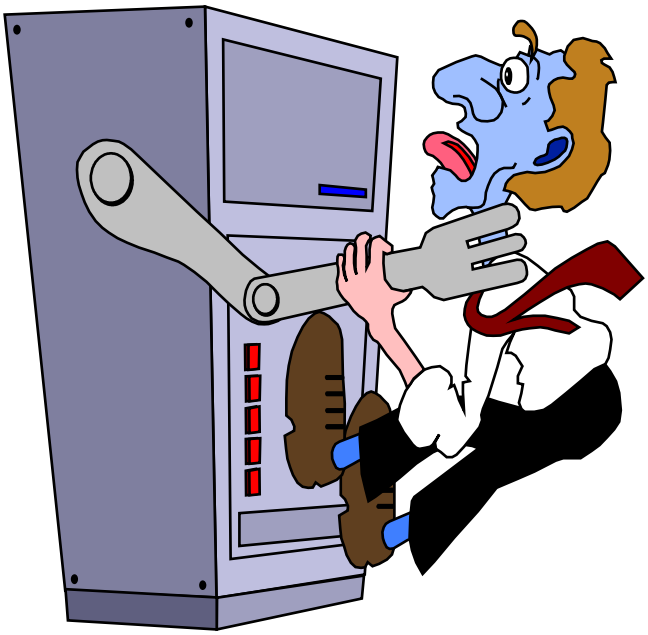
Seminar

Informationstechnologie



Grundkurs

Inhaltsverzeichnis

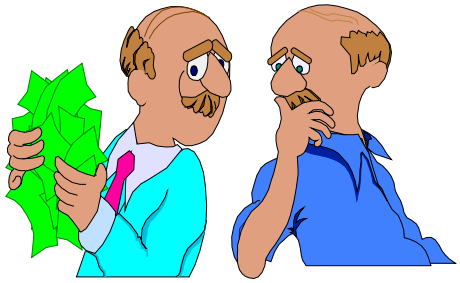
Begriffliche Grundlagen der Informatik

Informationen - Nachrichten – Daten

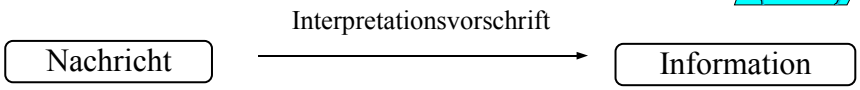
Zunächst gilt es, Begriffe wie **Informationen, Nachrichten, Daten,** Kommunikation, die für die Informationstechnologie eine wesentliche Rolle spielen, gegeneinander abzugrenzen.

Information ist Wissen, das zweck- und zielgerichtet ist.

Eine **Nachricht** ist eine weitergeleitete Information, die vom Empfänger interpretiert werden muss. Eine Nachricht besteht aus einer sinnvollen Aneinanderreihung von Zeichen und Signalen. **Nachrichten sind Träger von Informationen**.



Eine **Interpretationsvorschrift** gibt an, wie eine Nachricht zu interpretieren ist:



Eine Information kann man somit als eine Erkenntnis betrachten, die man aus einer Nachricht gewinnen kann.

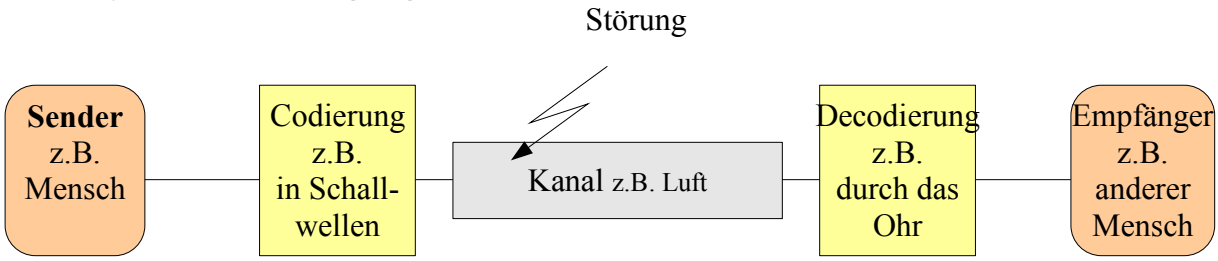
Auch **Daten** sind Träger von Informationen, allerdings zum Zweck der Weiterverarbeitung.

Den Austausch von Informationen bezeichnen wir als Kommunikation.

Für die **Übertragung einer Information** legt man fünf Stationen fest:

Sender - Codierung - Übertragungsmedium (Kanal) - Decodierung - Empfänger.
Auf dem Übertragungsweg (Kanal) kann es zu Störungen kommen, so dass je nach Art der Störung die Nachricht u. U. verstümmelt wird.

Modell für die Übertragung von Daten:



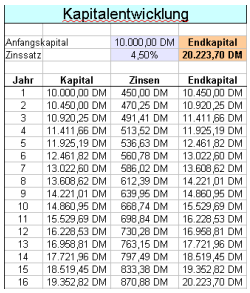
Die Informationen fließen entweder in nur einer (unidirektional), in zwei (bidirektional) oder in vielen Richtungen (multidirektional).

Informationsdarstellung

Informationen können in ganz unterschiedlichen Formen dargestellt werden:

- als gedruckter Text (wie dieses Dokument)
- Zahlen in Tabellen

- Bilder oder Grafiken (auch solche sind in diesem Dokument vorhanden)
- Töne (Sprache, Musik, Geräusche)

Text	*Foto*	*Grafik*	*Zahlen*
			

Kapitalentwicklung

Anfangskapital		10.000,00 DM	Endkapital
Zinssatz		4,50%	20.223,70 DM
Jahr	**Kapital**	**Zinsen**	**Endkapital**
1	10.000,00 DM	450,00 DM	10.450,00 DM
2	10.450,00 DM	470,25 DM	10.920,25 DM
3	10.920,25 DM	491,41 DM	11.411,66 DM
4	11.411,66 DM	513,52 DM	11.925,19 DM
5	11.925,19 DM	536,63 DM	12.461,82 DM
6	12.461,82 DM	560,78 DM	13.022,60 DM
7	13.022,60 DM	586,02 DM	13.608,62 DM
8	13.608,62 DM	612,39 DM	14.221,01 DM
9	14.221,01 DM	639,95 DM	14.860,95 DM
10	14.860,95 DM	668,74 DM	15.529,69 DM
11	15.529,69 DM	698,84 DM	16.228,53 DM
12	16.228,53 DM	730,28 DM	16.958,81 DM
13	16.958,81 DM	763,15 DM	17.721,96 DM
14	17.721,96 DM	797,49 DM	18.519,45 DM
15	18.519,45 DM	833,38 DM	19.352,82 DM
16	19.352,82 DM	870,88 DM	20.223,70 DM

Alle diese Darstellungsformen von Informationen können sowohl weitergegeben als auch weiterverarbeitet werden. Sie stellen also **Daten** im weitesten Sinn dar und sind so auch zentrale Inhalte der Informationstechnologie.

