

# OOP2

## Objektorientierte Programmierung unter C++, Java und PHP

<http://worgtsone.scienceontheweb.net/worgtsone> – <mailto:worgtsone@hush.com>

Mitarbeit: Tobias Langbein (danke toby)  
13. Oktober 2011

### Inhaltsverzeichnis

<b>1</b>	<b>Intro</b>	<b>5</b>
1.1	OOP in anderen Lehr-Unterlagen . . . . .	5
1.2	Under Construction . . . . .	5
1.3	Vorsicht mit <code>pdflatex</code> . . . . .	5
<b>2</b>	<b>Theorie und Terminologie der OOP</b>	<b>6</b>
2.1	Was ist ein Objekt ? . . . . .	6
2.2	Was ist eine Klasse? . . . . .	6
2.2.1	Beispiele . . . . .	6
2.3	Was ist in einer Klasse? . . . . .	6
2.4	Was ist Abstraktion? . . . . .	6
2.5	Was ist Modellierung? . . . . .	7
2.6	Was ist Vererbung? . . . . .	7
2.7	Was ist Generalisierung? . . . . .	8
2.8	Was ist Spezialisierung? . . . . .	8
2.9	Was ist Überladung? . . . . .	9
2.10	Was ist Kapselung? . . . . .	9
2.11	Wozu ist Kapselung gut? . . . . .	9
2.12	Was sind Modifizierer ( <code>public</code> , <code>private</code> , <code>protected</code> )? . . . . .	10
2.13	Was ist eine API? . . . . .	10
2.14	Was ist eine Relationale Datenbank? . . . . .	11
2.15	Welche Vorteile bietet OOP ? . . . . .	11
2.16	Welche Konventionen gibt es? . . . . .	12
2.17	Was ist eine abstrakte Klasse? - [Needs Rewrite Urgently. Optional.] . . . . .	12
<b>3</b>	<b>C++-Besonderheiten</b>	<b>14</b>
3.1	Kleine Fallstricke . . . . .	14

3.2	Speichermanagement . . . . .	14
3.2.1	Statische Variablen . . . . .	14
3.2.2	Dynamisch allozierte Variablen . . . . .	14
3.2.3	Dynamisch allozierte Skalare . . . . .	16
3.2.4	Dynamisch allozierte Arrays . . . . .	16
3.3	WAHR und FALSCH . . . . .	17
<b>4</b>	<b>C++</b>	<b>18</b>
4.1	Beispiel: Hund . . . . .	18
4.2	Klassen . . . . .	19
4.3	Was ist <code>this</code> ? . . . . .	19
4.4	Was ist ein Konstruktor? . . . . .	19
4.5	Was ist ein Destruktor? . . . . .	19
4.6	Wrapper-Klassen . . . . .	19
4.7	In Dateien aufspalten . . . . .	20
4.7.1	Inline - für kleine Programme . . . . .	20
4.7.2	Header- und Definitionsdateien - für "normale" Programme . . . . .	20
4.8	Übung . . . . .	22
4.8.1	Bringen Sie sich selbst was bei . . . . .	22
4.8.2	Gewünschte Ausgabe des Programms . . . . .	22
4.8.3	Hunde-Namen . . . . .	22
4.8.4	Bell-Counter . . . . .	22
4.8.5	Schrottplatz . . . . .	22
4.8.6	Hund 1940 . . . . .	22
4.8.7	Auseinanderrupfen . . . . .	22
4.8.8	Meute . . . . .	22
4.8.9	Neugierig? . . . . .	22
4.9	Das komplette Beispiel - mit Strings . . . . .	23
<b>5</b>	<b>Objekt-Interaktion unter C++</b>	<b>25</b>
5.1	Dereferenzierungs-Operator . . . . .	25
5.2	Statisches Massaker . . . . .	25
5.3	Interaktion dynamischer Objekte . . . . .	25
5.4	Massaker : Das Programm . . . . .	27
5.5	Übung : Bauen Sie das Programm um wie folgt: . . . . .	28
5.5.1	Geschwätzigkeit . . . . .	28
5.5.2	Tiere . . . . .	28
5.6	In Dateien aufspalten . . . . .	28
5.7	Dynamisches Massaker . . . . .	28
5.7.1	Populationsschwankung . . . . .	28
5.8	Programmlauf zu Übung Geschwätzigkeit . . . . .	28
<b>6</b>	<b>Java-Besonderheiten</b>	<b>30</b>

6.1	Falle für Programmierer, die in C++ nicht aufgepaßt haben . . . . .	30
<b>7</b>	<b>Java-Architektur</b>	<b>31</b>
7.1	Consolen-Programm . . . . .	31
7.2	Applet . . . . .	32
<b>8</b>	<b>Java-Windos-App</b>	<b>33</b>
<b>9</b>	<b>Java-GUI-Programme (under heavy construction)</b>	<b>34</b>
9.1	Layout und Verhalten der Widgets . . . . .	34
<b>10</b>	<b>Java-Events - Under Construction</b>	<b>36</b>
10.1	Verwendung lokaler und anonymer Klassen . . . . .	36
<b>11</b>	<b>Layout-Manager</b>	<b>42</b>
11.1	HLayout und VLayout . . . . .	43
11.2	FlowLayout . . . . .	43
<b>12</b>	<b>Praktische Anwendung unter QT</b>	<b>44</b>
12.1	Signals mit Parametern . . . . .	45
12.2	IDE für QT . . . . .	45
12.3	IDE für GNOME . . . . .	45
<b>13</b>	<b>Fazit OOP GUI Programmierung</b>	<b>45</b>
<b>14</b>	<b>PHP-Grundlagen</b>	<b>46</b>
14.1	PHP - PHP Hypertext Preprocessor . . . . .	46
14.1.1	Client-Server-Modell . . . . .	46
14.1.2	Präprozessor . . . . .	47
14.1.3	Einbetten . . . . .	47
14.1.4	Home-Verzeichnisse . . . . .	47
14.1.5	PHP-Variablen . . . . .	48
14.2	Ausgewählte Sprachelemente . . . . .	48
14.2.1	Ausgabe . . . . .	48
14.2.2	Variablen . . . . .	48
14.2.3	Zufallszahlen . . . . .	49
14.2.4	Verzweigung . . . . .	49
14.2.5	For-Do-Schleife . . . . .	49
14.2.6	Variablenübergabe . . . . .	49
14.3	Übung . . . . .	51
14.3.1	Gib "Hallo Welt" aus. . . . .	51
14.3.2	Gib die Sinusse von 0 bis 9 aus. . . . .	51
14.3.3	Zufallszahl . . . . .	51
14.3.4	Aufrufzähler . . . . .	51

14.3.5	Eingabeüberprüfung mit <code>isset</code> . . . . .	51
14.3.6	Eingabe verarbeiten . . . . .	51
14.3.7	Mail . . . . .	51
14.3.8	Gästebuch . . . . .	51
<b>15</b>	<b>Strukturiert Programmieren unter PHP</b>	<b>52</b>
15.1	Problem . . . . .	52
15.2	Lösung . . . . .	52
15.3	Sicherheit???	52
15.4	Beispiel . . . . .	53
<b>16</b>	<b>OOP mit PHP</b>	<b>55</b>
16.1	Übung . . . . .	55
16.1.1	Konstruktor . . . . .	55
16.1.2	Hunde . . . . .	55
16.1.3	Katz und Maus . . . . .	55

## Disclaimer

Wissen ist zum Teilen da. Ich teile mein Wissen mit Ihnen, lieber Kollege.

Ich bin aber nicht perfekt. Unter [worgtsone@hush.com](mailto:worgtsone@hush.com) nehme ich dankbar Ihre Verbesserungsvorschläge entgegen.

\*

**Legal Blurb:** Alle Informationen in diesem Dokument sind falsch, unvollständig, irreführend, irrelevant und / oder funktionieren einfach nicht.

Wenn Sie es trotzdem benutzen, und es geht dabei etwas kaputt, ist das Ihr Problem, nicht meins.

\*

**Bitte teilen Sie meine Web-Adresse nicht Ihren Schülern mit.**

# 1 Intro

## 1.1 OOP in anderen Lehr-Unterlagen

In anderen Lehr-Unterlagen sind die allgemeinen Begriffe der OOP und die speziellen Begriffe der jeweils benutzten Programmiersprache völlig gemischt.

Wenn Sie das gut finden, lesen Sie woanders weiter.

Wenn Sie es allerdings methodisch sauber trennen wollen - wenn Sie OOP zum einen (was trocken und unanschaulich ist) und Programmierung zum anderen (was mischgefährdet ist) unterrichten wollen - dann sind Sie hier richtig.

## 1.2 Under Construction

Dieses Dokument ist im Bau. Ihre Hilfe ist willkommen. S. Disclaimer.

## 1.3 Vorsicht mit `pdflatex`

Ich schreibe dies unter `pdflatex`. Ich habe `pdflatex` im Verdacht, daß es Tilden und Minuszeichen verschwinden läßt.

Überprüfen Sie, ob Sie die rechts angegebenen Zeichen links alle sehen:

```
i--;           // i minus minus semikolon  
Maus::~Maus   // Maus colon colon tilde Maus
```

## 2 Theorie und Terminologie der OOP

### 2.1 Was ist ein Objekt ?

Ein Objekt ist eine Instanz einer Klasse.

### 2.2 Was ist eine Klasse?

Eine Klasse ist ein Bauplan zur Konstruktion von Objekten.  
 Eine Klasse ist eine Idee oder Abstraktion von real existierenden Objekten.  
 Damit aus der Klasse) etwas Reales wird, damit man es anfassen kann, muß es instanziiert (= gebaut) werden.  
 Eine Klasse kann 0 bis viele Instanzen haben.

#### 2.2.1 Beispiele

Klasse (Idee)	Objekt (Reale Ausprägung)
Reihenhaus	Ginsterweg 1, Ginsterweg 2, ... Ginsterweg 24
VW Golf 1	OF-VW 1972, OF-VW 681, OF-XS 400
Affe	Affe Karl, Affe Knut, Äffin Lisa
Schüler	Hans, Franz, Fritz
Mercedes C 111	(wurde leider nie gebaut)
Verkauf (Geschäftsobjekt)	Verkauf eines C111 für 1 EUR an wortstone

### 2.3 Was ist in einer Klasse?

Eine Klasse kann 0..viele Variablen (auch: Daten) und 0..viele Funktionen (auch: Methoden, methods, functions) haben.  
 Variablen und Funktionen sind Mitglieder (auch: Bestandteile, Member) der Klasse.

### 2.4 Was ist Abstraktion?

Abstraktion ist das zweckgebundene Bilden von Klassen.  
 Abstraktion ist, wenn man vom Speziellen zum Allgemeinen kommt.

**Beispiel** Zweck sei das Modellieren einer Schule im Rechner.  
 Durch Betrachtung verschiedener realer Schüler stellen wir fest:

- Sie haben gemeinsame Attribute, zB Vorname, Nachname, ...)
- Sie haben gemeinsame Verhaltensweisen, zB stoeren(), weiterführendeFrageStellen()

Wir bilden daraufhin eine Klasse Schüler.

