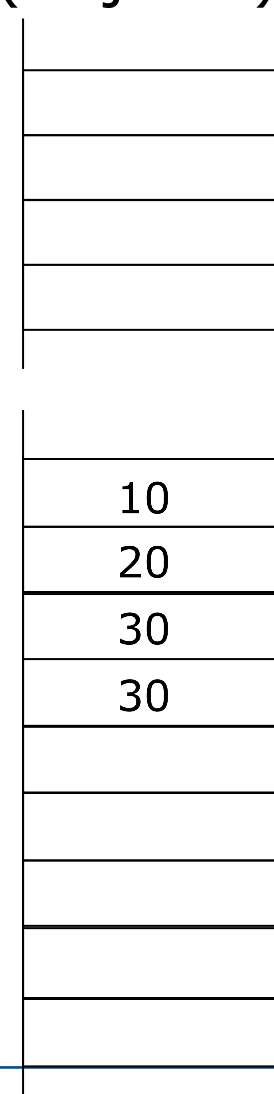


Parameterübergabe mit Referenz-Variablen (Objekte)

```
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() );
   }
}
```

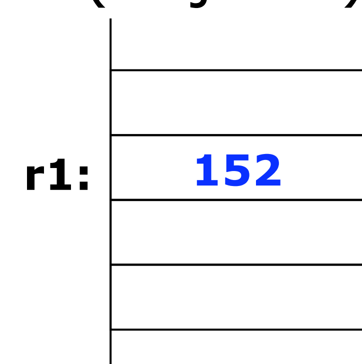


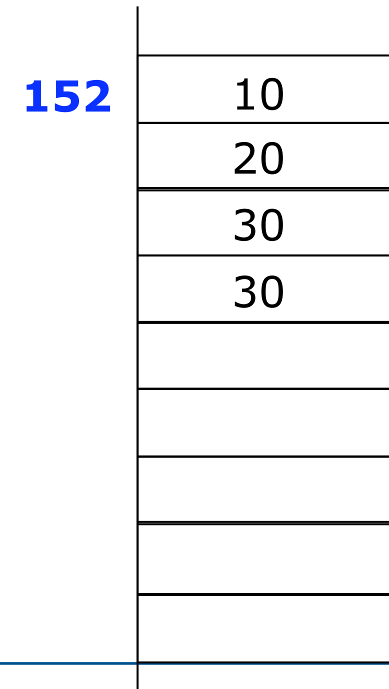
Parameterübergabe mit Referenz-Variablen (Objekte)

```
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() );
    }
}
```



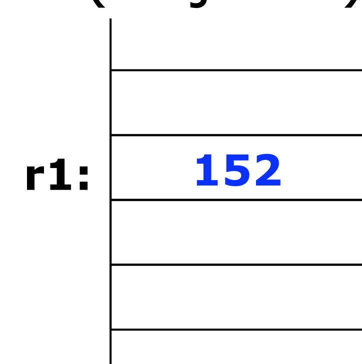


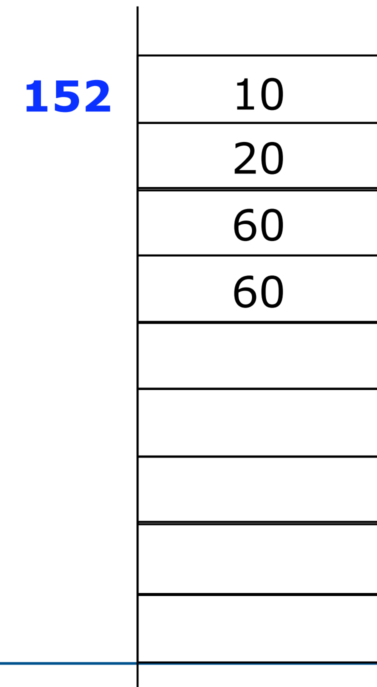
Parameterübergabe mit Referenz-Variablen (Objekte)

```
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() );
   }
}
```