Generell unterscheiden wir zwei Arten der Darstellung solcher Daten:

analog und digital

Die Darstellung von (Mess-)Werten in einer grafischen Form nennt man **analog** (kontinuierlich, stetig veränderbar). Eine Fieberkurve oder ähnliche Diagramme gehören hierzu. Aber auch Messinstrumente mit Zeigeranzeige (z.B. Tachometer, Voltmeter), bei denen die Auslenkung des Zeigers ein Abbild der gemessenen Größe ist, gehören zu den analogen Darstellungsformen.

Das Gegenstück zur analogen ist die **digitale** Darstellung von Daten als mehrstellige Ziffernfolge eines Zahlensystems oder in tabellarischer Form. Zur Verarbeitung mit einem Computer müssen Messwerte in digitaler Form vorliegen.

Bei der Digitalisierung erfolgt eine Verschlüsselung über einen sog. Binär**code** (engl. binary code), bei der die zu codierenden Ausdrücke als logisch binäre Zeichen (0 und 1) bzw. Zeichenketten dargestellt werden. Ein einzelnes Binärzeichen (engl. binary digit) nennt man ein **Bit**. Die zusammenfasste Menge von **8 Bit ergibt 1 Byte**. Weitere Größenordnungen für binäre Maßeinheiten können den folgenden Tabellen entnommen werden. Im Gegensatz zur Bedeutung der Präfixe im Dezimalsystem bedeuten die Vielfachen von Binäreinheiten das 1024^n-fache der jeweiligen Grundeinheit. (siehe hierzu auch die entsprechenden Seiten in der Online-Enzyklopädie 'Wikipedia')

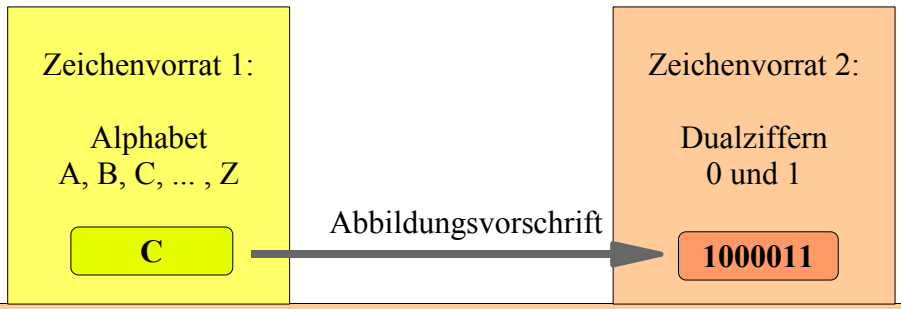
Dezimalpräfixe				Binärpräfixe:			
Symbol	Name	Wert		nächstliegende Zweierpotenz		Name	Symbol
k	kilo	10^3	1.000	2^{10}	1.024	kibi	Ki
M	mega	10^6	1.000.000	2^{20}	1.048.576	mebi	Mi
G	giga	10^9	1.000.000.000	2^{30}	1.073.741.824	gibi	Gi
T	tera	10^{12}	1.000.000.000.000	2^{40}	1.099.511.627.776	tebi	Ti

Mit Hilfe der sog. **Binärcodes** können numerische und alphanumerisch Zeichen codiert werden. Ein wichtiger Binärcode für alphanumerisch Zeichen ist der **ASCII-Code** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) bzw. unter Windows der ANSI-Code (**A**merican **N**ational **S**tandards **I**nstitute). In der folgenden Tabelle sind die Dezimalwerte der Standardtastaturzeichen angegebenen.

32	leer	48	0	64	§	80	P	96	’	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	”	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	´	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	Ä	107	k	123	ä
44	,	60	<	76	L	92	Ö	108	l	124	ö
45	-	61	=	77	M	93	Ü	109	m	125	ü
46	.	62	>	78	N	94	≠	110	n	126	ß
47	/	63	?	79	O	95	_	111	o	127	

Die Codetabelle umfasst insgesamt 256 Zeichen. Die nicht direkt über die Tastatur verfügbaren Zeichen können dennoch in ein Dokument eingefügt werden: entweder dadurch, dass man auf dem Ziffernblock den Dezimalwert bei gedrückter *ALT-Taste* eintippt und dann die *ALT-Taste* loslässt oder in der Regel einfacher über das Menü *Einfügen – Sonderzeichen* das gewünschte Zeichen auswählt.

Allgemein ist ein Code eine eindeutige Abbildungsvorschrift von Zeichen aus einem ersten Zeichenvorrat auf Zeichen aus einem anderen Zeichenvorrat.