## 2.5 Was ist Modellierung?

**Modellierung ist das Bilden von Klassen und deren Beziehungen untereinander.**

**Modellierung ist, wenn man eine Vielzahl realer Gegenstände und Vorgänge abstrahiert, so daß man komplexe Gebilde (zB eine Schule) im Rechner nachbilden ("modellieren") kann.**

**Der Zweck der Abstraktion bestimmt dabei, welche Variablen und Methoden man in die Klassen aufnimmt.**

Wir erkennen, daß alle Schüler, alle Lehrer und alle Hausmeister gemeinsame Eigenschaften haben - zB daß sie Menschen sind.

Für einen Schulleiter, der Lehrer modelliert, ist es uninteressant, ob sich darunter Golfspieler befinden. (Es sei denn, er spielt selber Golf.) Für einen Sportartikelhändler, der Kunden modelliert, ist das dagegen existentiell.

## 2.6 Was ist Vererbung?

**Vererbung ist, wenn eine Klasse alle Variablen und Funktionen von einer anderen Klasse übernimmt und ggf. noch eigene hinzufügt. In der OOP können nur Klassen von Klassen erben.**

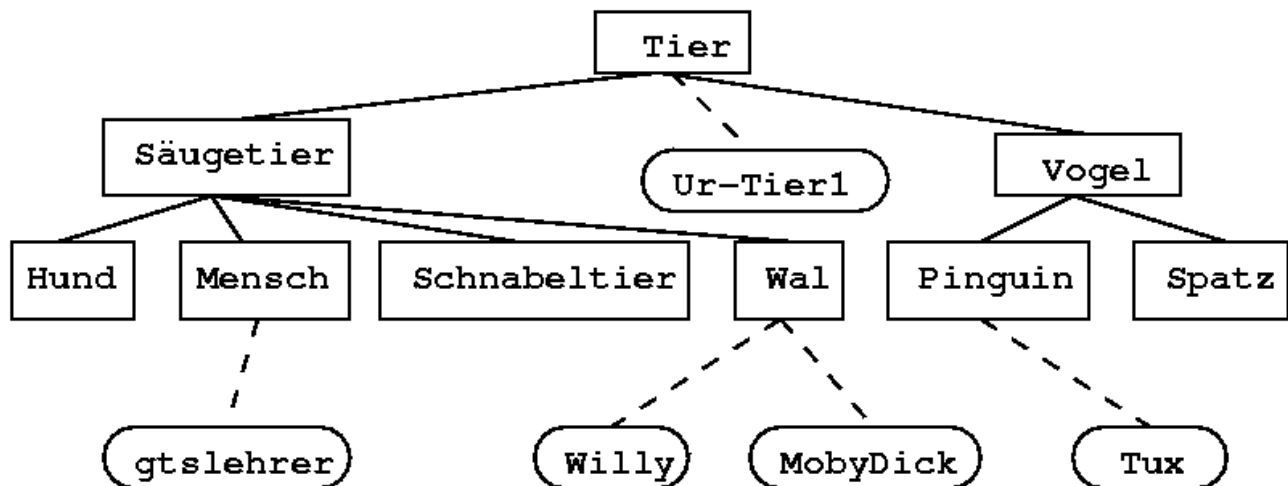
**Die erbende Klasse heißt Unterklasse (subclass). Die vererbende Klasse heißt Oberklasse (super class).**

**Vererbung kann man mit folgenden Worten ausdrücken: "(Unterklasse) ist ein Spezialfall von (Oberklasse)." oder (einfacher): "(Unterklasse) ist ein (Oberklasse)."**

Vererbung ist **nicht**, wenn aus der Zusammenarbeit zweier Hunde ein neuer Hund entsteht. Das Ergebnis ist bloß ein weiteres Objekt, ebenfalls der Klasse Hund.

**Beispiel** Tier: kann geboren werden, atmen, sterben.

Hund: kann alles, was Tier kann. Zudem: einen Namen haben, bellen und beißen.



Vererbung sollte nur eingesetzt werden, wenn sie die anstehenden Aufgaben erleichtert.

Vererbung mit mehr als drei Hierarchieebenen gilt als unübersichtlich, da man zum Verstehen einer Unterklasse stets in ihre Oberklassen schauen muß.

Achtung: Vererbung wird sehr gern als Baumdiagramm gemalt. Nehmen Sie für ihre Klassen (zB Affe) Kästen. Wenn Sie noch Objekte dazunehmen wollen (zB Äffin Lisa), nehmen Sie Kreise oder Kästen mit runden Ecken oder so.

## 2.7 Was ist Generalisierung?

**Bei der Generalisierung wird untersucht, ob mehrere Klassen eine gemeinsame Oberklasse haben.**

Zweck: wenn einige Variablen und/oder Funktionen sehr ähnlich oder gar gleich sind, modelliert man sie einmal in der Oberklasse (statt zB dreimal in den Unterklassen).

**Beispiel** Schulleiter, Hausmeister und Schüler sind Menschen.

## 2.8 Was ist Spezialisierung?

**Spezialisierung ist das Gegenteil von Generalisierung: wenn man vom Allgemeinen zum Speziesierten, von der Oberklasse zur Unterklasse kommt.**

**Beispiel** Ein Schulleiter ist ein Mensch - aber ein spezieller.

## 2.9 Was ist Überladung?

**Überladung ist, wenn eine geerbte Funktion geändert wird.**

**Beispiel** Ein Vogel **ist-ein** Tier. Er kann zudem durch die Luft fliegen.  
Ein Pinguin **ist-ein** Vogel. Er kann aber nicht fliegen. Statt

```
Vogel::fliege() {
    cout << "Ich fliege durch die Luft flap flap flap"<< endl;
}
```

muß es beim Pinguin heißen:

```
GespraechigerPinguin::fliege() {
    cout << "Ich kann nicht fliegen, aber prima schwimmen(),
    tauchen() und fischFressen()."<< endl;
}
```

## 2.10 Was ist Kapselung?

**Kapselung ist, wenn man auf manche Variablen oder Funktionen eines Objektes nur mit klasseneigenen `public` Methoden zugreifen kann.**

Zugreifen = lesen oder schreiben.

## 2.11 Wozu ist Kapselung gut?

### Kapselung

- modelliert das Verhalten eines Objekts.
- standardisiert die Zugriffe auf das Objekt.
- ermöglicht Plausibilitätsprüfungen.

Wenn Geburtsdaten änderbar sind, kann das zu Fehlern führen.

Wenn der Programmierer findet, daß keiner von außen seine Variable `i` lesen oder ändern soll, dann schreibt er keine Funktion dazu, und fertig.

Wenn der Programmierer findet, daß bei derzeit lebenden Hunden Geburtsdaten vor 1950 oder in der Zukunft Blödsinn sind, schreibt er die entsprechenden Kontroll-Routinen in die Methode `setGeburtsdatum()`.

**Bei einem guten Modell ist von außen nicht sichtbar, welche Daten für sein Verhalten verantwortlich sind.**

Es ist eine **Black Box.**

**Beispiele** Schulleiter wechseln selten die Haarfarbe - Schüler öfter. Obwohl beides Menschen sind.

Wenn es im Programm "Mahnwesen Tischlerei" eine funktionierende Klasse "Kunde" gibt, dann kann man sie im Programm "Mahnwesen Klempnerei" oder "Mahnwesen Versandhaus" wiederverwenden.

## 2.12 Was sind Modifizierer (public, private, protected)?

**Modifizierer bestimmen, auf welche Daten oder Funktionen man von außen zugreifen kann.**

- **public (öffentlich)**  
Auf diese Mitglieder kann man von außen zugreifen.
- **private (privat; nicht zugänglich)**  
Auf diese Mitglieder kann man NICHT von außen zugreifen.
- **protected (geschützt)**  
Auf diese Mitglieder kann die aktuelle Klasse und alle ihre Kinder (Erben) zugreifen.
- **Auf private Daten kann man nur mit public Methoden zugreifen. Oder gar nicht.**

## 2.13 Was ist eine API?

Zum Zugreifen über `public` Funktionen muß man natürlich wissen, wie die Klassen-Funktionen heißen und was für Parameter sie verlangen.

**Die Gesamtheit der Funktionen ist die Schnittstelle einer Klasse nach draußen.**

**Nur hier kann man Funktionen aufrufen (= Befehle erteilen = Reize einleiten), worauf die Klasse dann mit ihrer eingebauten Logik reagieren wird.**

**Die Gesamtheit der Funktionen heißt**

- **Schnittstelle oder**
- `interface` **oder**
- **API (Application Programming Interface).**

Eigentlich wäre es schön, wenn jede Klasse eine Methode `talkAboutYourself()` hätte. Diese Methode würde die Änderungsmöglichkeiten bekanntgeben.

## 2.14 Was ist eine Relationale Datenbank?

Eine Datenbank kann verstanden werden als eine Ansammlung von Klassen, die Variablen, aber keine Funktionen haben.

Jede Tabelle gibt die Struktur der einzelnen Zeilen (= Datensätze = Tupel) vor. Die Tupel sind Instanzen der Klasse.

## 2.15 Welche Vorteile bietet OOP ?

- **Man kann Klassen optimieren, ohne das Hauptprogramm ändern zu müssen.**
- **Man kann funktionierende Hauptprogramme in kurzer Zeit schreiben. Man muß bloß ein paar funktionierende Klassen nehmen und ihre Interfaces korrekt verbinden.**
- **Man kann Klassen wiederverwenden, sofern man ihr Interface kennt.**  
**Diese Wiederverwendung spart Geld, weil man die Klassen nicht erneut schreiben und nicht erneut debuggen muß.**
- **Man muß das zu schreibende Programm modularisieren.**

Die Wiederverwendung fehlerfreier Klassen ist das Hauptargument für OOP. Es wird behauptet, das spare VIEL Geld.

Dafür muß man seine Mitarbeiter

- von der Qualitäts- und Effektivitätssteigerung durch OOP überzeugen;
- ihnen strukturierte<sup>1</sup> Programmierung abgewöhnen;
- sie kontinuierlich anhalten, ihre Klassen gut dokumentiert und wiederverwendbar zu schreiben und;
- sie überzeugen, daß das nicht ihren Arbeitsplatz kosten wird.

Es wird behauptet, das koste viel Geld. Und Zeit.

## 2.16 Welche Konventionen gibt es?

1. Klassennamen schreibt man im Singular, ohne Leerzeichen und groß.
2. Hauptwortanfänge schreibt man zur Steigerung der Lesbarkeit auch groß.
3. Variablen- und Funktionen-Namen schreibt man klein.
4. Die Funktion zum Ändern einer Variablen heißt `setVariable()`, die zum Lesen `getVariable()`.

### Beispiele

```
int meineZahl;
int getMeineZahl();
void setMeineZahl();
class Pinguin {};
class GeschwaetzigerPinguin {}
BigInteger bi;
BorderLayout mybl = new BorderLayout();
```

## 2.17 Was ist eine abstrakte Klasse? - [Needs Rewrite Urgently. Optional.]

**Beispiel** Ein Löwe ist ein Tier. Eine Schlange ist ein Tier.

Ein Greif ist eine Mischung aus Löwe und Schlange. Er soll deshalb sowohl von Löwe als auch von Schlange erben.

Der Compiler sieht die Funktionen von `Tier` in beiden Oberklassen (`Löwe` und `Schlange`) und weiß daraufhin nicht, welche er nehmen soll.

**Lösung:** `Tier` als abstrakte Klasse definieren.

**Eine abstrakte Klasse kann mehrpfädig vererben.  
Aber: Eine abstrakte Klasse kann nicht mehr instanziiert werden.**

Äh.

