

# **Objektorientiertes Programmieren**

Wintersemester 2007/08  
Hochschule Furtwangen University  
Wirtschaftsinformatik  
Studiengang WNB

# Inhaltsverzeichnis

1. Programmieren .....	3
1.1. Erstellung eines Programmes .....	3
2. Syntax eines Programmes .....	3
2.1. Java Syntax.....	3
2.2. Beispiel für ein Programm .....	4
3. Datentypen .....	4
3.1. Ganzzahlige Datentypen .....	4
3.2. Gleitkommazahlen .....	5
3.3. Datentypen für Zahlen.....	5
3.3.1. Hexadezimalzahlen .....	5
3.4. String .....	6
3.5. Beispiel für ein Programm .....	6
4. Bezeichner und Namen .....	7
5. Kommentare .....	7
6. Eingabe und Ausgabe.....	8
7. Operatoren .....	8
7.1. Zuweisungsoperatoren .....	8
7.2. Vergleichsoperatoren .....	8
7.3. Booleschen Operatoren .....	8
8. Ablaufsblock .....	9
8.1. Anweisungsblock .....	9
8.2. Notation für Anweisungsblock.....	9
8.3. Entscheidungsanweisungen.....	10
9. Verzweigungsbefehle .....	10
9.1. If/Else .....	10
9.2. Switch.....	10
10. Loops .....	11
10.1. Die for-Schleife .....	11
10.2. Die while-Schleife.....	11
10.3. Die do-Schleife.....	12
11. Felder.....	12
11.1. Felddefinition .....	12
12. Klassenkonzept.....	15
13. Überladen von Methoden .....	17
13.1. Rekursive definierte Methode .....	18
14. $N!$ = Fakultät .....	18
15. Vererbungen .....	19
16. Constructor .....	22
16.1. Aufbau eines Constructors .....	22
16.2. Konstanten.....	22
16.3. Überladen von Konstruktoren .....	22
17. Softwareprojekte .....	24

# 1. Programmieren

Programmieren bedeutet Anweisungen für den Computer schreiben.

Computer sind schwer von Begriff, deshalb ist eine exakte Syntax notwendig

→ führt zu besserer Software

## 1.1. Erstellung eines Programmes

1. Programmiersprache (z.B. Delphi, Java, Visual Basic, Cobol, C++, Pascal, Fortran)
2. Editor (z.B. Notepad, Textpad, Eclipse)  
→ Programmcode schreiben und speichern (Dateiname.java)
3. übersetzen (compilieren)
4. Programm starten (JVM = Java Virtual Machine)  
→ JVM läuft auf gewünschtem OS  
→ Programm läuft auf allen Rechnern  
→ wenig Programmabstürze dank sehr guter Fehlerkontrolle  
→ genaue Dateneingabe

## 2. Syntax eines Programmes

```
public class Berechnung {  
    public static void main (String[] args) {  
        int I;  
        I = 3 + 4;  
        System.out.println(I);  
    }  
}
```

### 2.1. Java Syntax

Trennzeichen trennen für den Rechner einzelne Elemente (Wortzwischenräume)

- Leerzeichen (eins oder mehrere)
- Neue Zeile/New Line
- Tabulator (unterschiedliche Einrückungen, deshalb eher ungeeignet für das Programmieren)

Wo ein Trennzeichen steht, dürfen auch viel (auch gemischt) stehen.

## 2.2. Beispiel für ein Programm

```

/*****
* Berechnung und Ausgabe von 7
*
* @author Melanie Lubjuhn
* @date 2007-10-17
* @version 1.0
*****/

Public class Berechnung {
    Public static void main (String []args) {
        int = I;
        I = 3 + 4;
        System.out.println(I);
    }
}

```

---

Konsolenfenster:  
7  
Programm beendet

---

In die Ausgabe soll nun mehr Text eingebaut werden...

```

...
...
...
...
        System.out.println(„Ergebnis:“ + I);
    }
}

```

---

Konsolenfenster:  
Ergebnis:7  
Programm beendet

---

## 3. Datentypen

### 3.1. Ganzzahlige Datentypen

Länge	Kleinster Wert		Größter Wert
Byte (1 Byte)	- 128	0	127 (255 – 128)
Short (2 Byte)	- 32.768	0	32.767
Integer (4Byte)	- 214.783.648	0	214.783.647
Long (8Byte)	- 9.223.372.036.854.775.808	0	9.223.372.036.854.775.807

### 3.2. Gleitkommazahlen

= Möglichkeit, Kommas darzustellen und Stellen nach dem Komma zu bestimmen. Die interne Darstellung besteht aus folgenden Komponenten:

+ oder -	1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 1
Vorzeichen	Zahlenwert	Vorzeichen der Potenz	Potenz Exponent

- 0.0
- 1.0
- 58531259.0
- 58.571
- 3.14159

!! Achtung !!