Aufbau und Funktion einer EDV-Anlage

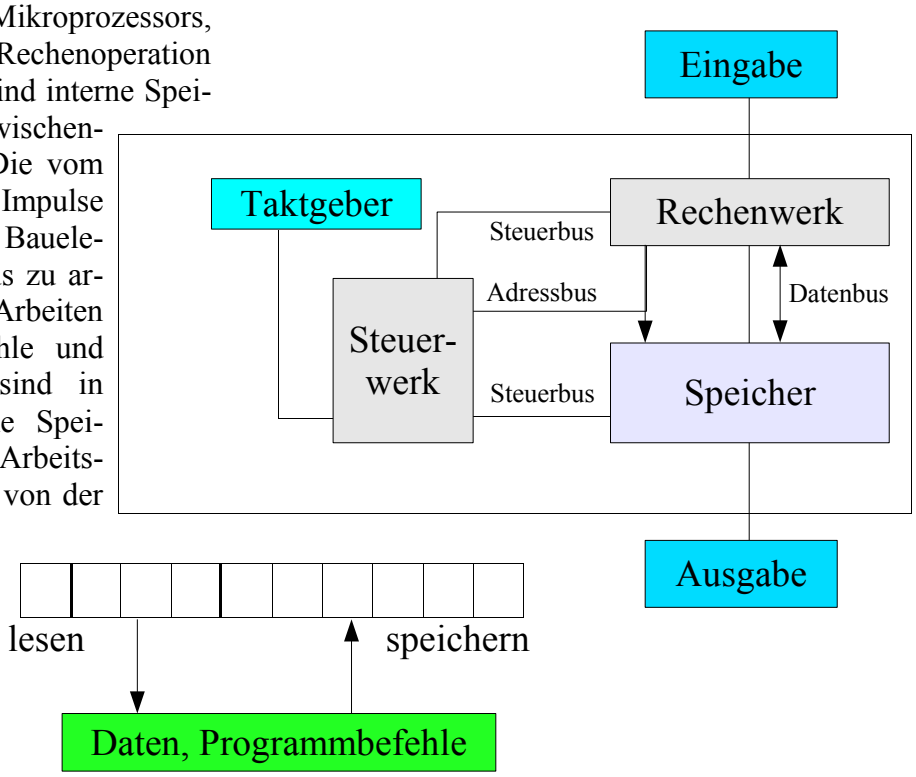
John von Neumann Konzept

Bereits im Jahr 1946 stellte der Amerikaner **John von Neumann** ein Konzept für den Bau universeller Rechner vor, das auch heute noch Gültigkeit hat:

- Jeder Rechner besteht aus Steuer- und Rechenwerk, Hauptspeicher, Ein- und Ausgabewerk.
- Der Aufbau des Computers ist unabhängig vom zu lösenden Problem.
- Programme und Daten werden im gleichen Speicher (Hauptspeicher) abgelegt.
- Der Speicher ist in gleich große, fortlaufend nummerierte Zellen unterteilt.
- Aufeinander folgende Befehle des Programms befinden sich auch in aufeinander folgenden Speicherzellen.
- Es gibt Sprungbefehle, sodass man von der eingegebenen Reihenfolge abweichen kann.
- Es gibt folgende Befehlsarten: arithmetische Befehle, logische Befehle, Transportbefehle, Sprunganweisungen, Befehle zum Warten, Schieben, Unterbrechen.
- Alle Daten werden binär codiert, die Decodierung erfolgt durch geeignete Schaltungen.

Die Zentraleinheit (CPU)

Das **Steuerwerk** ist die Schalt- und Kommandozentrale, das alle Funktionen des Computers steuert und kontrolliert. Es liest die Befehle des laufenden Programms aus dem Arbeitsspeicher und interpretiert sie. Alle logischen und mathematischen Grundoperationen werden im wichtigsten Bestandteil des **Rechenwerks**, der ALU (Abk. für engl. **A**rithmetic **L**ogic **U**nit) ausgeführt. Sie besitzt zwei Eingangsregister für die Operatoren und ein Ausgangsregister für das Ergebnis. Der Akkumulator ist das Hauptregister eines Mikroprozessors, in dem das Ergebnis einer Rechenoperation gespeichert wird. **Register** sind interne Speicher in der CPU, die zur Zwischenablage von Daten dienen. Die vom **Taktgenerator** erzeugten Impulse zwingen alle elektronischen Bauelemente im gleichen Rhythmus zu arbeiten. Um die CPU zum Arbeiten zu veranlassen, sind Befehle und Daten notwendig. Diese sind in **Speichern** abgelegt. Interne Speicher (Primär-, Haupt- oder Arbeitsspeicher) sind immer direkt von der CPU ansprechbar.



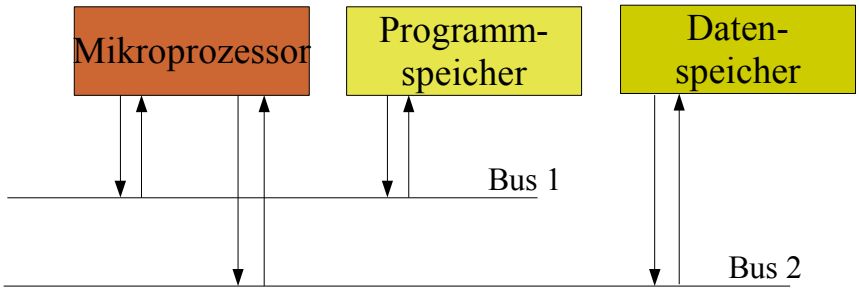
Der **Arbeitsspeicher** enthält das auszuführende Programm sowie die von ihm erzeugten Daten in Maschinensprache. In den Arbeitsspeicher gelangen Daten und Programme über die **Ein-/Ausgabeeinheit**, die auch für den Datenaustausch mit den Peripheriegeräten zuständig ist. Da die Peripheriegeräte in der Regel sehr viel langsamer arbeiten, sorgt man durch Zwischenspeicher (Cache-Speicher) für einen reibungslosen Ablauf.

Das **RAM** (Abk. für engl. **R**andom **A**ccess **M**emory) ist ein Schreib-Lese-Speicher, bei dem jede Speicherzelle einzeln adressierbar und inhaltlich veränderbar ist. Es können Daten und Programme mit sehr schnellen Zugriffszeiten gespeichert werden. RAMs, die durch Halbleiter realisiert sind, verlieren bei Stromausfall ihren Inhalt. Um dem vorzubeugen, rüstet man Computer und auch Taschenrechner mit Schaltungen aus, die bei Stromausfall oder versehentlichem Ausschalten die RAMs über Akkus oder Batterien mit Strom versorgen, so dass ihr Speicherinhalt erhalten bleibt.

ROM (Abk. für engl. **R**ead **O**nly **M**emory = NurLeseSpeicher) nennt man einen Festwertspeicher, dessen Inhalt bereits bei der Herstellung festgelegt wird und nicht mehr verändert werden kann. ROMs werden vorwiegend als Speicher für feste Programme (z.B. für Betriebssystemkomponenten) und für unveränderbare Daten verwendet.

Harvard-Architektur

Im Gegensatz zum Von-Neumann-Konzept sind bei der **Harvard-Architektur** Daten- und Befehlsspeicher physikalisch getrennt und mit unabhängigen Bussystemen organisiert. Der Vorteil dieser Architektur liegt darin, dass Befehle und die zugehörigen Daten in einem einzigen Taktzyklus in das Rechenwerk geladen werden können. Bei der Von-Neumann-Architektur sind dafür zwei aufeinander folgende Taktzyklen notwendig.



Verwendung findet die **Harvard-Architektur** vor allem in digitalen Signalprozessoren (DSP), die auf bestimmte Rechenoperationen geschwindigkeitsoptimiert sind.

Konkrete Einsatzbereiche sind Handy, CD-Player, Digitales Fernsehen, Waschmaschinen, ABS (Auto), Barcodelesegeräte und viele mehr.