Abstrakte Klassen können Prototypen von Funktionen enthalten (also ohne definierenden Rumpf) und können deshalb nicht instanziiert werden.

Prototypen sind Funktionen ohne Rümpfe, zB

<sup>1</sup>ist lt. PHP-Tutorial imperative Programmierung ohne GOTO.

```
belle () {}
```

Erst wenn in der Vererbungshierarchie alle abstrakten Funktionen (???) einer abstrakten Klasse implementiert werden, kann man die Klasse instanzieren.

## 3 C++-Besonderheiten

### 3.1 Kleine Fallstricke

- = ist der Zuweisungsoperator
- == <= >= > < != sind Vergleichsoperatoren
- && ist logisches Und
- & (kaufm. Und; Ampersand) ist bitweises Und
- || ist logisches Oder
- | (senkrechter Strich; bar) ist bitweises Oder
- ^ (Dach; Caret) ist bitweises XOR
- Short-Circuit (Kurz-Schluß):

```
a=0;
if (a++ == 0) || (++b==5) { ...} // b wird nicht inkrementiert
```

- ```
int foo=0;
int bar() {
    double foo;
    ...
} // das äußere foo ist nicht sichtbar
```

### 3.2 Speichermanagement

#### 3.2.1 Statische Variablen

Statische Variablen werden zu Programmbeginn alloziert (d.h. sie belegen Speicher) und existieren bis Programmende. Dabei ist es egal, ob man sie noch braucht oder nicht.

```
int main () {
    int aa; // ein int namens aa, belegt 4 Byte
    int b=9; // Variablen d"urfen auf einmal initialisiert und
            // deklariert werden
    long double f; // belegt 10 Byte
    int a[45]; // belegt 45*4 = 180 Byte
            // die Feldelemente hei"sen a[0] bis a[44]
}
```

#### 3.2.2 Dynamisch allozierte Variablen

Dynamisch allozierte Variablen werden zur Laufzeit auf dem Heap angelegt.

Das OS weiß, wieviel freier Hauptspeicher noch da ist. Am einen Ende davon wird der Heap ("Haufen") angelegt und wächst nach oben. Am anderen Ende wird der Stack ("Stapel", "Keller") angelegt und wächst nach unten.

**Stack** Auf dem Stack werden die Rücksprungadressen und CPU-Register beim Aufruf von Unterprogrammen abgelegt (PUSH). Programme mit zahlreichen (z.B. rekursiven) Funktionen und Prozeduren verbrauchen viel Stack.

**Heap** Zum Allokieren dynamischer Variablen bittet das Programm das OS um freien Speicher einer gewissen Menge. Es erhält ihn in Gestalt eines Zeigers auf den Anfang des Speicherbereiches. Das OS markiert den Speicher anschließend als belegt.

**Freigeben von Speicher** Speicher wird durch Aufruf von `delete (p_pointer)` freigegeben und kann anschließend vom OS weiterverwendet werden.

Wenn das Programm nicht ordnungsgemäß beendet wird, wird der allozierte Speicher nicht freigegeben und steht bis zum nächsten Reboot nicht mehr zur Verfügung.

**Memory Leak - Speicherüberlauf** Heap und Stack wachsen aufeinander zu, der (noch) freie Speicher liegt dazwischen. Unter ungünstigen Umständen geht der freie Speicher auf 0 zurück, der Rechner "friert" dann ein.

Dies kann z.B. passieren, wenn

- der Programmierer vergißt, allozierten Speicher wieder freizugeben oder
- das Programm aufgrund eines Laufzeitfehlers abbricht, bevor `delete` ausgeführt wird.

### Pointer - Zeiger

```
int a;           // a ist eine variable
int *b;         // *b ist ein zeiger auf eine variable
a=10;           // a wird 10 zugewiesen, d.h. in die speicherzelle,
                // die a enthält, wird 10 hineingesteckt.
b = &a;         // b wird die adresse der speicherzelle von a
                // zugewiesen; & heißt: sag mir die adresse
*b = 5;         // in die speicherzelle, auf die b zeigt, wird 5
                // hineingesteckt - also ist a jetzt 5
```

Die vom OS zugewiesene Speicheradresse wird in einem Pointer gespeichert, der durch \* gekennzeichnet wird.

**&** ist der Referenzierungsoperator und gibt eine Speicheradresse zurück.  
**\*** ist der De-Referenzierungsoperator und bearbeitet eine Speicheradresse.

In C war es üblich, die Pointer durch den Vorsatz `p_` zu kennzeichnen. Ich folge im folgenden diesem Brauch.

### 3.2.3 Dynamisch allozierte Skalare

Im folgenden Beispiel wird

- eine `int`-Variable dynamisch alloziert und belegt,
- Adresse und Inhalt der Speicherzelle ausgegeben;
- der Speicher wieder freigegeben.

```
int main () {
    int *p_int = new int;
    *p_int = 19;
    cout << "p_int zeigt auf Speicherzelle " << p_int << endl;
    cout << "Darin steht der Wert 19 :      " << *p_int << endl;
    delete (p_int);
}
```

Programmlauf:

```
p_int zeigt auf Speicherzelle 0x80499d0
Darin steht der Wert 19 :      19
```

### 3.2.4 Dynamisch allozierte Arrays

Im folgenden Beispiel wird

- ein Feld aus `int` dynamisch alloziert und belegt,
- Adressen und Inhalte der Speicherzelle ausgegeben;
- der Speicher wieder freigegeben.

```
int main () {
    int *p_int2 = new int[5];
    int *p_int2_work = p_int2;          // create pointer for working
    for (i = 0; i < 5; ++i) {
        cout << i << "p_int2_work zeigt auf " << p_int2_work;
        *p_int2_work = i * i;
        cout << " mit Wert : " << *p_int2_work << endl;
        p_int2_work++;                  // increment p_int2
    }
    delete[] (p_int2);                  // must use [] to delete array
}
```

Programmlauf:

```
0 p_int2_work zeigt auf 0x8049b88 mit Wert : 0
1 p_int2_work zeigt auf 0x8049b8c mit Wert : 1
2 p_int2_work zeigt auf 0x8049b90 mit Wert : 4
3 p_int2_work zeigt auf 0x8049b94 mit Wert : 9
4 p_int2_work zeigt auf 0x8049b98 mit Wert : 16
```

**Inkrement** `p_int2_work` ist ein Pointer auf eine 4 Byte lange `int`-Variable. Die nächste `int`-Variable liegt also 4 Bytes höher im Speicher.

Deshalb wird `p_int2_work` beim Inkrementieren nicht um 1, sondern um 4 erhöht.

### 3.3 WAHR und FALSCH

Das 80er-Jahre-Paradigma lautete:

**Falls ein Programm wegen eines Fehlers abgebrochen wird, gibt es einen Fehlercode  $> 0$  zurück.**

**Falls ein Programm korrekt beendet wird, gibt es Fehlercode = 0 zurück. Error 0 = no error.**

Das C++-Paradigma lautet:

**Jede Anweisung gibt einen Ergebniscode zurück.**

**Ungleich 0 bedeutet WAHR. 0 bedeutet FALSCH.**

Dieser Vorzeichenwechsel ist in erster Linie lästig für die Leute, die sich umgewöhnen mußten. Er ist auch ein bißchen gefährlich:

```
if (x = 5) {  
    cout << "Immer erfuehlt!" << endl;  
}
```

Was so aussieht wie ein Vergleich, ist eine Zuweisung. Der Ergebniscode einer Zuweisung ist das, was zugewiesen wurde, in diesem Fall also 5. 5 ist WAHR.

Man verwende die folgende Syntax, dann gibt es einen Compilerfehler, falls man `=` statt `==` schreibt:

```
if (5 = x) {  
    cout << "Kompiliert nicht!" << endl;  
}
```

## 4 C++

### 4.1 Beispiel: Hund

```

class Hund {                                     // mit vielen Fehlern

private:
    int geburtsJahr;
    *char name;

public:
    int getGeburtsJahr () {
        return (geburtsjahr);
    }

    void belle() {
        cout << this <<" : Wauwauwau!" << endl;
    }

    Hund () {
        geburtsJahr = 2000;
    }

    Hund (char *newName, int newGeburtsJahr) {
        strcpy (name, newName);
        if (newGeburtsjahr < 1950) {
            geburtsjahr = 2000;
        } else {
            geburtsJahr = newGeburtsJahr;
        }
    }

    ~Hund () {
        cout << this << " Ouh - ich sterbe..." << endl;
    }
};   // nich midde ssemikolons sspaasam sein

// ***** main() *****

int main () {
    Hund bello(1998);
    Hund defaul();
    Hund *p_hund = new Hund();
    Hund *p_hund2 = new Hund("Wuffi", 2002);
    bello.belle();
    defaul.belle();
    p_hund->belle();
    p_hund2->belle();
}

```

## 4.2 Klassen

Klassen beginnen mit dem reservierten Wort `class`.

Die Default-Einstellung für Members und Methods ist `private`.

Eine Klassen-Definition muß durch `;` abgeschlossen werden. Immer.

Beachten Sie die Kapselung im folgenden Beispiel: die Member sind `private` und können nur durch `public` Methoden gelesen oder gesetzt werden.

## 4.3 Was ist `this`?

`this` ist ein Zeiger auf das aktuelle Objekt und enthält somit eine Speicheradresse. Er braucht aber kein `*` (Sternchen).

## 4.4 Was ist ein Konstruktor?

Ein Konstruktor heißt genau wie die Klasse.

Ein Konstruktor dient zum Initialisieren eines Objekts.

Ein Konstruktor hat keinen Rückgabetyt (auch nicht `void`).

Der Konstruktor wird automatisch beim Instanzieren eines Objektes aufgerufen.

Ein selbstgeschriebener Konstruktor kann die Anfangsparameter prüfen und Daten initialisieren.

Wenn man keinen Konstruktor schreibt, wird der Default-Konstruktor genommen. Dieser kann nichts prüfen oder initialisieren.

Man kann Konstruktoren überladen. Man schreibt dazu mehrere, die dann verschiedene Parameterlisten haben müssen.

## 4.5 Was ist ein Destruktor?

Ein Destruktor heißt genau wie die Klasse, bloß mit einer Tilde davor.

Ein Destruktor dient zu Aufräumarbeiten beim De-Instanzieren eines Objekts.

Ein Destruktor hat keinen Rückgabetyt (auch nicht `void`) und nimmt keine Parameter.

Ein Destruktor wird automatisch beim De-Allozieren eines Objekts aufgerufen.

**Beim Aufruf von `new` wird automatisch der passende Konstruktor aufgerufen.**

**Beim Aufruf von `delete` wird automatisch der Destruktor aufgerufen.**

## 4.6 Wrapper-Klassen

In C++ gilt es als elegant, wenn in `main()` nur noch eine Zeile steht:

```
MyClass object1 = new MyClass();
```

In anderen OO Programmiersprachen gibt es gar kein `main`.

Für beide Fälle müssen wir den Code, der unsere eigentlichen Objekte instanziiert, woanders hineinpacken.

Unsere Hunde bellen natürlich nicht im luftleeren Raum, sondern auf der Straße oder im Wald oder im Vorgarten. Streng gedacht können wir den Wald auch noch modellieren (d.h. als Klasse formulieren) - auch wenn seine einzige Aufgabe darin besteht, ein paar Hunde zu enthalten, die vor sich hin kläffen.