In der Buchhaltung: Java würde 1.000 als 1 interpretieren, da Java den Punkt als Abtrennungspunkt liest.

Integer:  $1 : 2 = 0$ , da nur ganze Zahlen  
 Gleitkommazahlen:  $1 : 2 = 0.5$

### 3.3. Datentypen für Zahlen

char b;  
 intern mit 2 Byte gespeichert  
 16 Bit  $\rightarrow 2^{16}$  Zeichen  $\rightarrow 65.356$  verschiedene Zeichen  
 Unicode: Darstellung „aller“ Zeichen in Schriftsprachen

Normale Zuweisung:  
 b = 'z' (90)

Alternative Zuweisung:  
 Darstellung als Hexadezimalwert  
 b = / u 005A  $\rightarrow$  Hexadezimalwert

#### 3.3.1. Hexadezimalzahlen

0 1 2 3 4 5 6 7 8 9 A B C D E F  
 Stellensystem zur Basis  $2^4 = 16$

Faktor	$16^3$	$16^2$	$16^1$	$16^0$
Dezimal	4.094	256	16	1
90	0	0	5	A
255	0	0	F	F

FF ist der größte Wert im Hexadezimalsystem mit zwei Ziffern (kommt oft bei Farben vor).

### 3.4. String

= Festlegung von Zeichenketten

```
String mein Satz;  
mein Satz = „Das ist ein Text“;
```

#### Wahrheitswert

Boolean *Wahrheit*

*Wahrheit* = true;

*Wahrheit* = false;

Typumwandlung:

```
int klein = 1;
```

```
float groß;
```

```
groß = (float)klein;
```

```
klein = (int)groß;
```

### 3.5. Beispiel für ein Programm

```
Public class Berechnung2 {  
    Public static void main (String []args) {  
        int result;  
        float mwst = (float)0.19;  
        float netto;  
        float brutto;  
        String textblock = "result";  
        String curenacy = "Euro";  
        netto = (float)12.34;  
        brutto = netto + netto * mwst;  
        System.out.println(textblock + " = " + brutto);  
    }  
}
```

Oder:

```
System.out.println("Nettopreis =" + netto + " " + currency)
```

```
System.out.println("mwst Satz = " + mwst + " ")
```

```
System.out.println("Brutto = " + brutto + " " + currency)
```

→ auf logische Reihenfolge achten!!

---

Konsolenfenster:

Nettopreis = 12.34 Euro

mwst Satz = 0.19

Brutto = 14.6846 Euro

Programm beendet

---



## 6. Eingabe und Ausgabe

Es gibt eine Toolbox:

Prog1Tools.IOTools (Input/Output Tools)

Prog1Tools = Paket mit Hilfsprogrammen

IOTools = Klasse für Ein - & Ausgabe

import Prog1Tools.IOTools

Einlesen eines Integer von der Tastatur

`i = IOTools.readInteger( )`

Einlesen eines Double von der Tastatur

`i = IOTools.readDouble ( )`

## 7. Operatoren

### 7.1. Zuweisungsoperatoren

`+`            `a + b`            oder: `a = a + b` → `a += b`

`-`            `a - b`

`*`            `a * b`

`/`            `a / b`            bei Integer: Rest/Nachkommawert wird weggelassen

Achtung bei der 0: Keine Division durch 0!!!

Beispiel: `5 / 2 = 2`

`%`            `a % b`            Rest der Division

Beispiel: `5 % 2 = 0,5`

`a = a + 1`            Der Term `a + 1` wird `a` zugewiesen

Abkürzung: `a ++`

`a = a - 1`            Der Term `a - 1` wird `a` zugewiesen

Abkürzung: `a --`

### 7.2. Vergleichsoperatoren

Gleichheit:

`a == b`            true/false

`c = (a == b)`            `c` muss boolean sein, d.h. entweder true oder false

`a > b`            `a` größer `b`

`a < b`            `a` kleiner `b`

`a <= b`            `a` kleiner gleich `b`

`a >= b`            `a` größer gleich `b`

`a != b`            `a` ungleich `b`

### 7.3. Boolschen Operatoren

a	b	a & b (und)	a   b (oder)	!a (ungleich)
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Aufgabe:

Zahl soll zwischen 10 und 20 liegen. C soll den Wert true bekommen, sonst false.

Lösung:  $c = (a > 10) \ \& \ (a < 20)$

Beispiele:

Für  $a = 10$ : false, true  $\rightarrow$  false

Für  $a = 11$ : true, true  $\rightarrow$  true

Für  $a = 21$ : true, false  $\rightarrow$  false

## 8. Ablaufsblock

### 8.1. Anweisungsblock

$\rightarrow$  Anweisung 1

$\rightarrow$  Anweisung 2

$\rightarrow$  Anweisung 3

$\rightarrow$  etc.