Übersicht über die Peripheriegeräte eines Computers

Eingabegeräte	*Ausgabegeräte*	*Speichergeräte*
Tastatur	Monitor	Magnetspeicher
Maus	Drucker	Optische Speicher
Scanner	Plotter	Halbleiterspeicher
Soundkarte	Sprachausgabegeräte	
Videokarte		
:	:	:

Betriebssystem

Nach dem Einschalten eines Computers dauert es zunächst eine Weile bis das Gerät einsatzbereit ist. Wie lange das dauert, hängt von verschiedenen Faktoren ab. Während dieser Zeit erscheinen am Bildschirm verschiedene Systemmeldungen. Was danach auf dem Bildschirm sichtbar ist, hängt vom verwendeten Betriebssystem ab.

Das **Betriebssystem** (engl. **O**perating **S**ystem) ist eine Gruppe von Programmen, die alle organisatorischen Abläufe innerhalb eines Computers oder der angeschlossenen Peripheriegeräte steuert und verwaltet. DIN 44 300 definiert ein Betriebssystem wie folgt: „Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen. Eine Sprache, der ein Betriebssystem gehorcht, heißt Betriebssprache."

Bekannte Betriebssysteme: DOS, Linux, OS/2, UNIX, Windows (95/98, ME, NT, 2000, XP, Vista).

Aus Benutzersicht ist das Betriebssystem die Software, die als Mittler zwischen dem Anwender auf der einen Seite und der Hardware sowie den Anwendungsprogrammen auf der anderen Seite fungiert. Ein Programmierer, der ein Programm in einer höheren Programmiersprache auf einer Rechenanlage ausführen möchte, braucht sich in der Regel nicht darum zu sorgen, wie verwendete Variablen in Adressen des Speichers umgesetzt werden, an welcher Stelle des Speichers sein Programm während der Ausführung steht, wie und wo Daten auf Hintergrundspeichern (Speicherhierarchie) abgelegt werden, in welcher Reihenfolge verschiedene Programme ablaufen oder wie seine Programme und Daten vor fremdem Zugriff geschützt werden. Diese und weitere Verwaltungsaufgaben übernimmt das Betriebssystem.

Die **Aufgaben** des Betriebssystems im Einzelnen:

- **Booten des Rechners**
- **Speicherverwaltung**
- **Ein- und Ausgabesteuerung**
- **Verwalten des File-Systems**
- **Ressourcenverwaltung**
- **Zeitgeberfunktion**
- **Treiberorganisation**

Zur Bewältigung der vielfältigen Aufgaben verfügt das Betriebssystem über zahlreiche Komponenten, die jeweils eine oder mehrere Teilaufgaben wahrnehmen. Die drei wichtigsten Komponenten sind **Organisationsprogramme, Dienstprogramme, Übersetzungsprogramme.**

Organisationsprogramme: In der Speicherverwaltung übernehmen Organisationsprogramme die Lösung der folgenden Aufgaben:

- Kontrolle aller im System vorkommenden Speicher
- Zuteilung von Speicher an Benutzerprogramme
- Organisation von Speicherhierarchien
- Kommunikation mit anderen Rechenanlagen.

Die Organisationsprogramme der Prozessorverwaltung veranlassen die Zuteilung des Prozessors an eines der zu bearbeitenden Programme. Die Organisationsprogramme der Geräteverwaltung koordinieren die Auswahl und Bereitstellung der für Eingabe bzw. Ausgabe geeigneten Geräte entsprechend den Anforderungen der Benutzerprogramme, die Anpassung an die speziellen physikalischen

Eigenschaften der Geräte und die Überwachung der Datenübertragung zwischen Programm und Gerät. Zunehmend an Bedeutung gewinnen die Organisationsprogramme für die Kommunikation zwischen Programmen, Geräten oder Dateien. Sie sind Voraussetzung für Dienste in Rechnernetzen (z. B. elektronische Post) und allgemein für die Datenübertragung.

Übersetzungsprogramme: (Übersetzer): Diese übersetzen Programme höherer Programmiersprachen in auf einer Rechenanlage ausführbare Programme.

Dienstprogramme: Diese lösen Standardanwendungsprobleme. Typische Dienstprogramme sind z.B. Sortierprogramme (Sortieren), Dateiverwaltungsprogramme, Lader, Binder, Editor, Debugger.

Hinzu kommen Statistik- und Abrechnungsprogramme, um z.B. die Auslastung der Geräte oder die Benutzungsgebühren zu ermitteln.

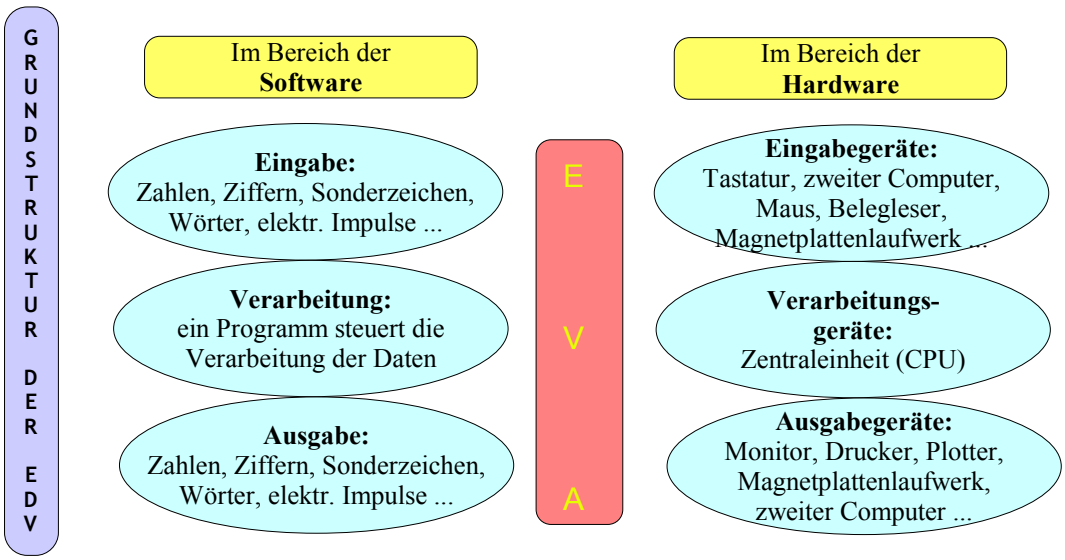
Zusätzliche Anforderungen an Betriebssysteme entstehen, wenn die Komponenten des Betriebssystems über verschiedene Rechner verteilt werden. Verteilte Programme zur Verwaltung von Datenbanken und dezentral aufgestellten Geräten sind für moderne Computer bereits verfügbar.