Damit er wenigstens dies tut, muß die Instanzierung und Belegung der Hunde in die einzige Methode, die automatisch beim Instanzieren der Klasse Wald aufgerufen wird: ihren Konstruktor.

**Wrapper-Klassen enthalten im Konstruktor das, was beim strukturierten Programmieren nach `main()` gehört hätte.**

## 4.7 In Dateien aufspalten

Weil die Klasse eine BlackBox sein soll, ist es jedem Programmierer EGAL, WIE eine Methode definiert wird - Hauptsache er weiß, welche Methoden es überhaupt gibt.

Es hat sich eingebürgert, das Verzeichnis der Methoden einer Klasse `myclass` in eine extra-Datei zu schreiben, die auf `.h` endet und Header-Datei heißt: `myclass.h`.

Die Methoden werden anschließend in der Datei `myclass.cpp` definiert.

Das verbessert die Übersicht, die Möglichkeit rekursiver Aufrufe aus verschiedenen Klassen und die Compilierzeit.

**Deklarationen stehen in `myclass.h`.**  
**Definitionen stehen in `myclass.cpp`.**  
**Das Hauptprogramm muß nur die `.h`-Dateien inkludieren. Diese inkludieren ihre `.cpp`-Dateien gefälligst selbst, man muß sich nicht mehr darum kümmern, wenn sie einmal korrekt geschrieben sind.**

### 4.7.1 Inline - für kleine Programme

```
class Katze {

    public:
    void schnurre () {
        cout << "Purrrrrrrrr..." << endl;
    }
};
```

### 4.7.2 Header- und Definitionsdateien - für "normale" Programme

```
// this is Katze.h
#include <iostream>
class Katze {
```

```
public:
    void schnurre ();
};
#include <"Katze.cpp">

// this is Katze.cpp
void Katze::schnurre () {
    cout << "Purrrrrrrrr..." << endl;
}
```

## 4.8 Übung

### 4.8.1 Bringen Sie sich selbst was bei

Unterstreichen und numerieren Sie die Sachen, die Sie halb oder gar nicht verstehen.

Raten Sie, wo ich die Datenkapselung versteckt habe.

Bringen Sie das Programm in eine lauffähige Version.

### 4.8.2 Gewünschte Ausgabe des Programms

```
0xbffffba4 Ich lebe seit 1998
0xbffffba0 Ich lebe seit 2000
0x8049d40 Ich lebe seit 2000
0x8049d50 Ich lebe seit 2002
0xbffffba4 : Wauwauwau!
0x8049d50 Ouh - ich sterbe...
0xbffffba0 : Wauwauwau!
0x8049d40 Ouh - ich sterbe...
0xbffffba0 Ouh - ich sterbe...
0xbffffba4 Ouh - ich sterbe...
```

### 4.8.3 Hunde-Namen

Bringen Sie den Hunden auch noch Namen bei. Der Default-Hund heißt `default`, die anderen beiden Wuffi und Bello. Benutzen Sie dabei die `c++`-Klasse `String`.

### 4.8.4 Bell-Counter

Ändern Sie die Klasse so, daß jeder Hund beim Sterben sagt, wie oft er gebellt hat.

### 4.8.5 Schrottplatz

Modellieren und benutzen Sie die Wrapper-Klasse `Schrottplatz`.

### 4.8.6 Hund 1940

Versuchen Sie einen Hund Baujahr 1940.

Bauen Sie die Klasse `Hund` so um, daß zu alte Hunde nicht mehr bellen, sondern bloß knurren können.

### 4.8.7 Auseinanderrufen

Rufen Sie das Programm in mehrere Dateien auseinander (`Hund` `Katze` `Vorgarten`) und erschaffen Sie einen geräuschvollen Vorgarten.

Interaktion zwischen Objekten kommt im nächsten Kapitel, also können die Hunde keine Katzen jagen und die Katzen keine Hunde kratzen.

### 4.8.8 Meute

Versuchen Sie nach den üblichen `C++`-Regeln ein Array of `Hund` und lassen Sie sie mit Hilfe von Zufallszahlen benennen, bellen und fressen.

Erschaffen Sie so viele Hunde, daß Ihr Rechner langsamer wird.

Versuchen Sie, einen der Hunde aus der Meute zum Sterben zu überreden.

### 4.8.9 Neugierig?

Wir empfehlen zum Nachschlagen:

1. `LösungAllerProbleme: volkards c++ tutorial;`
2. `www.cplusplus.com.`

## 4.9 Das komplette Beispiel - mit Strings

```

#include <iostream.h>           // leider c-strings statt c++-String

class Hund
{

private:
    int geburtsJahr;
    char name[30];

public:
// 2 Konstruktoren für Hund
    Hund ();
    Hund (char *newname, int newGeburtsJahr);
    ~Hund ();                               // TILDE Hund
    int getGeburtsJahr ();
    void belle ();
};

Hund::Hund () {
    geburtsJahr = 2000;
    strcpy (name, "Default");
    cout << this << " " << name << " lebt seit " << geburtsJahr << endl;
}

Hund::Hund (char *newname, int newGeburtsJahr){
    strcpy (name, newname);
    if (newGeburtsJahr < 1950){
        geburtsJahr = 2000;
    } else {
        geburtsJahr = newGeburtsJahr;
    }
    cout << this << " " << name << " lebt seit " << geburtsJahr << endl;
}

Hund::~Hund () {
    cout <<this <<" " <<name <<" ist tot, geburtsJahr= " << geburtsJahr <<
    endl;
}

int Hund::getGeburtsJahr () {
    return (geburtsJahr);
}

void Hund::belle () {
    cout << this << " : Wauwauwau!" << endl;
}

int main () {

```

```

Hund bello ("Bello", 1998);
Hund noname;
Hund *p_hund = new Hund;
Hund *p_hund2 = new Hund ("Abraxas", 2002);
bello.belle ();
delete (p_hund2);
noname.belle ();           // statische Hunde sterben nicht
p_hund->belle;
p_hund2->belle ();        // dynamische sterben jedoch
delete (p_hund);
}

```

```

***** Sat Jan 17 19:53:05 EST 2004 ***** c++ start *****
0xbffffb74 Bello lebt seit 1998
0xbffffb50 Default lebt seit 2000
0x8049e40 Default lebt seit 2000
0x8049e68 Abraxas lebt seit 2002
0xbffffb74 : Wauwauwau!
0x8049e68 Abraxas ist tot, geburtsJahr= 2002
0xbffffb50 : Wauwauwau!
0x8049e68 : Wauwauwau!           // kann nicht sein
0x8049e40 Default ist tot, geburtsJahr= 2000
0xbffffb50 Default ist tot, geburtsJahr= 2000
0xbffffb74 Bello ist tot, geburtsJahr= 1998
***** Sat Jan 17 19:53:05 EST 2004 ***** c++ end *****

```

## 5 Objekt-Interaktion unter C++

**Beispiel** Katzen fressen Mäuse. Die Mäuse sterben dabei, und die Katzen nehmen Nahrung auf.

In C++ liegt die Haupt-Arbeit bei der Maus.

- Die Maus muß Nährwert abgeben. Wieviel Nährwert eine Maus gibt, hängt von vielen Faktoren ab, die aber alle Bestandteil und somit gekapselte Daten der gefressenen Maus sind.

Deshalb weiß auch nur die Maus, wieviel Nährwert sie hat:

```
Maus::gibNaehrwert() {
    return (gewicht * fett * (isFemale + 1) / alter) ;
    Maus::~~Maus();
}
```

- Vielleicht noch ein letztes Quieken?

```
Maus::~~Maus {
    cout << this << " Quiek - ich bin tot. << endl;
}
```

- Die Katze nimmt nur den Nährwert der fraglichen Maus auf:

```
Katze::frissMaus (Maus &m) { // s. DEREFERENZIERUNG
    sattheit = sattheit + m.gibNaehrwert();
}
```

### 5.1 Dereferenzierungs-Operator

Offensichtlich muß man in diesem Beispiel der Katze die zu fressende Maus übergeben. Ein Objekt übergibt man als Zeiger. Die Katze jedoch soll die echte Maus fressen (nicht den Zeiger).

### 5.2 Statisches Massaker

Auf der folgenden Seite ist ein Katz-und-Maus-Massaker mit statischen Katzen und Mäusen.

**Beachten Sie:** Die Mäuse `micky` und `goofy` können unendlich oft Nährwert geben, da sie bis Programmende nicht aufhören zu existieren.

### 5.3 Interaktion dynamischer Objekte

Falls Sie dynamische Katzen haben wollen, müssen Sie die Klasse Katze ändern:

```
Katze::frissMaus (Maus *m) { // * statt &
    sattheit = sattheit + m.gibNaehrwert; // statische Maus
}
```

Falls Sie auch dynamische Mäuse haben wollen:

```
Katze::frissMaus (Maus *m) {                // * statt &
    sattheit = sattheit + m->gibNaehrwert;  // dynamische Maus
}
```

Dynamische Objekte arbeiten sehr zuverlässig und haben sich auf breiter Front durchgesetzt.

## 5.4 Massaker : Das Programm

```
#include <iostream.h>

class Maus {

public:
    int gibNaehrwert () {
        cout << this << " gab Nährwert." << endl;
        return 5;
    }
};

class Katze {

private:
    int geburtsjahr;
    int sattheit;

public:
    Katze (int gebjahr) {
        geburtsjahr = gebjahr;
        sattheit = 60;
    };

    Katze () {
        geburtsjahr = 1999;
        sattheit = 60;
    }

    Katze::frissMaus (Maus & m) {                // statische Maus
        sattheit = sattheit + m.gibNaehrwert ();
        cout << this << "lecker Mäuschen... sattheit = " << sattheit <<endl;
    }
};

int main () {
    Maus goofy;
    Katze kitti (1982);
    Katze karlo;
    Maus micky;

    kitti.frissMaus (micky);
    kitti.frissMaus (goofy);
}
```

## 5.5 Übung : Bauen Sie das Programm um wie folgt:

### 5.5.1 Geschwätzigkeit

Katzen und Mäuse sollen beim Geborenwerden und Sterben sagen, was passiert ist und in welcher Speicherzelle sie sitzen.

Untersuchen Sie anhand der Bildschirm-Ausgabe, in welcher Reihenfolge die Mäuse und Katze geboren werden und sterben.

### 5.5.2 Tiere

Mäuse und Katzen sollen von einer geschwätzigen Klasse `Tier` erben.

## 5.6 In Dateien aufspalten

Zerlegen Sie das Programm so, wie es sich gehört:

- Jede Klasse kommt in ihre eigene Datei.
- Am Anfang von `myprogram.cpp` wird inkludiert.

## 5.7 Dynamisches Massaker

Erzeugen Sie dynamische Katzen und Mäuse.

### 5.7.1 Populationsschwankung

Programmieren Sie - nach Wahl - Arrays of Tier oder dynamisch erzeugte Tiere (mit einem `array of pointer`) unter folgenden Bedingungen:

1. Die Mäuse sollen sich exponentiell vermehren;
2. Die Katzen sollen sich exponentiell vermehren, sobald mehr als 9 Mäuse pro Katze da sind;
3. Einzelne Katzen sollen verhungern, wenn weniger als 5 Mäuse pro Katze da sind.