$\rightarrow$  über 18  $\rightarrow$  nein  $\rightarrow$  kein Eintritt

$\rightarrow$  Kino

$\rightarrow$  etc.

### 8.2. Notation für Anweisungsblock

```
{ Anweisung 1;
```

```
  Anweisung 2;
```

```
  Anweisung 3;
```

```
}
```

Wichtiger Hinweis: Im Anweisungsblock gelten lokale Variablen, außerhalb nicht!

```
{ Anweisung 1;
```

```
  Anweisung 2;
```

```
  Anweisung 3;
```

```
  If (Frage/Antwort)
```

```
    { Kino; }
```

```
  Else
```

```
    { kein Eintritt; }
```

```
  Normal;
```

```
}
```

### 8.3. Entscheidungsanweisungen

Entscheidungen sind eindeutig. Entweder true oder false.

```

if (logischer Ausdruck)
{ wird ausgeführt, falls der logische Ausdruck true ergibt.
  Anweisung;...
}
else
{ wird ausgeführt, falls der logische Ausdruck false ergibt.
  Andere Anweisung;...
}

```

## 9. Verzweigungsbefehle

### 9.1. If/Else

```

if (logischer Ausdruck) {
<< Anweisung >> }
else {
<< Anweisung 2 >>
}

```

### 9.2. Switch

→ bei mehreren Fällen (> 2 Fälle)

```

switch (<< logischer Ausdruck >> ) {
  case Konstante:
    << Anweisungen >>
    Break;
  Case Konstante:
    ...
    ...
    ...
  default:
    << Anweisungen >>
}

```

} alle Fälle werden abgearbeitet, wenn nichts anderes vereinbart wurde

→ ist nach dem ersten Fall die Anweisung erfüllt, bewirkt der **break**, dass das Programm die anderen Fälle nicht auch noch abarbeitet.

Beispiel:

```
int a = 2
switch (a) {
    case 1:
        System.out.println("kleine Zahl");
        break;
    case 2:
        System.out.println("Sie haben gewonnen");
        break;
    case 3:
    default:
        System.out.println("Nix los!");
}
```

→ für a = 2 wird die Anweisung erfüllt, das Programm wird durch den break beendet, die nächsten Anweisungen werden nicht mehr abgearbeitet.

## 10. Loops

### 10.1. Die for-Schleife

```
for ( << Initialisierung >>; << logischer Ausdruck >>; << updateoperation >> ) {
    << Anweisungen >>
}
```

Beispiel:

```
for ( int i = 0; i < 10; i + + )
{
    System.out.println("Ich bin ein Computer" + i);
}
```

### 10.2. Die while-Schleife

```
while ( << logischer Ausdruck >> ) {
    << Anweisungen >>
}
```

Beispiel:

Es soll die Zahl 7 erraten werden.

```
int a = 0;
while (a != 7) {
    System.out.println („Bitte geben Sie eine Zahl ein:“);
    IO.readInteger(a);
}
System.out.println („Sie haben die Zahl erraten!“);
```

Erläuterung:

„Bitte geben Sie eine Zahl ein:“ erscheint so lange, bis die Zahl 7 eingegeben wird, also für a = 7. Wird für a die 7 eingegeben, ist die Anweisung a != 7 false und es erscheint „Sie haben die Zahl erraten!“.

Innerhalb der while-Schleife muss die Testvariable verändert werden, sonst gibt es eine Endlosschleife.

### 10.3. Die do-Schleife

```
do {
<< Anweisung >>           // die Anweisung wird immer ausgeführt
while (<<logischer Ausdruck >>)
```

Beispiel:

Login

```
do {
System.out.println („Login eingeben:“);
a = IOTools.readChar ( );
}
While (a != „I“);           //eingeloggt sobald a = I
System.out.println („Sie sind eingeloggt“);
```

## 11. Felder

Problem: mehrere ähnliche Variablen

String Stunde 1;

String Stunde 2;

...

...