Zur Kommunikation mit dem Betriebssystem dient eine Kommandosprache. Mit ihrer Hilfe kann der Benutzer Aufträge formulieren.

Betriebssystem		
Organisation-programme	**Übersetzungs-programme**	**Dienstprogramme**
Speicherverwaltung	BASIC-Interpreter	Lader
Geräteverwaltung	PASCAL-Übersetzer	Sorter
Kommunikation	Compiler	Binder
:	:	:

Grundstruktur der Datenverarbeitung

Das Grundprinzip der Datenverarbeitung sowohl bei der Hardware als auch bei der Software ist das sogenannte **EVA-Prinzip (Eingabe - Verarbeitung – Ausgabe)**.



Die **Eingabe** ist die Versorgung der CPU (Central Processing Unit = Zentraleinheit) eines Rechners mit Daten. Die Eingabe kann manuell **von geeigneten Peripheriegeräten** erfolgen (z.B. Tastatur, Maus) oder automatisch durch Überspielen oder Erfassen von Daten mit Hilfe entsprechender Transport-, Speicher- und Erfassungsmedien (z.B. Diskette, Magnetband, Scanner). Daten sind dabei sowohl Verarbeitungsdaten z.B. Texte, Kalkulationstabellen als auch Programmdaten.

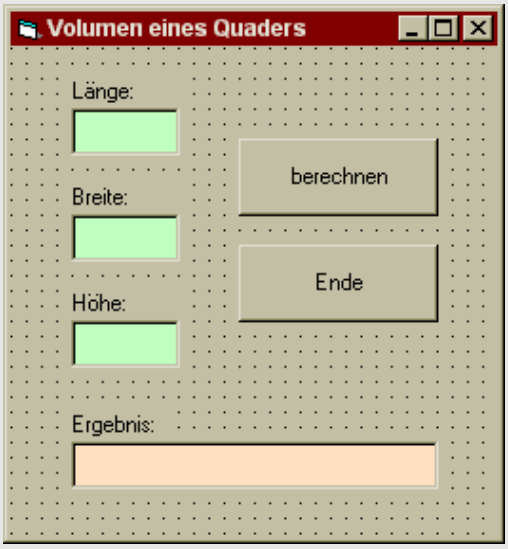
Die **Verarbeitung** der Daten erfolgt in der zentralen Funktionseinheit, der **CPU** ***(engl. Central Prozessing Unit).***

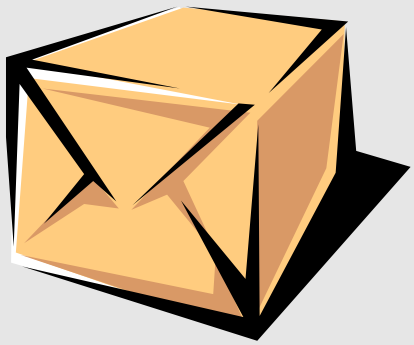
Ausgabe ***(engl. output)*** nennt man die Weiterleitung von Daten **an Peripheriegeräte** (Geräte in der Umgebung der Zentraleinheit) wie Bildschirm, Drucker oder Speichermedium.

Auch jeder Software liegt dieses fundamentale Prinzip zugrunde, sowohl beim Programmieren selbst als auch bei der Anwendung. Jedes Programm benutzt sowohl Ein- und Ausgabeschnittstellen als auch Ereignisse, die bestimmte Verarbeitungsschritte auslösen.

Programmbeispiel zur Demonstration des EVA-Prinzips:

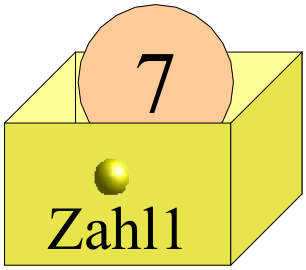
```
Private Sub cmdBerechnen_Click()
Dim l as Single, b as Single, h as Single
Dim V as Single, O as Single

l = txtLänge.Text
b = txtBreite.Text
h = txtHöhe.Text

V = l * b * h
O = 2 * (l * b + l * h + b * h)

txtErgebnis.Text = "Volumen:" & V & "        " & "Oberfläche:" & O
End Sub
```





Wenn man Daten in den Computer eingibt, werden sie im Arbeitsspeicher abgelegt. Diese Speicherzellen sind mit einem Schubfach in einem Schranksystem vergleichbar. Weil ein Computer über sehr viele Speicherstellen verfügt, gibt man ihnen **symbolische Namen**, um später die Daten leichter wieder finden zu können.

Bleibt der Inhalt einer Speicherstelle immer gleich, spricht man von einer **Konstanten** (z.B. Kreiszahl π: ***Const Pi = 3.14***).



Werden im Laufe der Datenverarbeitung in einer Speicherstelle verschiedene Daten abgelegt, so liegt eine **Variable** vor.

Dabei ist zu beachten, dass der alte Inhalt verschwindet, sobald eine Variable mit einem neuen Wert belegt wird.

Ein Computer besitzt eine ungeheuer große Anzahl solcher Speicherstellen. Da sie während der Arbeit am Computer benötigt werden, nennt man ihn Arbeitsspeicher. Die Inhalte des Arbeitsspeichers werden gelöscht, sobald man die Stromzufuhr ausschaltet. Man spricht deshalb auch von einem flüchtigen Speicher. Für dauerhafte Speicherung benötigt man andere Medien: z.B. Festplatte, Memory-Stick, CD-Rom.

Problemlösungsmethoden

Grundsätzlich stellt auch die Informatik Methoden zur Lösung von Problemen bereit. An erster Stelle steht eine möglichst präzise Formulierung des Problems mit anschließender Analyse. Danach kommt die Wahl einer geeigneten Methode gefolgt von einem i.a. schriftlicher Entwurf. Dieser Entwurf muss nun noch praktisch umgesetzt und damit verifiziert werden. Man spricht von Realisierung bzw. Implementierung. Abschließend sollte noch eine Bewertung der gewählten Problemlösung erfolgen.