## 5.8 Programmlauf zu Übung Geschwätzigkeit

```
***** Wed Jan 14 18:51:36 EST 2004 ***** c++ start *****
0xbffffb87 Maus lebt!
0xbffffb7c Katze lebt!
0xbffffb74 Defaultkatze, gebjahr = 1999.
0xbffffb73 Maus lebt!
0xbffffb73 gab Nährwert.
0xbffffb7c lecker Mäuschen... sattheit = 65
0xbffffb87 gab Nährwert.
0xbffffb7c lecker Mäuschen... sattheit = 70
0xbffffb73 Maus tot.
0xbffffb74 Katze tot, gebjahr = 1999 sattheit = 60.
```

0xbffffb7c Katze tot, gebjahr = 1982 sattheit = 70.

0xbffffb87 Maus tot.

\*\*\*\*\* Wed Jan 14 18:51:36 EST 2004 \*\*\*\*\* c++ end \*\*\*\*\*

## 6 Java-Besonderheiten

Java hat zahlreiche Vorteile:

1. es gibt exzellentes Einstiegsmaterial (100 % shell commands) unter `www.javabuch.de`,
2. multithreading,
3. eingebaute Sicherheit,
4. eingebaute Krypto-Algorithmen,
5. plattformunabhängig - java classes kann in einer (plattformabhängigen) virtuellen Maschine (VM) auf jeder Hardware ausgeführt werden, für die es eine VM gibt,
6. kann alles incl. Grafik-Oberfläche, Serverbetrieb, Datenbank-Zugriffe, ...
7. fix mit JIT (Just-In-Time-Compiler),
8. Experten zufolge wesentlich einfacher für Anfänger als C++,
9. Range checking und automatische Speicherüberprüfung ist eingebaut und stabil.

Nachteile:

- keine bekannt

Indifferent:

1. Versionen 1.0 bis 1.4. Ab Version 1.2 heißt es Java 2.
2. Man braucht eine passende VM mit JIT. Dafür läuft dann alles.
3. Ältere Versionen haben bekannte Bugs, die Workarounds stehen im `www`, wie bei anderen Sprachen auch.
4. Moderne Versionen haben unbekannte Bugs mit unbekanntem Workarounds, wie bei anderen Sprachen auch.

### 6.1 Falle für Programmierer, die in C++ nicht aufgepaßt haben

```
Maus micky = new Maus();  
micky->setGeburtsJahr = 1940;  
Maus goofy = micky;  
goofy->setGeburtsJahr = 1960;
```

`micky` und `goofy` sind immer gleich alt, da sowohl `micky` als auch `goofy` Zeiger auf denselben Speicherplatz sind.

## 7 Java-Architektur

Es gibt

1. Konsolenprogramme;
2. GUI-Programme;
3. Applets;

Applets sind bunte anspruchslose Programme, die in Browsern laufen und mit denen der User interagieren kann. Besonders nett: java Asteroids.

Applets dürfen keine Dateien schreiben und keine Systemvariablen lesen. Und nicht drucken.

4. Servlets. Servlets sind kleine, auf dem Server ausgeführte Programme, die Eichhörchen überprüfen, Datenbanken abfragen, pdfs erzeugen und Kaffee kochen können und dies alles per html zum User schicken.

Ein Java-Programm enthält eine oder mehrere Klassen.

Es gibt kein Hauptprogramm. Statt dessen wird zur Laufzeit eine Instanz der geladenen Klasse instanziiert und deren Methode `public static void main()` ausgeführt.

### 7.1 Consolen-Programm

Sobald eine Klasse geladen ist, wird ein Objekt gemäß dem Bauplan der Klasse instanziiert und seine Methode `public static void main()` ausgeführt. Die folgenden Programme machen dasselbe:

```
program hello;                                (* in Pascal *)
begin
  writeln ('Hello World!!!');
end.
```

```
public class HelloWorld {                    /* in Java */
  public static void main (String[] args) {
    System.out.println ("Hello World!!!");
  }
}
```

1. Speichern Sie den obigen Quelltext in der Datei `HelloWorld.java`. Die Datei muß denselben Namen haben wie die Klasse, und sie muß auf `.java` enden.
2. Kompilieren: `javac HelloWorld.java`. Dabei entsteht `HelloWorld.class`.
3. Ausführen: `java HelloWorld`.

## 7.2 Applet

Ein Applet ist Bytecode, der vom Browser geladen wird. Browser laden normalerweise nur html-Seiten, also müssen wir in die html-Seite folgendes einbinden:

```
<html><head><title>Mein erstes Applet</title></head>
<body>
  <applet code="MeinErstesApplet.class" width=200 height=200>
  </applet>
</body> </html>
```

MeinErstesApplet erbt von der Oberklasse Applet alle Daten und Methoden eines Applets. Außerdem beinhaltet es die Schnittstellen (Übergabepunkte), so daß es seinen eigenen Grafik-Bereich erkennen und dort einen String hineinschreiben kann.

```
// MeinErstesApplet.java.

import java.applet.*;
import java.awt.*;

public class MeinErstesApplet extends Applet {
  public void paint (Graphics grafikbereich) {
    grafikbereich.drawString ("Hello Internett!", 50, 150);
  }
}
```

Kompilieren, html-Seite und a1.class ins selbe Verzeichnis stellen, html-Seite angucken.<sup>2</sup>

---

<sup>2</sup>Ich weiß nicht, wieso dieses Applet ohne start() und run() auskommt - ich wüßte es aber gern.

## 8 Java-Windows-App

GUI-Systeme wie Xwindows oder Windows stellen Programmen eine Umgebung zur Verfügung, die Befehle entgegennimmt. So etwas heißt in der Rechner-technik *Server*.

Java kann mit diesem Server reden, zB: "Mach mir ein Fenster der Höhe 200 und Breite 150 Pixel."

Dazu muß es den AbstractWindowToolkit (AWT) importieren und eine Klasse erzeugen, die von Frame erbt.

Dieses Objekt erzeugt dann ein neues Objekt vom Typ Grafikbeispiel, welches ein paar Parameter setzt und anschließend automatisch seine Methode `paint` aufruft:

```
/* GrafikBeispiel.java */
import java.awt.*;

public class GrafikBeispiel extends Frame {
    int xsize=150;
    int ysize=200;

    public static void main (String[]args) {
        GrafikBeispiel wnd = new GrafikBeispiel (); // speicher allozieren
    }

    public GrafikBeispiel () {
        super ("GrafikBeispiel");
        setBackground (Color.white);
        setForeground (Color.blue);
        setSize (xsize, ysize);
        setVisible (true);
    }

    public void paint (Graphics g) {
        for (int i = 0; i < xsize; ++i) {
            for (int j = 0; j < ysize; ++j) {
                g.drawLine (i, j, i, j);
            }
        }
    }
}
```

Zum Starten: Kompilieren, nach Xwindows wechseln, `java Grafikbeispiel` aufrufen. Das entstandene Fenster weiß nicht, wie es auf Events (Tastendrucke, Mausclicks) reagieren soll und muß daher hart abgebrochen werden.

Schauen Sie, wie Java die Speicherzeiger für Sie verwaltet! Unter C++ wäre `wnd` ein unhandlicher Zeiger. Unter Java ist es ein Objekt. Wenn Sie möchten, daß `wnd` tanzt, ist das eine Zeile weit weg. Wenn `wnd` ein Sound-Interface braucht, auch: <sup>3</sup>

```
wnd.tanze(); // mit java ist es wieder ein ., kein ->
SoundMachine mysm = new SoundMachine();
OutputDevice od = mysm.autoGetOutputDevice();
od.setVolume (od.MAXIMUM);
FileJuggler myfj = new FileJuggler();
mysm.playSound(myfj.getSoundFromUrl("http://www.go.org/sm.wav"));
```

---

<sup>3</sup>Hups da hab ich mich vergaloppiert, `SoundMachine` und `FileJuggler` gibts noch nicht. Freiwillige?

## 9 Java-GUI-Programme (under heavy construction)

Sie halten Winword für einen Textprozessor. Ich halte Winword für eine Spielwiese für Widgets.<sup>4</sup>

**Ein Widget ist ein Bildschirm-Objekt.**

**Beispiele:** Schaltflächen, DropDownMenus, Bitmaps, Eingabefelder, Checkbuttons, ...

### 9.1 Layout und Verhalten der Widgets

Gegeben sei folgendes Programm: ein Fenster, links ein Button mit Beschriftung BOOOO!, rechts einer mit Beschriftung Exit.

Das BOOOO! sei ein farbenfrohes `boooo.gif`, schwarze Kinderschrift auf gelbem Grund. Zunächst haben wir hier einen Widget Tree. Widgets, die auf anderen Widgets sitzen, habe ich eingerückt. Parameter habe ich als Name-Wert-Paare dahintergestellt.

```
MainWindow (Titel=GrafikBeispiel)
  LinksRechtsLayout // ein linker Teil, Trennung senkrecht,
                    // ein rechter Teil.

  linker Teil
    Button1 (Lieblingsgröße=40x40)
            boooo.gif
  rechter Teil
    Button2 (Beschriftung=Exit)
```

Sie haben schon bemerkt, daß alle Buttons sich ähnlich verhalten:

1. Sie befinden sich irgendwo auf dem Bildschirm.
2. Sie haben eine Beschriftung.
3. Wenn man draufklickt, passiert irgend etwas.

Alle Buttons verhalten sich ähnlich. Also gibt es wahrscheinlich eine Klasse `Button`. Damit er weiß, wo er liegt und mit was er beschriftet ist, hat er möglicherweise die Daten `Position` und `Label` und die Methode `oopsIHaveBeenClicked()`.

---

<sup>4</sup>Eigentlich halte ich nichts von bunten Benutzer-Oberflächen (engl.: GUI, Graphical User Interface).

1. Schade um die Rechner-Resourcen, die dafür verbraucht werden.
2. Die User sitzen stundenlang ohne Bewegung davor, verkrampft von der linken Maustaste bis zur Wirbelsäule, und blinzeln vornübergebeugt auf zu kleine Schrift.
3. Man weiß nie, was ein Windos-Programm gerade im Hintergrund anstellt.
4. Aber die User wollen es so : sonst müßten sie was lernen.

Ich würde deshalb normalerweise keine GUI-Programme schreiben.

Methodisch-didaktisch finde ich aber: man kann damit schön und bunt Objektorientierung demonstrieren.

Das, was der Button auslösen soll (zB Fenster schließen, Programm beenden, boooo.wav abspielen), kann man in den Button nicht einbauen. Nicht, weil er zu komplex würde, sondern weil so etwas in eine Klasse `Button` einfach nicht hineingehört.

Fenster schließen wäre eine angemessene Methode für `MainWindow`. Ein Fenster sollte am besten selbst wissen, wie es sich zeigt oder verbirgt oder zum Symbol verkleinert.

Programm beenden wäre eine angemessene Methode für `System`.

Sounds abspielen wäre eine angemessene Methode für `SoundMachine`. Diese müßte

1. eine Instanz von `SoundMachine` - errr - instanzieren (ach nennen wir sie `mysm`);
2. von `FileIO` die Datei `boooo.wav` holen;
3. eine Instanz von `SoundAusgabeGerät` - errr - instanzieren (ach nennen wir sie `mysag`);
4. `boooo.wav` auf `mysag` abspielen.