String Stunde 23;

String Stunde 24;

### 11.1. Felddefinition

```
<< Datentyp >> [ ] << Variablenname/Feldname >>
```

Beispiel:

String [ ] Wochentag;

Wochentag = new String [7]

\_\_\_\_\_ 0 (Fach: 0)

\_\_\_\_\_ 1

\_\_\_\_\_ 2

\_\_\_\_\_ 3

\_\_\_\_\_ 4

\_\_\_\_\_ 5

\_\_\_\_\_ 6

Beispiel:

```
String [ ] Jahreszeit = { "Frühling", "Sommer", "Herbst", "Winter"};
System.out.println(Jahreszeiten[1])
```

```
int Nummer;
Nummer = 1;
System.out.println(Jahreszeiten[Nummer]);    // zeigt Sommer an
Nummer = 0;
System.out.println(Jahreszeiten[Nummer]);    // zeigt Frühling an
```

Lösung:

```
public class Jahreszeiten {
    public static void main (String []args) {
        for (int Nummer = 0; Nummer <= 3; Nummer + +) {
            System.out.println(Jahreszeiten[Nummer]);
        }    // for-Schleife
    }    // main
}    // Jahreszeiten
```

### 1-dimensionale Felder (1D)

Anzahlen der Zellen:  $z = n + 1$

0
Datentyp
n

### 2-dimensionale Felder (2D)

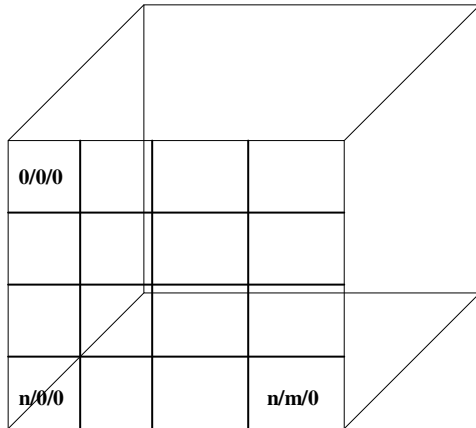
Anzahlen der Zellen:  $z = (n+1) * (m+1)$

0/0					0/m
		1/j			
n/0					



### 3-dimensionale Felder (3D)

Anzahlen der Zellen:  $z = (n+1) * (m+1) * (p+1)$



Jahreskalender:

$$z = (n+1) * (m+1) * (p+1)$$

$$z = 24 * 31 * 12 = 8928$$

Java Syntax für Felder:

2D String [ ] [ ] termine

2 D Integer:

Int [ ] [ ] Zahlenfeld

Wir haben einen Zeiger auf ein 2D Feld

Initialisierung der Feldliste erfolgt über einen Pointe 2D:

```
termine = new String [31] [ ];
```

0	...	...	...	...	...	30
---	-----	-----	-----	-----	-----	----

Initialisierung der Tagesfelder:

```
termine [0] = new String [24]
```

```
termine [1] = new String [24]
```

```
termine [2] = ...
```

...

...

```
termine [30] = new String [24]
```

Besser: for-Schleife verwenden!

```
for (int i = 0; i < termine.length; i++) {
  termine [i] = new String [24];
}
```

Erweiterung:

Jede Zelle wird bei der Initialisierung mit einem Startwert versehen.

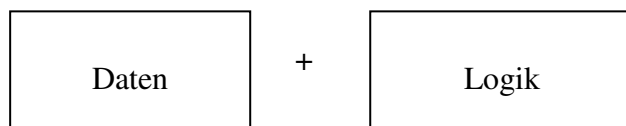
```
for (int i = 0; i < termine.length; i++) {
    termine [i] = new String [24]
    for (int j = 0; j < termine.length, j++)
        termine [i][j] = " ";
```

## 12. Klassenkonzept

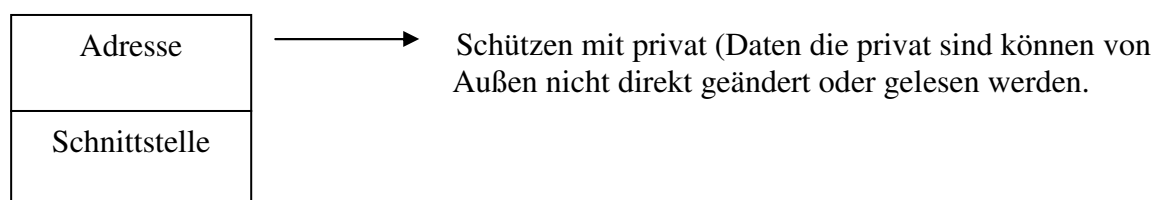
Begründung:

- Programme müssen strukturiert werden damit die Übersicht bewahrt wird
- möglichst gut gekapselt

Programme bestehen aus:



### Beispiel für datenorientierte Klassen



- Information an die Klassen abgeben
- Informationen holen
- Verarbeitung

### Einlese Methode

Set Methoden

Set Name (Übergabedaten) → Methodenname

{

Ablegen der Information

Ggf mit Test und Verarbeitung

}

Auslese Methode

Get Name → Datentyp

### Syntax:

Vereinbarung einer selbstständigen Klasse (in einer eigenen Datei)

```
public class << Klassenname >> {
    << Variablenname >>
```