Problemstellung
Analyse
Wahl der Problemlösungsmethode
Entwurf
Realisierung
Bewertung

Für die Problemlösung haben sich folgende drei Methoden herauskristallisiert:

- **ablauforientiert**
  Das bedeutet in der Praxis den Einsatz von Flussdiagrammen bzw. Programmablaufplänen, mit deren Hilfe die Problemlösungsschritte übersichtlich in graphischer Form dargestellt werden können. Das Gesamtproblem lässt sich hiermit in kleine, überschaubare Schritte zerlegen.
- **objektorientiert**
  Hierbei handelt es sich um eine Philosophie des Programmierens, bei der keine Unterschiede zwischen Daten und Aktionen gemacht werden und diese grundsätzlich als Objekte betrachtet werden. Dabei bedeutet Objekt ein nach außen hin abgeschlossenes Gebilde, das aus Daten und Algorithmen besteht.
- **regelbasiert**
  Hier stehen Fakten und Regeln im Vordergrund, die auf die Problemstellung angewandt werden und damit Schlussfolgerungen für die Problemlösung gezogen werden.

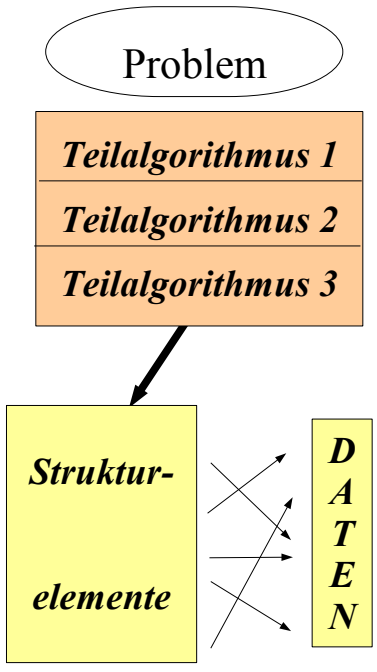
Basiskonzepte

In H. Balzerts 'Lehrbuch der Softwaretechnik' werden folgende grundlegenden Konzepte zu einer möglichen Problemlösung in der Informatik dargestellt:

Konzepte und Sichten	*Notationsformen*
Algorithmische Sicht	Pseudocode, Kontrollstrukturen, Struktogramm, Programmablaufplan (PAP)
Objektorientierte Sicht	Klassenstrukturen, Klassendiagramm
Datenorientierte Sicht	Entitäten und Beziehungen, Datenstrukturen
Regelbasierte Sicht	Wenn-Dann-Regeln, Entscheidungstabelle
Zustandsorientierte Sicht	Zustandsdiagramm, nebenläufige Strukturen
Funktionale Sicht	Datenflussdiagramm, Funktionsbaum

Bis Mitte der neunziger Jahre wurde in der Informatik der Realschule das **algorithmische bzw. prozedurale Problemlösen** (geht zurück auf die siebziger Jahre) bevorzugt, um nicht zu sagen, es war eigentlich die einzige Sicht, aus der die Lösung eines Problems in der Schule angegangen wurde. Dabei konnte das Problem durch **Strukturieren** in überschaubare Teile zerlegt und mit Hilfe einfacher **Algorithmen** gelöst werden. Die Verfeinerung des Problems führt zu einigen wenigen **Grundstrukturen**, die in ihrer Funktion unabhängig vom zu lösenden Problem und damit nahezu allgemein gültig sind. Durch Kombination mehrerer dieser Grundstrukturen lassen sich selbst komplexe Vorgänge beschreiben und bearbeiten (Methode der strukturierten Problemlösung). Jeder Algorithmus operiert auf Daten, die dem Problem angepasst sind.

Bei dieser Art des Problemlösens zeigt sich immer wieder folgende Schwierigkeit, dass trotz Strukturierung der Problemlösende das Gesamtkonzept kennen bzw. erkennen und im Auge behalten und bei jeder Teillösung die Zweckdienlichkeit für die Gesamtlösung bedenken muss. Nachträgliche Änderungen der Daten erfordern aufwendige Überarbeitung aller darauf operierenden Teilalgorithmen.



Da inzwischen die graphischen Darstellungsmöglichkeiten eines Computers sehr stark ausgebaut worden sind und sich auch die Softwareentwicklung dem angepasst hat, ist die **objektorientierte Sichtweise** gegenwärtig die am häufigsten Gebräuchliche bei den Entwicklungsumgebungen. Dem will auch die Realschule gerecht werden und dem Schüler diese **objektorientierte Problemlösungsmethode** vorstellen.

Um den Unterschied zwischen den beiden Sichtweisen etwas deutlicher zu machen, nebenstehend zunächst eine graphische Veranschaulichung der algorithmischen Problemlösung.

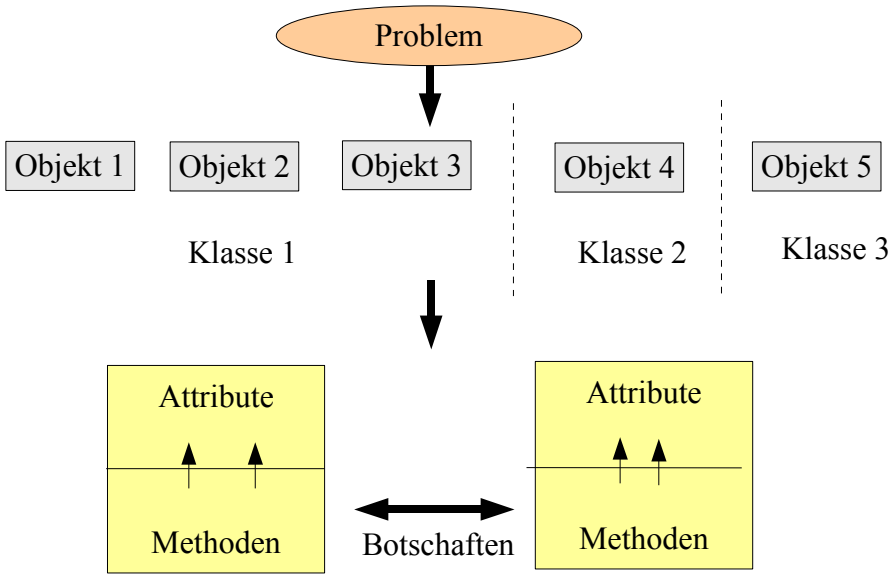
Bei der objektorientierten Sichtweise hingegen werden die das Problem tragenden **Objekte** zunächst analysiert. Für jedes Objekt werden die für das Problem wesentlichen **Eigenschaften (Attribute)** festgelegt. Neben diesen aktuellen Attributwerten besitzt ein Objekt auch noch gewisse Fähigkeiten, die man hier **Methoden bzw. Operationen** nennt. Damit kann es auf seinen Daten

operieren oder auch mit sich Handlungen ausführen.