**Merken Sie, wie Sie langsam anfangen, in Klassen zu denken? Eine Klasse ist eine Art Maschine. Wenn Sie den Konstruktionsplan haben, können Sie sie in Java bauen (erben), was drankonstruieren (überladen), die Maschine herstellen (instanzieren), einstellen (mit Parametern) und schließlich damit arbeiten (`myMachine.go()`).**

## 10 Java-Events - Under Construction

Damit aus dem Programm etwas Interaktionsfähiges wird, muß es auf Tasten und Mausklicks reagieren.

Was die Maus angeht, verwendet Java den kleinsten gemeinsamen Nenner. Xwindows kennt Drei-Tasten-Mäuse, Windos kennt nur Zwei-Tasten-Mäuse, und Apple kennt nur eine Taste. Ergo kennt auch Java nur eine.

Die Tastendrucke und Mausklicks werden vom darunterliegenden System aufgefangen und an die Java-Maschine weitergeleitet. Java-Programme haben insgesamt 4 Möglichkeiten, davon Kenntnis zu nehmen:

1. Implementieren eines Listener-Interfaces;
2. Verwendung lokaler und anonymer Klassen;
3. Verwendung lokaler Klassen;
4. Überlagern der Event-Handler in den Komponenten.

Ich beschränke mich hier auf die Verwendung lokaler und anonymer Klassen, da dabei schnell etwas Funktionierendes entsteht und dennoch die Klassen der GUI-Widgets und die Klick-Logik halbwegs getrennt bleiben.

### 10.1 Verwendung lokaler und anonymer Klassen

Der Grundgedanke ist, daß etwas passiert, wenn eine Maus- oder Tasten-Aktion ausgeführt wird.

Dazu braucht man eine "Maschine", die beständig "horcht", ob eine solche Aktion aufgetreten ist. Die entsprechenden Klassen in Java heißen KeyAdapter bzw. MouseAdapter. Sie sind fix und fertig definiert (damit sie wissen, wie sie auf Aktionen "horchen"), aber sie wissen nicht, welche Aktivität durch welche Aktion ausgelöst werden soll, d.h. ihre Methodenrumpfe sind leer.

Wir werden entsprechend in unseren kleinen Programmen eine Subklasse der Adapter definieren und darin die Methoden überlagern.

```
001 /* Listing2803.java */
002
003 import java.awt.*;
004 import java.awt.event.*;
005
006 public class Listing2803
007 extends Frame
008 {
009     public static void main(String[] args)
010     {
011         Listing2803 wnd = new Listing2803();
012     }
013
014     public Listing2803()
015     {
```

```
016     super("Nachrichtentransfer");
017     setBackground(Color.lightGray);
018     setSize(300,200);
019     setLocation(200,100);
020     setVisible(true);
021     addKeyListener(new MyKeyListener());
022 }
023
024 public void paint(Graphics g)
025 {
026     g.setFont(new Font("Serif",Font.PLAIN,18));
027     g.drawString("Zum Beenden bitte ESC drücken...",10,50);
028 }
029
030 class MyKeyListener
031 extends KeyAdapter
032 {
033     public void keyPressed(KeyEvent event)
034     {
035         if (event.getKeyCode() == KeyEvent.VK_ESCAPE) {
036             setVisible(false);
037             dispose();
038             System.exit(0);
039         }
040     }
041 }
042 }
```

Das obige Programm definiert die Klasse Listing2803, die von `Frame` erbt, also ein Fenster mit Rahmen und Titelleiste auf den Bildschirm zeichnen kann.

Die Funktion `main()` ruft den Konstruktor auf. Dieser ruft ein paar `set`-Methoden der Oberklasse `Frame` auf und instanziert anschließend noch ein Interface zum Lesen der Tastatur namens `MyKeyListener`.

Die Klasse `MyKeyListener` ist innerhalb der Klasse `Listing 2803` definiert. Sie erbt von `KeyAdapter`.

Sie überlagert deren Methode `keyPressed(KeyEvent event)`. Falls also ein `event` (ein Tastendruck) auftritt, wird dieser geholt und kann mittels seiner Methoden bearbeitet werden.

ZB kann `MyKeyListener` prüfen, ob der `KeyCode` des Events dem in der Klasse `KeyEvent` definierten `VirtualKey Escape` entspricht, und anschließend die eingebauten Methoden der Klasse (zB `setVisible()` und `dispose()`) sowie der darunterliegenden Java-Maschine `System` aufrufen.

Das folgende Programm zeigt, wie man alle möglichen Events abfängt.

```

/*!!
!!!!!!!!!!!! Events nach dem Modell ab JDK 1.2 aufwärts

Bedingung:
    unter windos keine

    unter grmlinux jikes.sh benutzen:
===== snip =====
#!/bin/sh

if [ z$1 = z ] ; then
    echo "Usage: $0 ClassFileName[.class]"
    exit 1
fi

jf=$(basename $1 .java)
echo "indenting $jf.java..."
indent $jf.java

if [ z$DISPLAY = z ] ; then
    joe $jf.java
else
    kwrite $jf.java
fi

clear
echo Compiling...
jikes -classpath /usr/share/kaffe/Klasses.jar $jf.java && \
    echo Executing... && java $jf
===== snap =====

!!*/

/*!!
    wer ereignisse verarbeiten will, muß events importieren
!!*/
import java.awt.event.*;
/*!!
    wer frames etc. haben will, muß awt importieren
!!*/
import java.awt.*;

/*!!
    also ich stell mir das so vor: meine klasse ist eine
    maschine, und diese ganzen verflixten interfaces sind
    dranschnacksbare module. erst mit modul dran kann ich,
    sagen wir, eine maus anschließen.
    überraschung! für die maus gibt es zwei module, für
    keyboard eins, für knöpfe eins namens ActionListener.

```





Diese Mechanismen und die relativ komplizierte Syntax machen das Programmieren von Hand zur Qual. Schlimmer noch sind die merkwürdigen und verdrehten Fehlermeldungen beim Compilieren und Laufenlassen.

Die Verwendung einer integrierten Entwicklungsumgebung (IDE) scheint notwendig. Diese sollte können:

- Anordnen der Widgets mittels Maus und Layout-Managern;
- Verknüpfen der Widgets durch Intelligente Dialoge;  
(Beispiel: Ich wähle meinen Knopf `FOO`, wähle aus der aufploppenden Liste `onMouseKlick` und wähle aus dem nächsten aufploppenden Fenster die Aktion `frobnify()` der Klasse `Erfunden`, wobei ich ein Integer und einen Pointer als Parameter übergebe.
- einen MAKE-Button. Anklicken aktiviert das Compilieren und Ausführen des Programms.

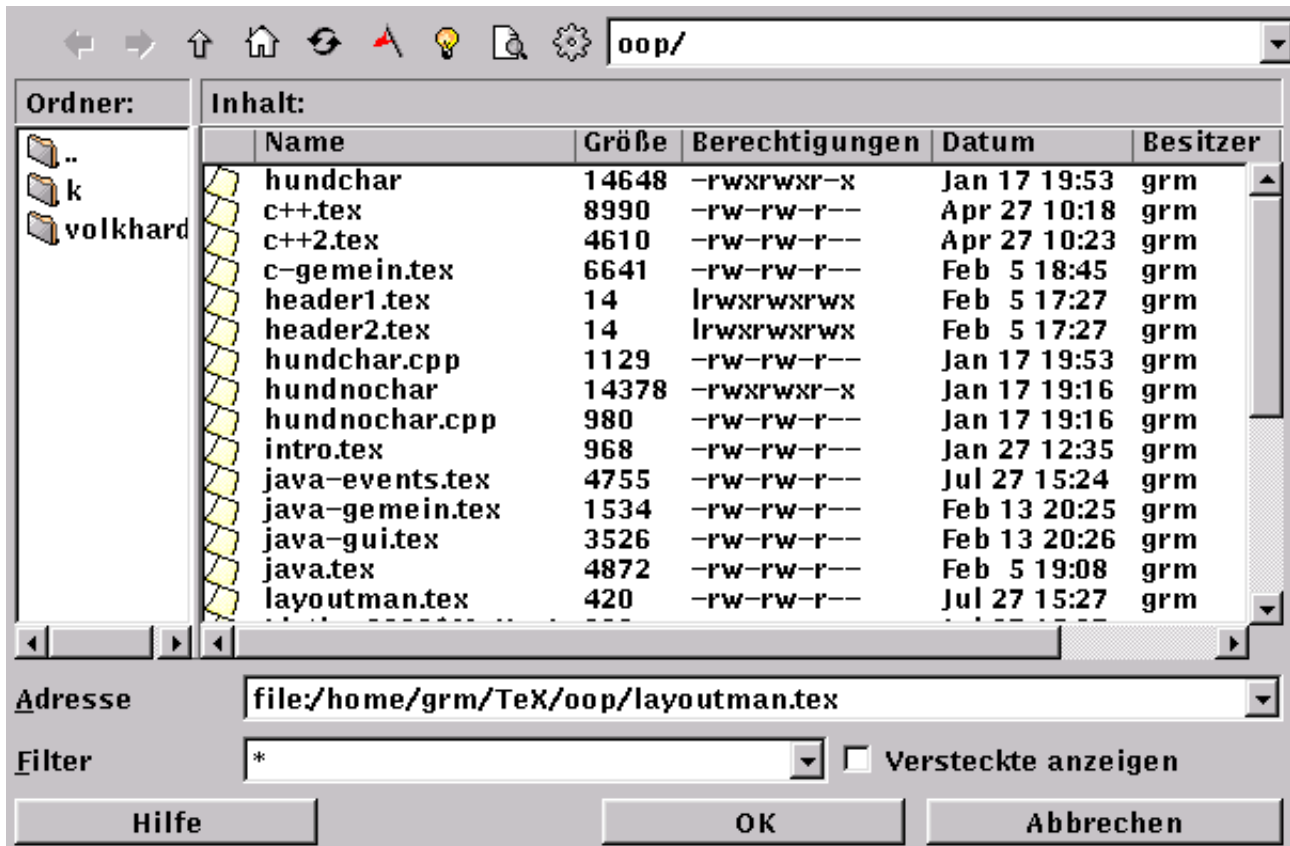
Ich suche seit einem halben Jahr danach. Ich höre, ECLIPSE sei ein entsprechendes Frontend, das u.a. Java könne, aber ich komm damit nicht klar, trotz der ganzen schönen Tutorials, die ich gelesen habe. Sobald ich etwas Passendes gefunden habe, melde ich mich hier.

## 11 Layout-Manager

Java protzt mit 6 eingebauten und beliebig vielen selbstbastelbaren Layout-Managern. Von denen versteh ich die meisten gar nicht und brauche eigentlich nur zwei.

Die Idee bei Layout-Managern ist, daß man ihnen auf intelligente Weise sagt, wie sie ihre Arbeit zu machen haben, und sich anschließend nicht mehr darum kümmern braucht.

Betrachten wir hierzu ein Layout für eine Bildschirmmaske:



Zunächst sehe ich nur eine verwirrende Vielzahl einzelner Widgets.

**Ein Widget ist ein Baustein für eine GUI-Anwendung.**

**Beispiele:** PushButtons ("OK" oder "Abbrechen"), ToggleButtons ("Sorted/Unsorted"), je mit Text oder Bild oder beidem geschriftet, Combo-Boxen (zum Ausklappen), RadioButtons ("Pizza: groß XOR klein XOR mittel"), CheckButtons ("Pizza: extraKäse, Anchovis, ohneKnoblauch"), Menübalken, Datei-Auswahl-Dialog, Farb-Auswahl-Dialog, etc.