Methoden Definition innerhalb einer Klasse.

```
Public static << Rückgabewert >> << Methodenname >> (<< Parameterliste >>) {  
}
```

Beispiel:

Eine Methode, die zwei Zahlen multipliziert.

```
public static double multiplizieren (double x, double y) {  
    double ergebnis;  
    ergebnis = x*y;  
    return ergebnis;  
}
```

In Java:

```
import Prog1Tools.IOTools;  
public class Methodenbeispiel {  
    public static void main (String [b]args) {  
        double a = 1,7;  
        double b = 3,14;  
        double erg;  
        System.out.println ("a=" + a);  
        System.out.println ("b=" + b);  
        System.out.println ("a=" + a);  
    }  
}
```

// Bauen einer Methode

```
public static double multi(double x, double y) {  
    double rückgabewert;  
    rückgabewert = x*y;  
    return rückgabewert;  
    } // multi  
    } // main  
} // Methodenbeispiel
```

## 13. Überladen von Methoden

- gleiche Methodennamen
- verschiedene Parameter und Datentypen

Beispiel:

Methode zum Bestimmen des Maximums:

```
public static double max(double x, double y) {
    return (x > y) ? x : y;
```

```

      ↓   ↓   ↓
    Test falls falls
           true  false

```

```

}
Public static int max (int x, int y) {
    return (x > y) ? x : y;
}

```

In der main:

```

Int a, b, c;
a = 10, b = 20;
c = max(a,b);

```

```

public static int max (int x, int y, int z) {
    int m = (x > y) ? x : y;
    return (m > z) ? m : z;
}

```

Beispiel:

```

public class AufrufTest {
    public static void main Unterprogramm (int [ ] n) {
        n[0] = n[0] + 5;           // n [0] = 7, also 7 + 5
        System.out.println(n[0]) // 12 wird ausgegeben
    }                             // Unterprogramm Ende
// main: hier startet das Programm
    public static void main (String []args) {
        int [ ] n = {7};         // Übergabe des Feldelements
        Unterprogramm(n);        // Aufruf des Unterprogramms
        System.out.println(n);   // 7 wird ausgegeben
    }                             // main Ende
}                                 // Aufruf Test Ende

```

### 13.1. Rekursive definierte Methode

```
public class Unendlich {
    public static void ausgabe () {
        System.out.println ("Sehr oft");
        ausgabe ();
    }
    public static void main (String []args) {
        ausgabe ();
    }
}
```

→ Endlosschleife, es wird immer „sehr oft“ gedruckt, da nirgends eine Bremse eingelegt ist.

### 14. N! = Fakultät

$$\begin{array}{l} n = 0 \rightarrow n! = 1 \\ n! = n * (n - 1) \end{array}$$

für  $n = 3$ :

$$n! = 1 * 2 * 3$$

Um Fakultäten auszurechnen ist es besser mit double zu arbeiten. Mit Integer ist es nur bis 13! möglich, da sonst Überlaufzahlen entstehen.

```
public class int fakultaet (int m) {
    if (n == 0) // Vergleichsoperator, kann 2 Werte haben: true/false
        return 1;
    else
        return n * fakultaet (n - 1);
}
```

## 15. Vererbungen

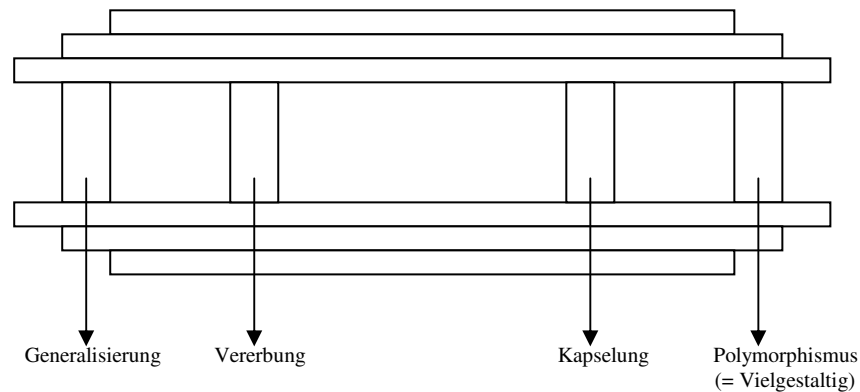
Wir leben in einer Welt der Objekte

- Objekte haben Eigenschaften
- Objekte kommen immer aus Klassen

Wahre Welt  $\leftrightarrow$  virtuelle Welt

Beispiel Adresse:

Name;  
Straße;  
Ort;



### Generalisierung:

z.B. Säugetiere

mindestens eine *übergordnetet Eigenschaft*

Säugetiere säugen ihre Jungen

Problem. Ausgangspunkt sind zu wenig Kenntnisse über die Objekte

- häufig Generalisierungsfaktoren

Lösung:

- neu strukturieren
- neue Generalisierung (Klassen)

### Vererbung:

Eltern (parent) = Superklasse

Kinder (child) = Klasse

→ Kinder erben alle Eigenschaften der Eltern und haben auch neue/eigene Eigenschaften.