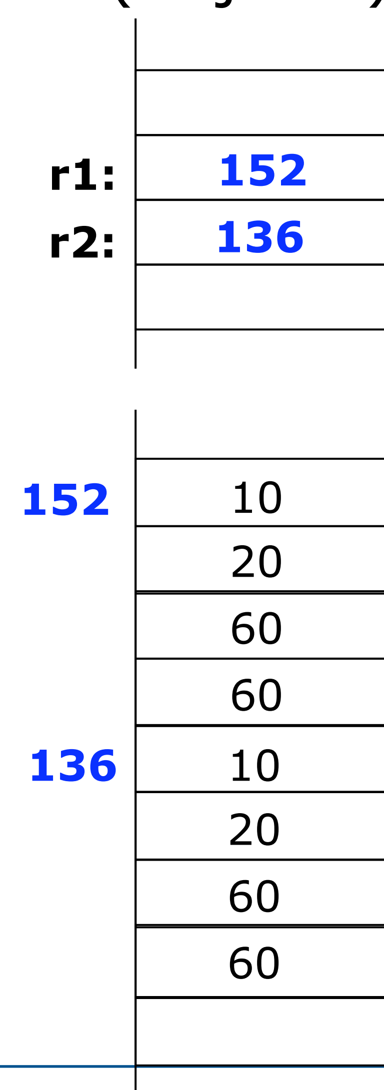


Parameterübergabe mit Referenz-Variablen (Objekte)

```
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() );
   }
}
```



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

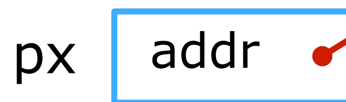
C/C++ Zeiger vs. Referenzen

Zeiger (Pointers)

```
...
int x = 78;
int *px = &x;
...
```

Referenzen (References)

```
...
int x = 78;
int & rx = x;
...
```



```
*px = 5;
```

```
rx = 5;
```

Um durch **px** die Variable **x** zu verändern, müssen wir explizit mit Hilfe des * Operators dereferenzieren.

Hier findet eine implizite automatische Dereferenzierung statt.

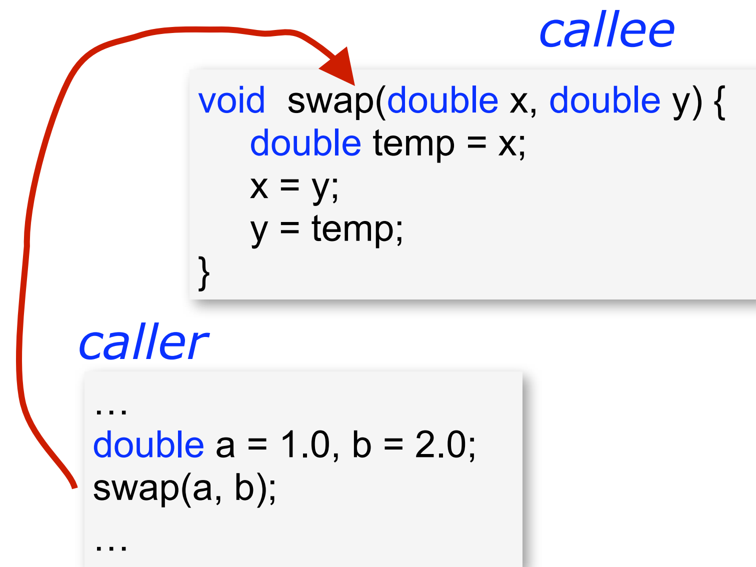
Freie Universität Berlin

C/C++ Zeiger vs. Referenzen

Zeiger (Pointers)	Referenzen (References)
Arithmetische Operationen wie **px++** sind erlaubt.	Arithmetische Operationen sind nicht möglich.
Sie müssen nicht initialisiert werden.	Sie müssen immer initialisiert werden.
Sie sind veränderbar, d.h. während ihrer Lebenszeit können sie auf verschiedene Objekte zeigen.	Sie können nicht verändert werden, d. h. während ihrer gesamten Lebenszeit zeigen sie auf das selbe Objekt.
sehr fehleranfällig	**sicherer**