**Auch ein Fenster (mit Titelbalken usw.) ist ein Widget.**

Dann sehe ich aber, daß die Widgets in Gruppen entweder übereinander oder nebeneinander angeordnet sind.

Beispiel: Die Schaltflächen und die ComboBox in der obersten "Zeile" sind nebeneinander, die Knöpfe in der untersten "Zeile" auch. Die so definierten "Zeilen" liegen übereinander.

## 11.1 HLayout und VLayout

Hier sind der vertikale und der horizontale Layout-Manager `VLayout` und `HLayout` am Werk. Man kann ihn auf etwas drauflegen, und er fügt es am Ende (also entweder darunter oder rechts daneben) an.

**Definition: Das Draufgelegte ist das Kind. Der Ablageort ist der Vater. Alle Widgets (außer das primäre Fenster) haben einen Vater.**

Das oben dargestellte Dialog-Fenster kann man also beschreiben wie folgt: "Zuerst kommt das Fenster. Darauf lege ich ein `VLayout`. Darauf lege ich ein `HLayout`. Darauf lege ich einen Knopf, und auf den Knopf das Symbol `PfeilNachLinks(inaktiv)`. Und noch einen Knopf..."  
Übersichtlicher wäre die Darstellung als Widget-Baum (widget tree):

```

window wnd
  vlayout vlayout1
    hlayout hlayout1
      button pfeillinks
        bitmap pfeillinks.png
      button pfeilrechts
        bitmap pfeilrechts.png
      button ...
    hlayout hlayout2
      dirtree verzeichnisbaum
      filelist filelist1
    ...
  hlayout hlayout5
    button hilfe
      text hilfe
    button ok
      text ok
    button abbrechen
      text abbrechen

```

Die Layout-Manager können ihre Minimal-Größe bestimmen, indem sie ihre Kinder nach deren Minimalgröße fragen und dann berechnen.

Fenster stellen sich so stets auf die kleinste Größe ein, in der man alles lesen und sehen kann.

## 11.2 FlowLayout

Java stellt das Flow-Layout zur Verfügung. Das Fenster wird dabei Zeile für Zeile und anschließend Spalte für Spalte mit Widgets gefüllt. Je nach Breite des Fensters erscheinen die Widgets in einer bis vielen Zeilen.

`FlowLayout` ist mir unsympathisch.

## 12 Praktische Anwendung unter QT

QT von `www.trolltech.no` unterstützt das OOP von GUIs mit einem ausgereiften Widget-Toolkit. Trolltech stellt ihn für Open-Source-Anwendungen kostenlos zur Verfügung, für kommerzielle Anwendungen kostet er Lizenz-Gebühren. Trolltech fördert so Open-Source-Software.

QT bietet alle Widgets, um komplette Benutzeroberflächen damit zu bauen. In der Tat ist das KDE Desktop Environment unter QT gebaut.

QT gibt es für Linux und Windos. Anwendungsentwickler, deren Kunden auf Linux umsteigen, müssen bloß den Quelltext auf einen Linux-Rechner übertragen, kompilieren, und schon läuft. Leider werden die Anwender anschließend die spektakulären Systemabstürze unter Windos vermissen.

```
// Entnommen aus dem QT-Tutorial, Kapitel 2
// kompiliert auf meinem Linux-System mit:
//  c++ -I/usr/lib/qt-2.1.0/include -o t2 \
//      -L/usr/lib/qt-2.1.0/lib/      -lqt \
//      t2.cpp

#include <qapplication.h>
#include <qpushbutton.h>
#include <qfont.h>

int main( int argc, char **argv )
{
    QApplication a( argc, argv );

    QPushButton quit( "Quit", 0 );
    quit.resize( 75, 30 );
    quit.setFont( QFont( "Times", 18, QFont::Bold ) );

    QObject::connect( &quit, SIGNAL(clicked()), &a, SLOT(quit()) );

    a.setMainWidget( &quit );
    quit.show();
    return a.exec();
}
```

Am Anfang von `main` wird eine `QApplication a` definiert.

Am Ende wird das `MainWidget` gesetzt und gezeigt sowie `a.exec` aufgerufen.

Das `MainWidget` ist in diesem Fall ein `QPushButton` namens `quit`. Er hat die Beschriftung "Quit" und keinen Vater - das braucht er als `MainWidget` aber auch nicht.

`Q` Widgets haben Signale und Slots. Signale emittieren Strom, wenn eine Aktion stattfindet. Slots reagieren auf Strom. Signale können beliebig viele Slots und Signale treiben.

Was wir noch machen müssen, ist ein Kabel ziehen: Wenn `quit` `clicked` wird, wird `a` geschlossen.

## 12.1 Signals mit Parametern

Leider lassen sich nur Signals und Slots mit gleichen Parametern connecten. `clicked()` ist parameterlos.

Um also zB drei Knöpfe zu generieren, die die Schriftgröße wahlweise auf 10, 11 oder 12 Punkt setzen, muß man

- die Klasse `Mein10PtButton` von `QPushButton` erben lassen ("ableiten");
- ihr beibringen: "Wenn du `clicked()` erhältst, dann emittiere das Signal `setFontSize(10)`."
- das Ganze mit dem `MetaObjectCompiler moc` unter Zuhilfenahme einer kruden Syntax - `errr - meta-compilieren(????)`;
- und erst dann kann man es durch den `Gnu C++ Compiler` jagen.

## 12.2 IDE für QT

Die IDE für QT heißt `kdevelop` und ist ab Linux 8.0 auf jeder Distribution dabei. Und ich komme damit nicht klar.

## 12.3 IDE für GNOME

Das GNOME Desktop Environment hat ein ähnliches Programm namens `Glade`. Leider produziert `Glade` nur C-Code und hat keine Signals und Slots - die Kabel-Syntax ist extrem abschreckend.

`Glade-- (glade-minus-minus)` produziert als Snap-In angeblich C++-Code, aber es kompiliert auf meinem System nicht.

# 13 Fazit OOP GUI Programmierung

Ich komme mit den drei großen Entwicklungsumgebungen unter Linux nicht klar. Allerdings verliert man sofort, nachdem man diese langsamen Monster mal angefaßt hat, den Überblick über den Quellcode.

Das ist zwar auch so gedacht - schließlich bedeutet OOP ua das Zusammenschnacksen vorhandener Module, egal was drin ist - aber eignet sich nicht besonders für Klausuren oder Unterricht.

## 14 PHP-Grundlagen

Wir empfehlen zum Nachschlagen:

- Lösung Aller Probleme: Manuals - PHP-Manual
- [php3.de](http://php3.de)
- [dclp-faq.de](http://dclp-faq.de)

### 14.1 PHP - PHP Hypertext Preprocessor

PHP ist eine in html-Seiten eingebettete Skriptsprache. Sie wird unter RedHat6.2 als Modul in den Apache eingebunden.

Sie unterstützt die üblichen Programmiersprachenelemente wie Funktionen, Skalare, Arrays, Hashes, Listen, Verzweigungen, Schleifen etc. und stellt darüber hinaus eine unglaubliche Anzahl von Funktionen zur Verfügung, mit denen man mit wenigen Zeilen Code folgendes kann:

- Berechnungen durchführen;
- Dateien öffnen, lesen, schreiben, anhängen, löschen;
- Datenbanken updaten und auslesen;
- GIFs erzeugen;
- Authentifizierung;
- Rechtschreibprüfung;
- (De)Kompression;
- Cookie-Erzeugung und -verwaltung;
- IMAP, LDAP, PDF, ODBC, Perl, SNMP, virtuelle Mailaccounts;
- etc.

Diese Liste ist nicht vollständig.

#### 14.1.1 Client-Server-Modell

Die Client-Anwendung (Browser) fragt bei der Server-Anwendung (Server, httpd) eine URL nach. Der Server liefert sie (oder eine Fehlermeldung).

Client-Programm und Server-Programm dürfen auf demselben Rechner laufen, müssen aber nicht.

**Besonderheit bei http:** Nachdem der Server geantwortet hat, wird die Verbindung geschlossen.

### 14.1.2 Präprozessor

Wenn der Server statische Seiten liefert, holt er sie von der Festplatte und schickt sie direkt zum Client.

Wenn der Server dynamische Seiten liefert, holt er sie von der Festplatte, schickt sie zunächst durch einen Präprozessor und dann erst zum Client.

Der Präprozessor

- sieht in der Datei nach, ob er Code ausführen soll;
- nimmt den Code heraus und
- packt ggf. Ergebnisse hinein.

PHP-Code wird durch die Tags `<? --Code-- ?>` oder `<?php --Code-- ?>` markiert. Es dürfen 0..viele Codeblöcke in einer html-Datei stehen, sie werden als zusammenhängendes Programm ausgeführt.

Die Lösung aller Probleme parst nur Dateien, die auf `.php3` enden.

### 14.1.3 Einbetten

```
<html> <body> <center>

<?php
    echo "Hallo Welt!";
?>

</center> </body> </html>
```

Beliebig viele PHP-Blöcke sind erlaubt, HTML-Tags sind optional:

```
<?php
    echo "<html> <body> <center>";
?>
<?php
    echo "Hallo Welt!";
    echo "</center> </body> </html>";
?>
```

PHP-Anweisungen müssen mit einem Semikolon abgeschlossen werden. Die gezeigte Einrückung ist nicht vorgeschrieben, aber gewährleistet Lesbarkeit.

### 14.1.4 Home-Verzeichnisse

Die Benutzer müssen im Home-Verzeichnis ein Unterverzeichnis `public_html` anlegen. Dort müssen sie eine Datei `index.html` anlegen. Andere Dateien dürfen sie anlegen und verlinken. Nur Dateien, die auf `.php3` enden, werden durch den Präprozessor geführt.

Die Seiten von `john.doe` liegen unter der Url `http://servername/~john.doe.` Außerdem müssen die User Mitglied der Gruppe des `httpd` sein.

### 14.1.5 PHP-Variablen

Variablen beginnen **immer** mit einem \$ und werden klein geschrieben.

Datentypen gibt es nicht. Zuweisungsüberprüfungen gibt es nicht.

Variablen werden nicht initialisiert, außer man benutzt sie. Sobald man eine Variable benutzt, gibt es sie auch.

Einer Variablen weist man durch = einen Wert zu. Das kann auch das Ergebnis einer Dateioperation (ein `filehandle`) oder einer Datenbank-Operation (zB eine Verbindungs-Nummer oder ein `select`-Resultat) sein.

```
$k = 1; $l=1;
```

macht folgendes:

- Falls k nicht existiert, wird es erschaffen.
- Falls k existiert, wird sein Datentyp auf integer gesetzt.
- k wird auf 1 gesetzt.
- Der Befehl gibt das Resultat 1 zurück (was weggeworfen wird).

## 14.2 Ausgewählte Sprachelemente

Variablenamen werden zu Variablenwerten expandiert.

### 14.2.1 Ausgabe

`print` gibt einen String aus.  
`echo` gibt eine Liste von Strings aus.  
**Mehrere Strings können durch den String-Concatenation-Operator (.)  
zusammenghängt werden.**

Beide ersetzen in den auszugebenden Strings die aufgefundenen Variablenamen durch die Werte der Variablen.