Auf die Attribute eines Objekts kann nur über seine Methoden zugegriffen werden **(Datenkapselung)**. Die Objekte ihrerseits können sich gegenseitig **Botschaften** schicken, auf die der Empfänger wiederum mit seinen Methoden reagieren kann. Der Ablauf der Problemlösung ergibt sich durch den dynamischen Austausch der Botschaften.

Vielleicht sollte an dieser Stelle nicht unerwähnt bleiben, dass der Unterschied zwischen **algorithmischer und objektorientierter Problemlösung** bei einfachen, überschaubaren Problemen als eher gering betrachtet werden kann und nicht sehr stark ins Gewicht fällt, jedoch bei komplexeren Aufgabenstellungen der objektorientierten Sicht heute der Vorzug gegeben wird. Die Art, wie man an ein Problem herangeht, ist bei beiden Methoden höchst unterschiedlich.

Der **Vorteil** dieser Methode liegt darin, dass selbst bei komplexen Problemstellungen die einzelnen Objekte mit ihren Attributen und Methoden überschaubare Teilbereiche darstellen. Die Dynamik des Problems lässt sich durch einfaches Senden von Botschaften verständlicher realisieren. Nachträgliche Änderungen an der Objektstruktur bzw. an der Realisierung der Methoden beeinträchtigen nicht das Gesamtkonzept bzw. andere Teilbereiche und sind somit sehr viel leichter möglich als bei der algorithmischen Problemlösung.



Die ***objektorientierte Softwareentwicklung*** basiert im wesentlichen auf folgenden sieben Grundkonzepten:

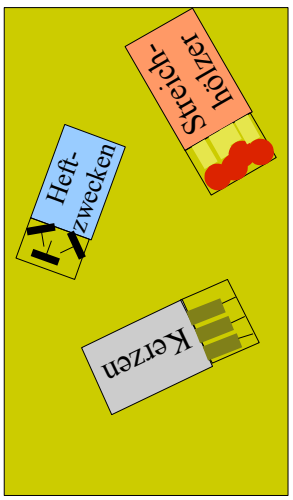
- **Objekte**
- **Klassen**
- **Attribute**
- **Methoden (Operationen)**
- **Botschaften (Ereignisse)**
- **Vererbung**
- **Polymorphismus**

1. Ein Objekt (Exemplar, Instanz) ist ein individuelles Exemplar von Dingen, Personen oder

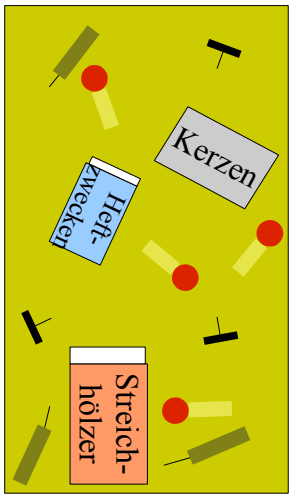
Begriffen. Es besitzt eine Objektidentität.

2. **Klassen fassen Objekte mit gleichen Attributen und Methoden zusammen.** Klassenattribute beschreiben die Eigenschaften von Klassen. Klassenoperationen sind den Klassen zugeordnet. Durch Verkapselung kann auf Attributwerte nur durch die Methoden zugegriffen werden.
3. **Attribute (Eigenschaften) beschreiben die Merkmale eines Objekts.** Die aktuellen Werte, die ein Objekt gerade besitzt, heißen **Attributwerte** (Eigenschaftswerte).
4. **Methoden (Operationen) beschreiben das Verhalten eines Objekts**, d.h. die Dienstleistungen, die es seiner Umwelt oder sich selbst zur Verfügung stellt. Methoden kommunizieren mit der Umwelt über Ein-/Ausgabeparameter. Die fachliche Semantik (Wortbedeutung) von Methoden wird in Spezifikationen festgelegt.
5. Durch **Botschaften (Ereignisse) kommunizieren Objekte und Klassen untereinander.**
6. **Durch die Vererbung werden Attribute und Methoden an alle Unterklassen einer Oberklasse weitergegeben.** Es entsteht eine Klassenhierarchie. Geerbte Methoden können in Unterklassen redefiniert werden. Eine Einfachvererbung liegt vor, wenn jede Unterklasse nur eine direkte Oberklasse besitzt, sonst handelt es sich um eine Mehrfachvererbung. Klassen, einschließlich ihrer Vererbung, werden in einem Klassendiagramm dargestellt.
7. **Polymorphismus (Vielgestaltigkeit)** erlaubt es, gleiche Botschaften an Objekte unterschiedlicher Klassen zu senden, die ihrerseits zwar prinzipiell gleichartig darauf reagieren, jedoch in unterschiedlichen Ausprägungen.

Dass die objektorientierte Denkweise nicht nur bei Schülern sondern auch bei Erwachsenen tief verwurzelt ist, soll das folgende Experiment (von K. Duncker) verdeutlichen:



Mehrere Versuchspersonen erhielten einzeln die Aufgabe, an einer Wand in Augenhöhe nebeneinander drei Kerzen zu befestigen und anzuzünden. Hierfür stand den Probanden eine Reihe von Gegenständen zur Verfügung. Unter den meist nutzlosen Dingen befanden sich auch einige für die Lösung brauchbare: Heftzwecken, Streichhölzer und drei kleine in Farbe und Größe etwas unterschiedliche Pappschachteln von der Form einer Streichholzschachtel.



Lösung der Aufgabe: Mit je einer Heftzwecke werden zunächst die Pappschachteln an der Wand befestigt; sie dienen den Kerzen als Standflächen. Anschließend werden die Kerzen angezündet und mit etwas Wachs auf den Schachteln festgeklebt.

Die Versuchspersonen mussten diese Aufgabe in zwei leicht unterschiedlichen Ausgangssituationen lösen: Bei den Personen der einen Gruppe waren die drei Pappschachteln mit Versuchsmaterialien gefüllt, die erste mit den Kerzen, die zweite mit Heftzwecken und die dritte mit Streichhölzern. Bei der zweiten Gruppe waren die Schachteln leer. Kerzen, Heftzwecken und Streichhölzer lagen hier auf dem Tisch verstreut.