call-by-value vs. call-by-reference

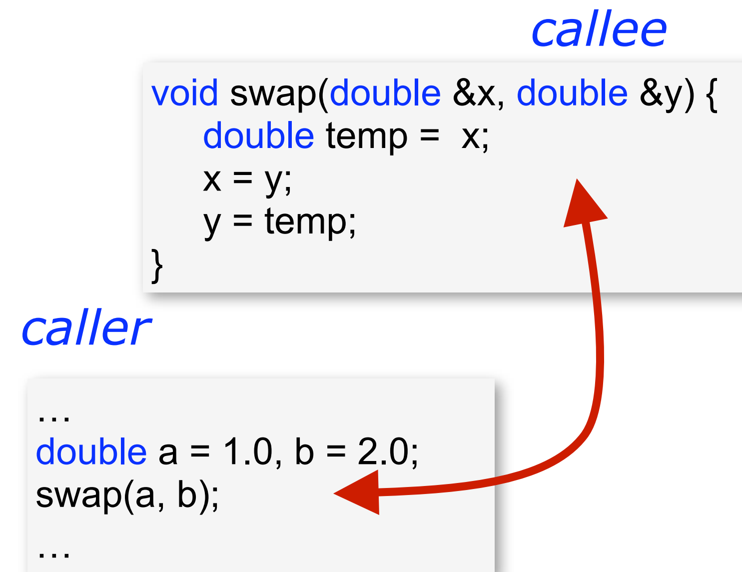
IN-Modus



Die Information (Kopie der aktuellen Parameter) geht nur von *caller* zum *callee.*

Die lokalen Variablen von *caller* können von *callee* nicht verändert werden.

IN/OUT-Modus



Die Information geht in beide Richtungen.

callee kann die lokalen Variablen von *caller* ändern.

call-by-value vs. call-by-reference

IN-Modus

callee

```
void  swap(double x, double y) {
    double temp = x;
    x = y;
    y = temp;
}
```

caller

```
…
double a = 1.0, b = 2.0;
swap(a, b);
…
```

a: 1.0
b: 2.0

C, Java

IN/OUT-Modus

callee

```
void swap(double &x, double &y) {
    double temp =  x;
    x = y;
    y = temp;
}
```

caller

```
…
double a = 1.0, b = 2.0;
swap(a, b);
…
```

a: 2.0
b: 1.0

Der Compiler sorgt dafür, dass die aktuellen und die formalen Parameter auf die gleichen Speicheradressen zeigen.

C++, Pascal

In C kann ***call-by-Pointer*** programmiert werden.

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 konstant, d.h. unveränderlich.

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

```
String s  = "hallo";        s.length()  → 5   (Wert)

s  =  s + "Welt!";

||

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

Arrays

In Java kann man eine Folge gleichartiger Objekte unter einem Namen zusammenfassen. (Reihungen)

Eindimensionale Felder (Arrays)

Beispiel:

```
int[] zahlen;
zahlen = new int[12];
```

```
int[] zahlen = new int[12];
```

0 1 ...