Beide ignorieren Funktionsaufrufe. Wenn das Ergebnis einer Funktion ausgegeben werden soll, muß man das zweistufig tun.

### 14.2.2 Variablen

**Zahlen und Rechnen** Sofern PHP die Strings als Zahlen interpretieren kann, kann man auch damit rechnen, sonst gibt es eine Fehlermeldung.

PHP unterstützt ( \* / - + ).

**Beispiel** Im folgenden Beispiel werden 5 Strings zusammenghängt, der fünfte String ist dabei das Ergebnis einer Berechnung.

```
$r=1.234;  
$s=2.345;  
echo $r."*".$s."=".(($r*$s));
```

### 14.2.3 Zufallszahlen

```

// dies ist ein Kommentar
srand(time());           // startet Zufallszahlengenerator neu
$myrand = rand(0,9);
echo $myrand;           // Zufallszahl zwischen 0 und 9 inclusive

```

### 14.2.4 Verzweigung

PHP unterstützt

```
== >= <= != > < && ||
```

Die Bedingung steht in Klammern, die Anweisungsblöcke in geschweiften Klammern. elseif- und else- Blöcke sind optional.

```

if ($a=="ein_string" || $b==3.141) {
    echo "Erfüllt!";
} elseif ($z==$z) then {
    echo "Ist immer erfüllt.";
} else {
    echo "Unvorhergesehener Fehler!";
}

```

### 14.2.5 For-Do-Schleife

```

// simuliert durch eine while-do-Schleife
$i=1;
while ($i<10)
{
    $temp=sin($i);
    echo "sin($i) = $temp <br>\n";           // \n ist hübscher
    $i++;
}

// oder in echt
for ($i=0 ; $i<10 ; $i++)           // Startanweisung, Weiter-Anweisung,
{                                     // Inkrementanweisung
    $temp=sin($i);
    echo "sin($i) = $temp <br>";
}

```

### 14.2.6 Variablenübergabe

Erinnern Sie sich an das Eingabefeld für Metacrawler? Die Parameter wurden in Gestalt von Name-Wert-Paaren in der URL transportiert.

HTTP unterstützt 2 Übertragungsmethoden: GET und POST. Bei GET kommen die Parameter auf der URL (wie bei metacrawler), bei POST stehen sie ohne ein abschließendes CR oder LF an STDIO.

Variablen zu extrahieren, war in früheren Zeiten sehr mühsam. Aber jetzt gibt es PHP.

Jedes Eingabefeld hat einen Namen. Kaum ist PHP gestartet, sucht es nach neuen Eingabeparametern.

Mit `isset()` kann man überprüfen, ob ein Parameter gesetzt wurde oder nicht.

Falls ja, existiert eine Variable namens `$Name`. Und diese hat automatisch den richtigen WERT!

Das kann zB eine Zahl sein oder ein String oder ein längerer String, z.B. ein komplettes Pascal-Programm aus einem `<TEXTAREA>`.

```
<?php
// this is eingabe.php3
    if (isset $temporaer)                // Datei eingabe.php3
        {
            echo "Ihre Eingabe war: <hr> <pre>";
            echo $temporaer;
            echo "</pre> <hr>";
        }
    else
        {
            echo "<h1> Eingabemaske</h1> <hr>";
            echo "<form method=post action=eingabe.php3>";
            echo "<textarea name=temporaer cols=60 rows=10>";
            echo "program p;\n\n";
            echo "begin\n";
            echo "  writeln ('sin(4) = ', sin(4));\n";
            echo "end.";
            echo "</textarea> <hr>";
            echo "<input type=submit value=Jetzt_losschicken!> </form>";
        }
?>
```

## 14.3 Übung

### 14.3.1 Gib "Hallo Welt" aus.

### 14.3.2 Gib die Sinusse von 0 bis 9 aus.

### 14.3.3 Zufallszahl

Setze eine Variable auf einen Zufallswert zwischen -3 und 3. Prüfe anschließend, ob das Ergebnis größer 0, gleich 0 oder kleiner 0 ist.

### 14.3.4 Aufrufzähler

Sobald diese Seite aufgerufen wird, wird der Zähler in count.txt um 1 erhöht, und es wird "Diese Seite wurde X mal besucht." ausgegeben.

### 14.3.5 Eingabeüberprüfung mit isset

Erzeugen Sie ein Eingabeformular, wo der User einen Text eingeben kann. Falls er einen eingibt, soll er zwischen zwei horizontalen Linien wieder ausgegeben werden.

### 14.3.6 Eingabe verarbeiten

Erzeugen Sie ein Eingabeformular mit einem Radio (Optionen: I will be back), einem Check-Balken (ene, mene, muh) und einem Runterfallmenü (Optionen: PHP ist toll, PHP ist spitze, PHP ist das Beste), und zeigen Sie, was der User gewählt hatte.

### 14.3.7 Mail

Erzeugen Sie ein Formular, wo der User seine eigene Emailadresse, den Empfänger, ein Betreff und einen Text eintragen kann.

Falls Absender gesetzt ist, soll bei "GoSendIt!" die Email verschickt werden, sonst soll "Gib mir eine Rückadresse!!!" ausgegeben werden.

Falls das Senden klappt, soll "Mail erfolgreich abgeschickt!" ausgegeben werden, sonst "Mail failed for delivery."

### 14.3.8 Gästebuch

Basteln Sie sich gaestebuch.php3. Wer vorbeikommt, darf Kommentare eintragen, muß aber seine email-adresse angeben. Kommentare mit dem Wort "sex" oder "porn" sollen zurückgewiesen werden.

## 15 Strukturiert Programmieren unter PHP

Strukturiert programmieren ist schön und interaktiv: Man kann den User gelegentlich fragen, was er machen will, etc.

### 15.1 Problem

```
var a,b: integer;

readln (a);
readln (b);
writeln ('a * b = ', a*b);
```

geht nicht ohne Weiteres in PHP. Naturgemäß sind PHP-Skripte SKRIPTE, d.h.

- sie erwarten ihre Parameter bereits auf der Kommandozeile;
- sie laufen exakt einmal;
- sie wollen keine interaktiven Abfragen.

### 15.2 Lösung

Immerhin kann man in PHP feststellen, von welchem Rechner eine Anfrage gekommen ist, und in eine Datei den Zwischenstatus schreiben. Der Zwischenstatus enthält selbst-definierte Breakpoints (bp), zB von 0..n, die bei den READLN-Anweisungen liegen. Man kann also so etwas machen:

```
welcher rechner ist es?
hat er schon eine status-datei?
  wenn nein: bp=0
  sonst:
    lies die status-datei
    parse die kommandozeile
    springe am bp ins programm.
speichere status in datei
exit
```

### 15.3 Sicherheit???

Die Status-Datei bekommt denselben Namen wie die IP des fremden Rechners. Das ist natürlich völlig unsicher. Es funktioniert auch nicht, wenn mehrere User gleichzeitig über einen Proxy auf das Skript zugreifen. Zudem ist es eine dumme Idee, diese Datei im `httpd`-Verzeichnis abzulegen - böse User können ihre Eingaben mit bösen PHP-Tags anreichern (zB Dateien löschen oder ändern), und ausführbar ist sie bereits.

Aber es funktioniert!

## 15.4 Beispiel

Das folgende Skript macht folgendes:

```
repeat
  readln (a);
  if (a<0) writeln ('a<0!!');
until (a>=0);
readln (b);
writeln ('a * b = ', a*b);
```

Die Breakpoints 1 und 2 sind dabei nach den readln-Zeilen, bp0 ist am Programm-Anfang. Die Variablen kommen entweder aus der Status-Datei oder von der Kommando-Zeile. Die von der Kommando-Zeile sind die aktuelleren.

Deshalb muß man

1. ALLE gesetzten Kommandozeilen-Variablen retten (there should be something nicer...),
2. die Status-Datei parsen (include \$session),
3. die geretteten Variablen überschreiben (... than this...).
4. das richtige Programm-Stückchen ausführen;
5. die neue Status-Datei mit allen Variablen schreiben (... a much better way!!!).

```
<?php

// Hilfsfunktion
function out ($string) {
  echo $string."<br>\n";
}

// Hilfsfunktion
function readln ($string) {
  echo "<form method=get>\n";
  echo "Gib mir einen Wert fuer $string :\n";
  echo "<input name=$string> <input type=submit> </form>";
}

////////// main //////////

// variables: a,b,bp;

$session=$REMOTE_ADDR;

if (isset($a)) $ta=$a;          // there should be something nicer...
if (isset($b)) $tb=$b;

if (file_exists($session)) {
  include ("$session");
```

```

} else {
    $bp=0;
}
if (isset($ta)) $a=$ta;
if (isset($tb)) $b=$tb;          // ...than this...

if ($bp==0) {
    out ("bp0: multipliziert a und b:");
    readln ("a");
    $bpn = 1;                    // another ugly variable
}

if ($bp==1) {
    out ("bp1:");
    if ($a < 0 ) {
        out ("a < 0 !!");
        readln ("a");
        $bpn = 1;                // kinda goto - i hate it, but what 2 do?
    } else {
        out ("a >= 0, good.");
        readln ("b");
        $bpn = 2;
    }
}

if ($bp==2) {
    out ("bp2:");
    out ("now we got $a and $b, that is ".$a * $b);
    echo "<a href=produkt.php3>enter 2 continue</a><br>";

// programmende
    unlink ($session);          // programmende -> löscht Statusdatei
    exit;
}

$bp = $bpn;

// save ($session)
    $fh = fopen ($session, "w");
    fwrite ($fh, "<?php\n");          // ...a much better way!!!

    if ($a != "") fwrite ($fh, "\$a = \"".$a ."\";\n");
    if ($b != "") fwrite ($fh, "\$b = \"".$b ."\";\n");
    if ($bp != "") fwrite ($fh, "\$bp= \"".$bp."\";\n");
    fwrite ($fh, '?>');
    fclose ($fh);

    exit;
?>

```

## 16 OOP mit PHP

Dateien werden inkludiert mit `include ("dateiname");`.

In PHP gibt es keine `private` und `public` Variablen - alles ist `public`. Private Variablen deutet man durch einen vorgestellten `Underscore` an. Er bedeutet: "Bitte ändere mich nicht direkt, sondern benutz die passende Methode!"

```
<?php

class Meerschwein {
    var $_hunger = 10;

    function getHunger() {
        return $_hunger ;
    }

    function wasBistDu ()    {
        $message = "Ich bin ein Meerschwein mit hunger =".
            $this->getHunger();
        return $message;
    }
}

$Meerschwein1 = new Meerschwein;
print $Meerschwein1->wasBistDu ();

?>
```

### 16.1 Übung

#### 16.1.1 Konstruktor

- Erzeugen Sie für das obige Beispiel einen Konstruktor().
- Erzeugen Sie einen Konstruktor(int Geburtsjahr, string name, string farbe).
- Erzeugen Sie einen Destruktor.

#### 16.1.2 Hunde

Bauen Sie eine Klasse geschwätziger Hunde mit Wrapper-Klasse `Wald`, und spielen Sie damit.

#### 16.1.3 Katz und Maus

Finden Sie heraus, wie man ein Massaker veranstaltet, und veranstalten Sie eines.

Finden Sie heraus, wie man eine Populationsschwankung modelliert, und lassen Sie sie laufen.