### Ontologie:

Struktur in Fragestellungen

→ Systematik der Struktur ist eine Ontologie

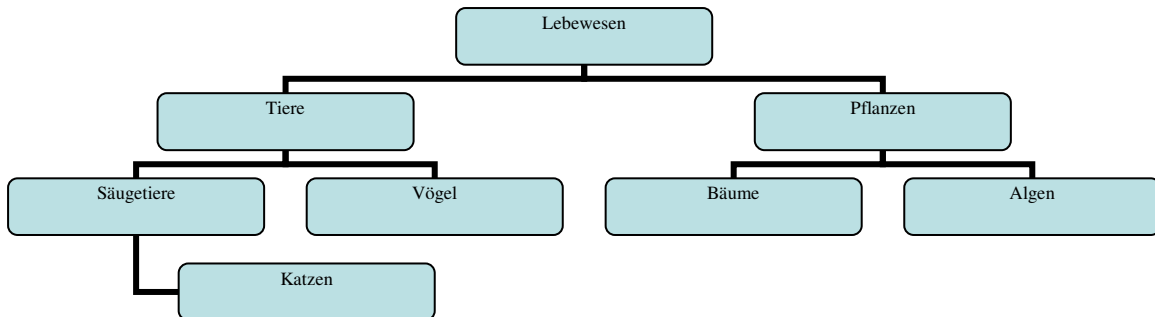
### Kapselung:

Jedes Element soll abgeschlossen sein

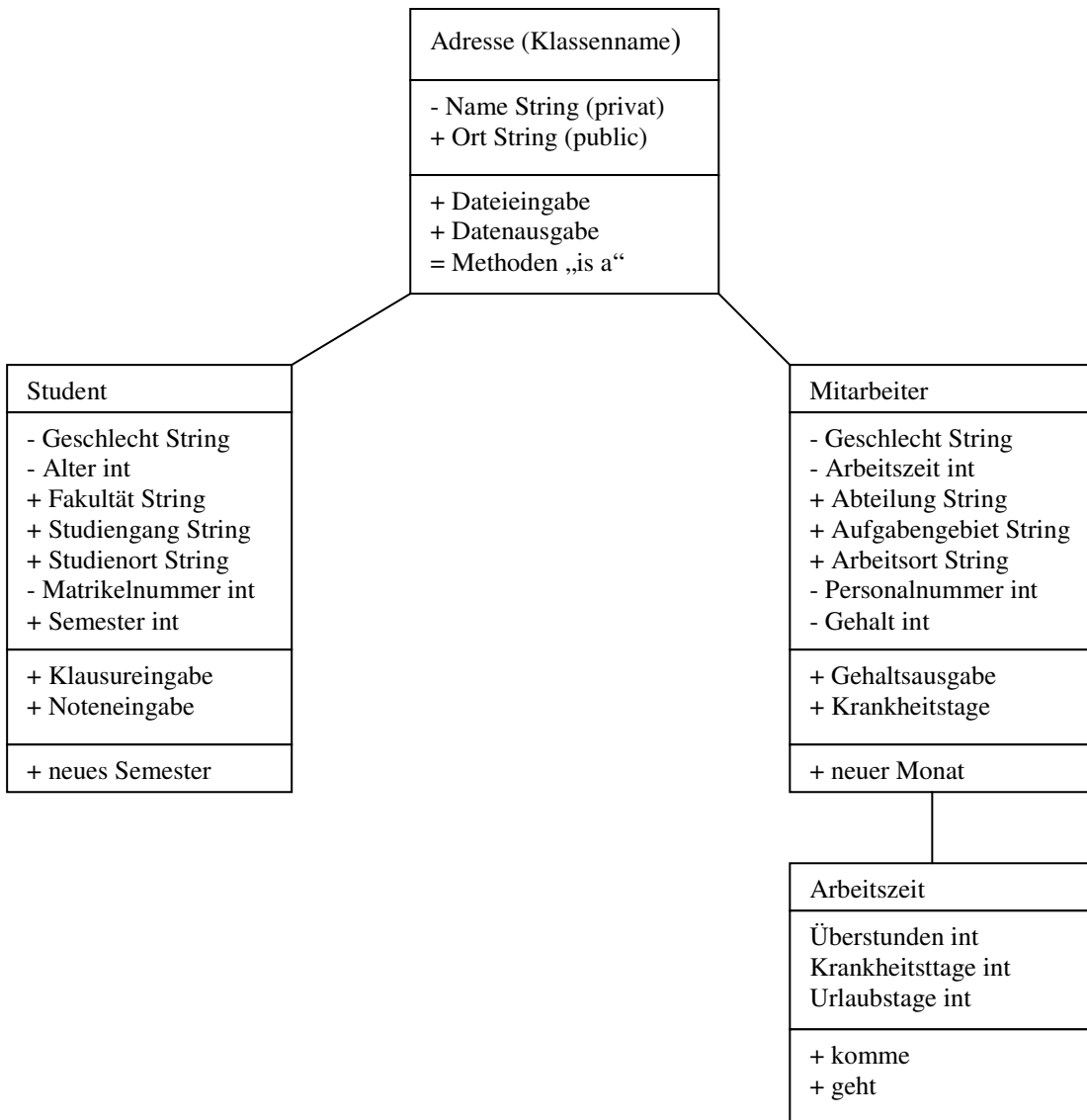
- innerer Bereich
- Schnittstelle → von Außen sichtbar und ansprechbar