```
int[] primes = { 2, 3, 5, 7, 11, 13, 17 };
```

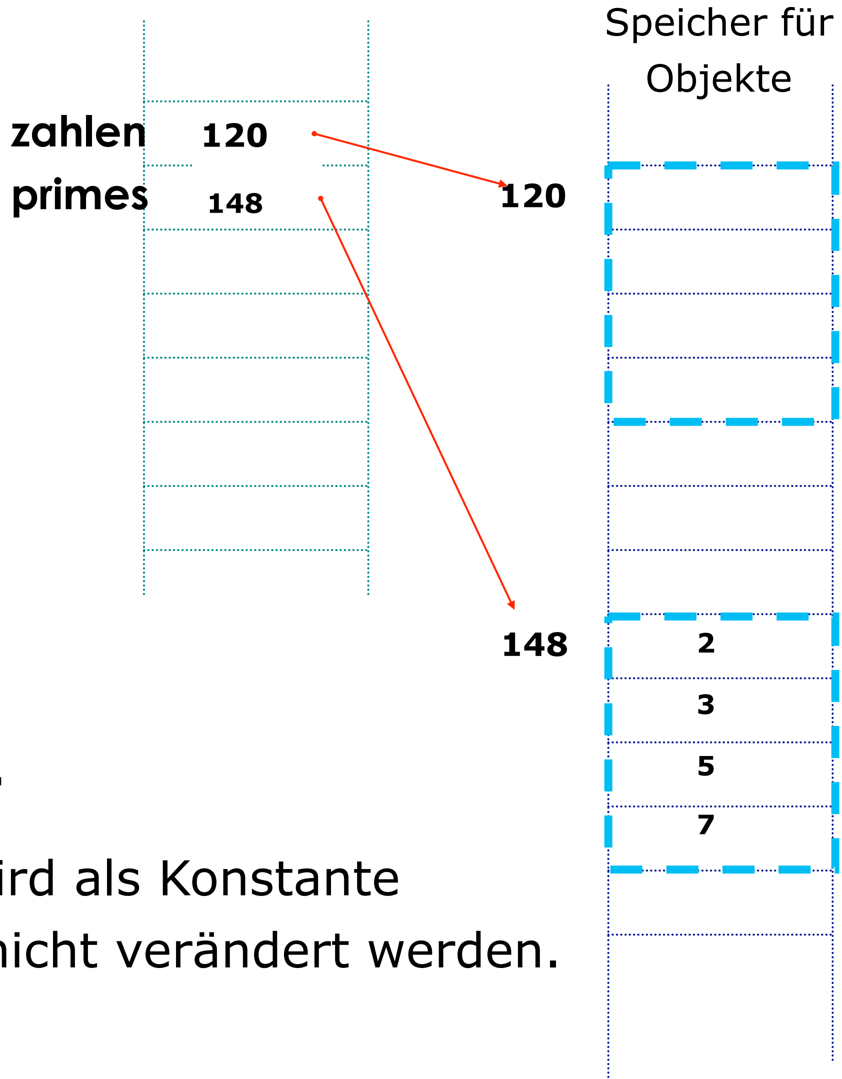
Arrays können beliebige Datentypen enthalten, auch weitere Arrays.

Arrays

```
int[] zahlen;

zahlen = new  int[4];

int[] primes = { 2, 3, 5, 7 };
```



primes. length → **4**

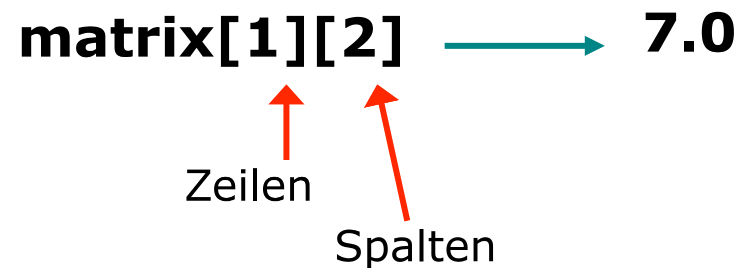
Das "length"-Attribut des Arrays wird als Konstante generiert und kann gelesen, aber nicht verändert werden.

Arrays

Mehrdimensionales Array: Array von Arrays.

```
double[][] matrix = new double[3][3];
```

$$\begin{bmatrix} 2.0 & 3.0 & 1.0 \\ 6.0 & 0.0 & 7.0 \\ 2.0 & 8.0 & 4.0 \end{bmatrix}$$




```
double[][] matrix = new double[3][3];
```

Arrays

```
java Main  args[0]  args[1] ...
```

```
public class Main {
    public static void main( String[] args ) {
        if ( args.length > 2 ) {
            System.out.print( args[0] );
            System.out.print( args[1] );
            System.out.print( args[2] );
        }
    }
} // end of class Main
```

```
java Main  eins  zwei  1000
```

Ausgabe:

```
 eins  zwei  1000
```

Vererbung

Ein wesentliches Merkmal objektorientierter Sprachen ist die Möglichkeit, Eigenschaften vorhandener Klassen auf neue Klassen zu übertragen. (**Wiederverwendbarkeit**)

Durch Hinzufügen neuer Elemente oder Überschreiben der vorhandenen kann die Funktionalität der abgeleiteten Klasse erweitert werden.

Was ist **Vererbung**?

Übernahme aller Bestandteile einer Klasse in eine Unterklasse, die als **Erweiterung** oder **Spezialisierung** der Klasse definiert wird.

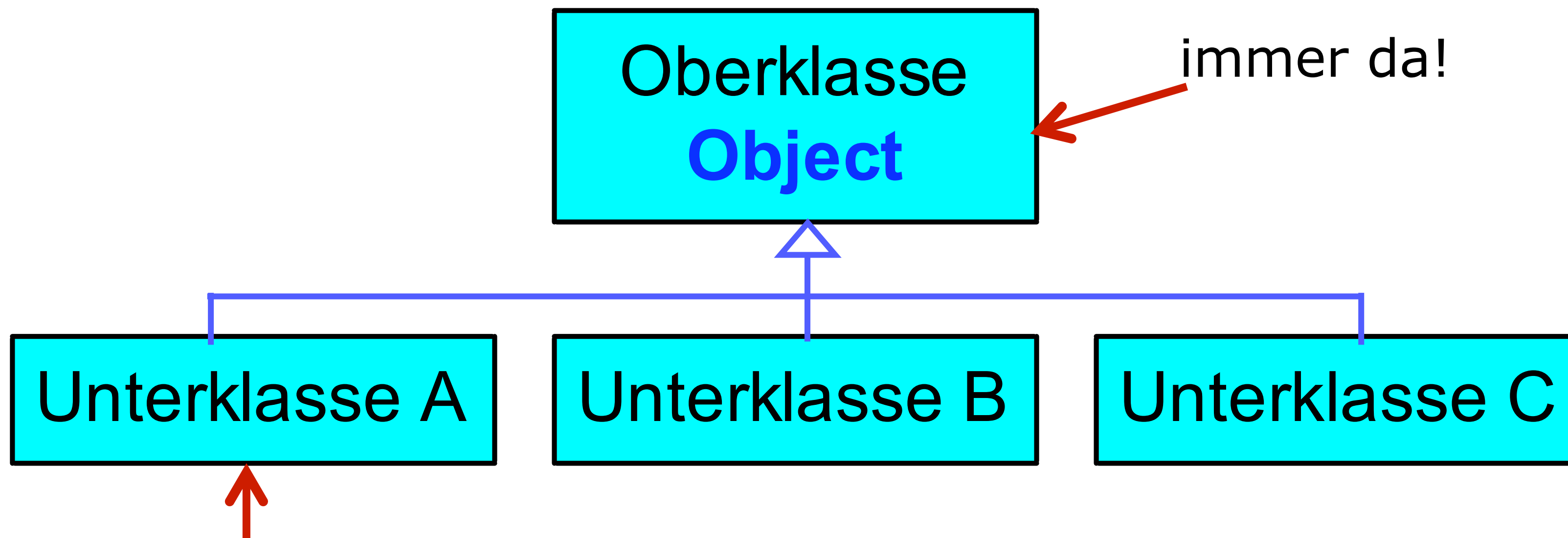
Vorsicht!

Die Unterklasse besitzt alle Eigenschaften und Methoden ihrer Oberklasse

+

die Erweiterungen, die in der Unterklasse selber definiert worden sind.