UML Notation (Unified Modelling Language):

- einheitliche grafische Darstellung von Prozessen und Elementen (in der Programmierung)
- verschiedene Diagrammtypen ~ 15
- international leichter verständlich

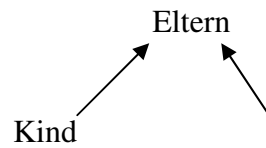


Klassendiagramm:

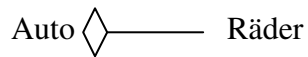


UML für Klassen

„is a“



„part of“



Klassenname
Variablen
- privat
+ public
Methoden
+ public (get/set Konvention)
+ set Matrikel ( )
- Zähler

→ statische Elemente unterstreichen

Hinweis: privat Variablen werden oft weggelassen um die Übersicht zu erhalten.

Speicher:

Zunächst steht keine Information im Speicher.

→ damit ein Objekt entsteht muss dieses erzeugt werden

→ erzeugen mit new

Variable = new Constructor

Student1 = new Student ( );

↓  
Objekt

↓  
Klassenname

## 16. Constructor

### 16.1. Aufbau eines Constructors

Entspricht einer Methode der Klasse die immer bei Initialisierung aufgerufen wird.

Methodenname = Klassenname

```
student ( )
```

```
static privat int Matrikelzähler
```

```
privat int Matrikelnummer
```

```
...
```

```
...
```

```
student {
```

```
    Initialisierung → Constructormethode
```

```
    Matrikelnummer = Matrikelzähler
```

```
    Matrikelzähler ++
```

```
    }
```

```
}
```

Für einen sinnvollen Inhalt, alle Variablen mit Werten versehen (default Werte), z.B. automatisch eine Matrikelnummer erzeugen.

### 16.2. Konstanten

- Der Wert bleibt immer gleich
- Der Wert kann nicht geändert werden

```
public static final int WNB = 2
```



Namen der Konstanten  
bestehen aus Großbuch-  
staben

```
public static final int POSTLEITZAHLFURTWANGEN = 78120
```

### 16.3. Überladen von Konstruktoren

Klasse Student als Beispiel:

```
...
```

```
privat int geburtsjahr
```

```
public Student (int geburtsjahr) {
```

```
    this.geburtsjahr = geburtsjahr;
```

```
}
```

```
public int getGeburtsjahr ( ) {
```

```
    Return geburtsjahr;
```

```
}
```

Aufbau der Klasse Person

```
public class Person {
```

```
    privat int geburtsdatum;
```

```
    public Person (int geburtsdatum) {
```

```
        this.geburtsdatum = geburtsdatum;
```

```
}  
  
public class Super {  
    public String x = "vor Superkonstruktor";  
    public Super () {  
        System.out.println ("Superkonstruktor");  
        System.out.println ("x=" + x);  
        x = "nach Superkonstruktor";  
        System.out.println ("Superkonstruktor");  
        System.out.println ("x=" + x);  
    }  
} // Super  
public class Sub extends Super {  
    public String y = "vor Superkonstruktor"  
  
    public Sub () {  
        System.out.println ("Superkonstruktor");  
        System.out.println ("x");  
        System.out.println ("y");  
        x = "nach Superkonstruktor";  
        y = "nach Superkonstruktor";  
        System.out.println ("Subkonstruktor beenden");  
        System.out.println ("x");  
        System.out.println ("y");  
    } // Sub
```

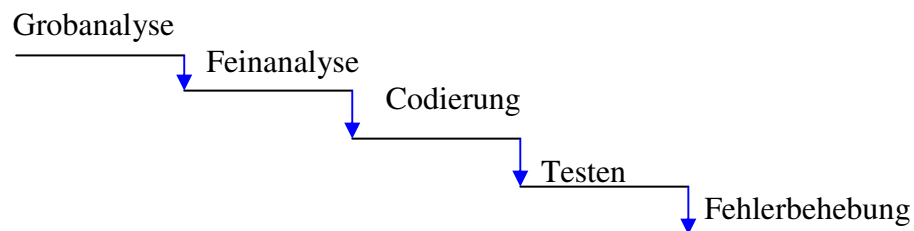
## 17. Softwareprojekte

### 1. Aufwandsschätzung

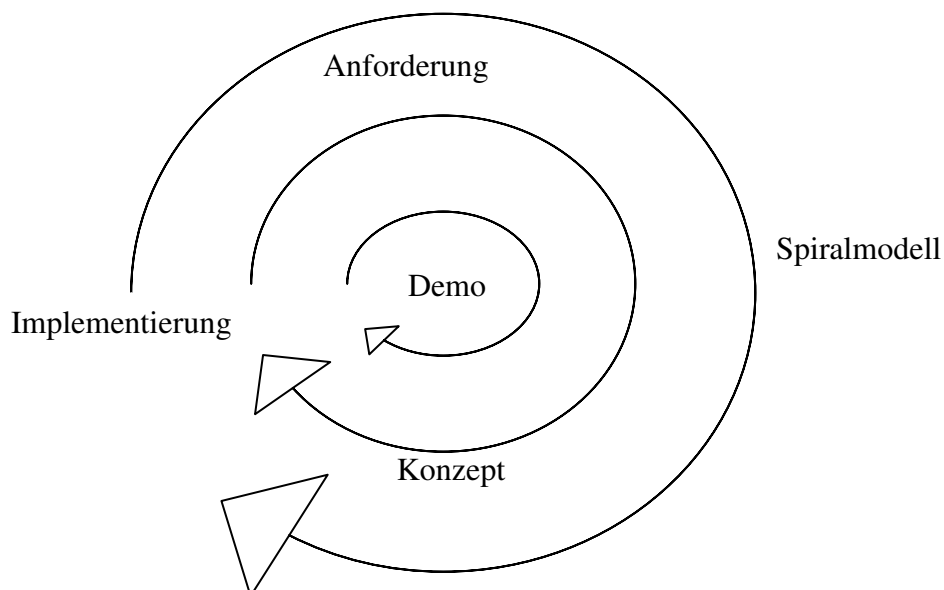
- a. detaillierte Problemanalyse
- b. Fragestellung (richtig verstehen)
- c. Interviews (die einzelnen Punkte durchgehen/offene Fragen vertiefen, d.h. was genau wird gemacht?)
- d. Entwicklung eines Lösungskonzepts (nun kann der eigene Aufwand in etwa eingeschätzt werden)
  - Schätzverfahren: LOC = Line of Code (Wie lange wird das Programm?)
  - Zählen der Detailleistungen/Features (z.B. Eingabe, Ausgabe, Berechnung, etc.)
  - Programmierleistung ca. 3 – 5 Zeilen/h
- e. Analyse z.B. Klassenaufbau
- f. Codieren
  - Aufbau der Kommentare
  - Test des Programms
  - Debugging (Fehlersuche- und behebung)

### 2. Vorgehensmodell

- a. Wasserfallmodell



### 3. Entwicklungszyklus



## Agile Entwicklung

### Xtrem Programmierung

- Zweier Teams
- ein Rechner
- einer tippt
- einer schaut auf Fehler
- mehrere Teams an einem Projekt
- häufige Builds → Qualität wächst

### 4. Fehlerquellen

- Software hat meist eine geringe Fehlertoleranz
  - Fehler auffangen, damit es nicht zum Programmabbruch kommt
  - vorab Fehlerquellen erkennen
  - externe Quellen
    - Benutzer (freie Eingabe)
    - Zugriff auf Internet-Ressourcen
  - Interne Fehler
    - Logikfehler
      - alle Fehler der Logik kennen
    - zu komplex
      - zerlegen in Teilaufgaben
    - unerwartete Wertebereiche
      - vorausschauend
      - Wertebereich definieren
    - undokumentierte Änderungen
      - Information im Kommentar und im Code sind nicht zusammenhängend
      - jede Änderung ernst nehmen
      - sprechender Code → Namen der Variablen, Klassen,...
      - Sprachmix vermeiden → alles in englisch
      - keine Sprachabhängigen Sonderzeichen
- gute Entwicklungstools verwenden
- übersichtliche Eingabe
  - Kooperation unterstützen
  - flexibel bei Spracherweiterungen
  - Versionierung