Klassenhierarchie



Die Unterklasse A besitzt alle Eigenschaften
und Methoden ihrer Oberklasse

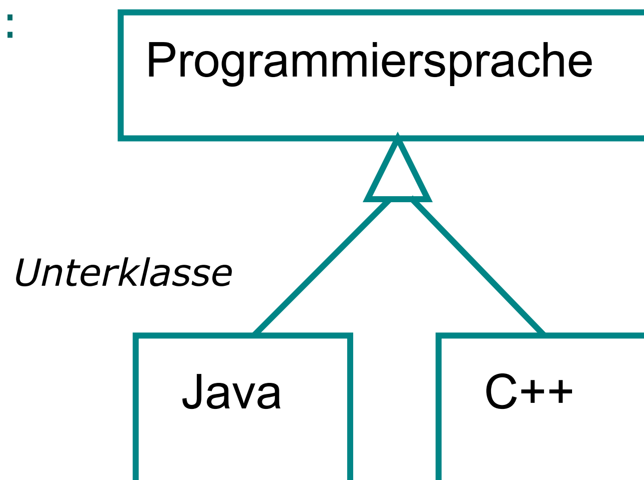
+

die Erweiterungen, die in der Unterklasse A
definiert worden sind.

Verfeinerung und Verallgemeinerung

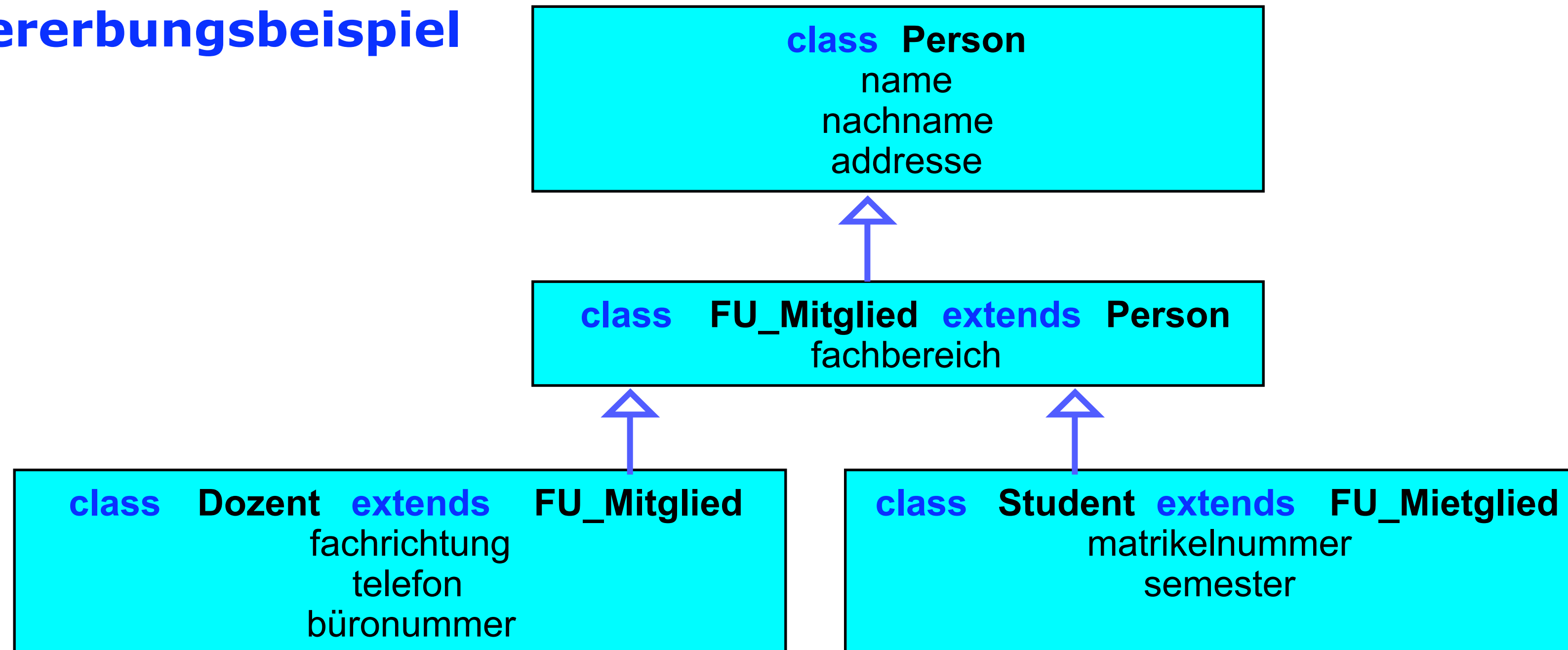
Verfeinerungen sind Beziehungen zwischen gleichartigen Elementen unterschiedlichen Detaillierungsgrades.

Beispiel:



Die Spezialisierung wird als Generalisierungs-Pfeil dargestellt. Er zeigt in Richtung der allgemeineren Klasse.

Vererbungsbeispiel



Dozent

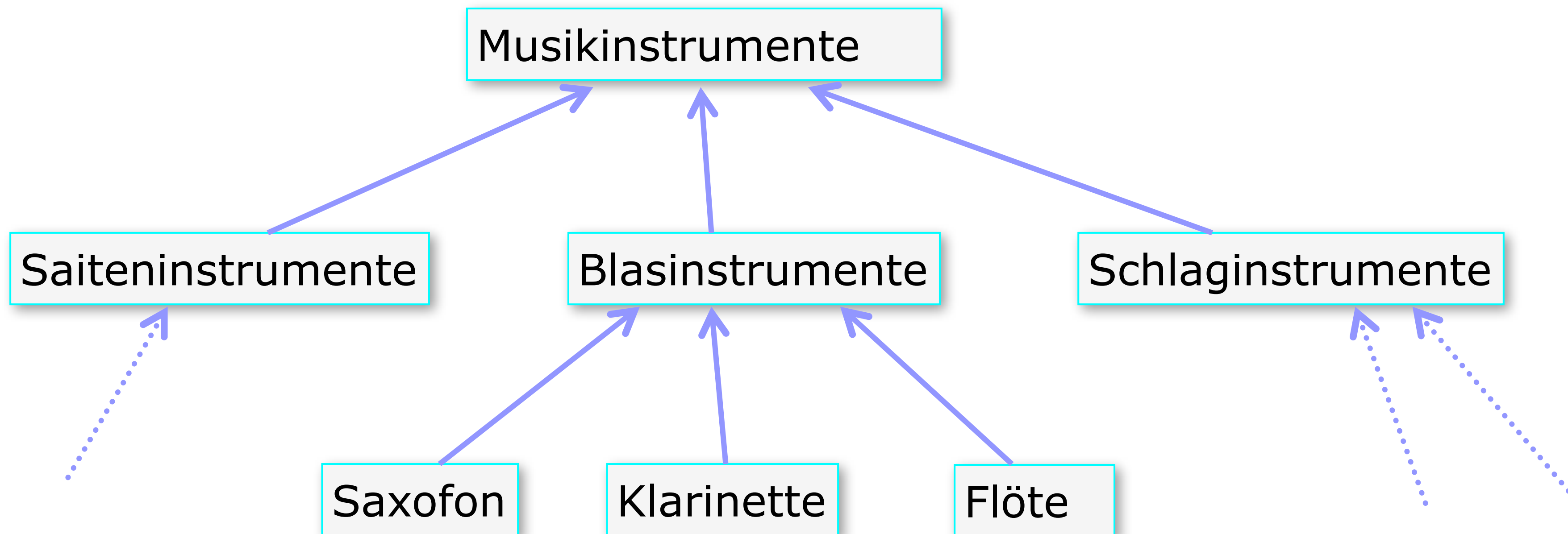
name
nachname
addresse
fachbereich
fachrichtung
telefon
büronummer

Student

name
nachname
addresse
fachbereich
matrikelnummer
semester

Die Methoden werden auch vererbt.

Klassifizieren von Objekten



Bei guten Modellierungen klingt es logisch, wenn gesagt wird, dass ein Element der Unterklasse auch ein Element aller ihrer Oberklassen ist.

Eine Flöte ist ein Blasinstrument

Eine Flöte ist ein Musikinstrument

Vererbung

FU_Mitglied ist eine Unterklasse

"Spezialisierung "

"Erweiterung"

"is-a-Relation"

"Ableitung"

von **Person**

- Die Kunst ist es, eine möglichst gute Klassenhierarchie für die Modellierung von Softwaresystemen zu finden.
- OOP ist keine Religion.
- Nicht alle Teile eines Problems können gut mit rein objektorientierten Techniken gelöst werden.
- Nicht immer gelingt es, eine saubere Klassenhierarchie zu finden!