Erstaunlicherweise wurde die Aufgabe von der zweiten Gruppe signifikant häufiger und schneller

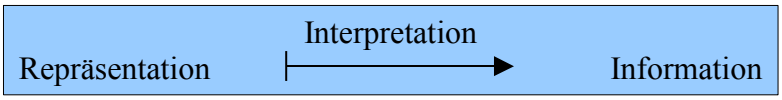
gelöst als von der ersten. Duncker erklärt dieses Ergebnis wie folgt: Die erste Gruppe lernt die Schachteln als Behälter für Kerzen, Heftzwecken und Streichhölzer kennen. Diese Funktion 'Behälter' ist anschließend so eng mit den Schachteln verknüpft, dass die Probanden ihr Denken kaum noch davon lösen können und unfähig sind, die Schachteln zu einem völlig anderen Zweck zu nutzen, nämlich als Standfläche für die Kerzen. Duncker nennt dieses Phänomen *funktionale Gebundenheit.* Die zweite Gruppe nimmt die Schachteln ohne die funktionale Bindung 'Behälter' wahr. Die Versuchspersonen können sie daher völlig frei auch zu scheinbar ungewöhnlichen Zwecken (z.B. als Standfläche) einsetzen.

Was bedeuten diese Beobachtungen aus Sicht der Informatik? Die Versuchspersonen denken offenbar objektorientiert: Gegenstände werden zuerst unter diesem Gesichtspunkt in Klassen eingeteilt, welche Operationen mit ihnen möglich sind. Eine Schachtel gehört damit für die erste Gruppe zur Klasse der Objekte, auf denen Operationen wie "öffnen" und "schließen" erlaubt sind und die einen Zustand wie 'leer' oder 'gefüllt' besitzen. Diese Operationen bestimmen das Denken. Dabei übersieht diese Gruppe, im Gegensatz zur zweiten, dass Schachteln auch als Objekte einer Oberklasse mit allgemeineren Eigenschaften aufgefasst werden können, etwa als Objekte der Klasse quaderförmiger, flacher Gegenstände mit Operationen wie 'als Unterlage verwenden', 'stapeln' usw.

Modellierung

Informationen darstellen

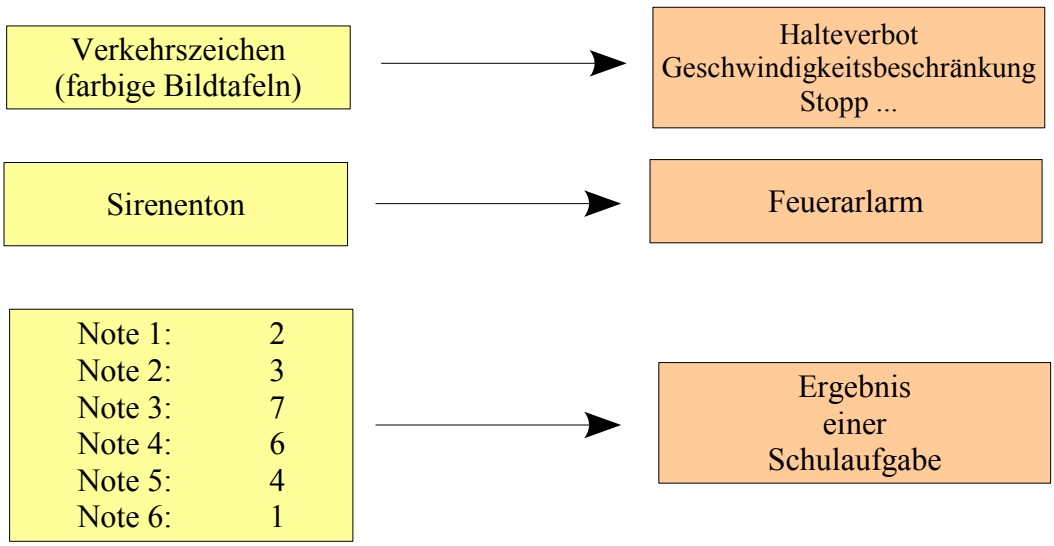
Einer (stets konkreten) Repräsentation wird durch eine Interpretation eine (stets abstrakte) Information zugeordnet:



Eine Information zusammen mit ihrer Repräsentation bezeichnet man als Objekt bzw. Datum. Somit sind Repräsentationen ebenfalls Träger von Informationen.

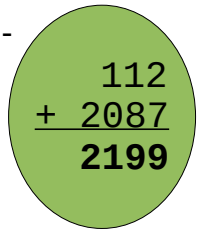
Beispiele:

konkretes Phänomen der realen Welt — abstrakte Idee, Konzept



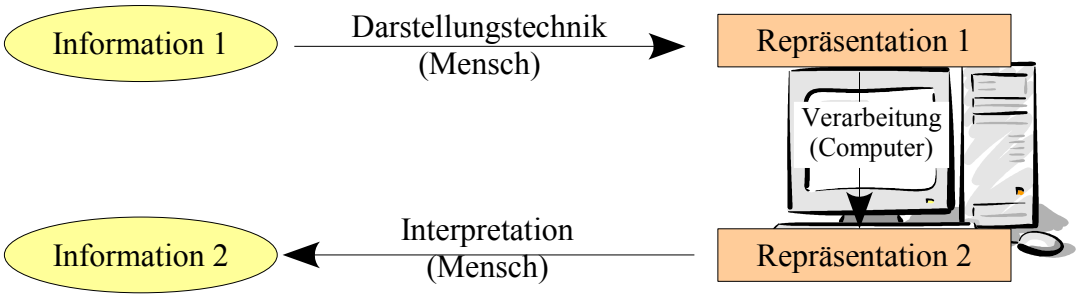
Dass die drei Begriffe Repräsentation, Interpretation und Information untrennbar miteinander verbunden sind, ist offensichtlich: Jede Information bedarf einer Repräsentation, sonst kann sie weder mitgeteilt, noch gespeichert, noch sonst wie verarbeitet werden. Umgekehrt ist jede Repräsentation ohne zugeordnete Information im wahrsten Sinn des Wortes sinnlos. Verarbeitungsprozesse operieren auf Repräsentationen.

Wie wichtig eine gut gewählte Repräsentation ist, wird am Beispiel der Zahlen besonders deutlich. Auf Grund der abstrakten Idee der Zahlen gilt *z.B. 'einhundertzwölf' plus 'zweitausendsiebenundachzig' ergibt 'zweitausendeinhundertneunundneunzig'*. Es wäre völlig aussichtslos, sich solche Fakten abstrakt merken zu wollen. Erst auf der Basis geschickt gewählter Repräsentationen, hier das Dezimalsystem, kann man mit solchen Problemen umgehen.

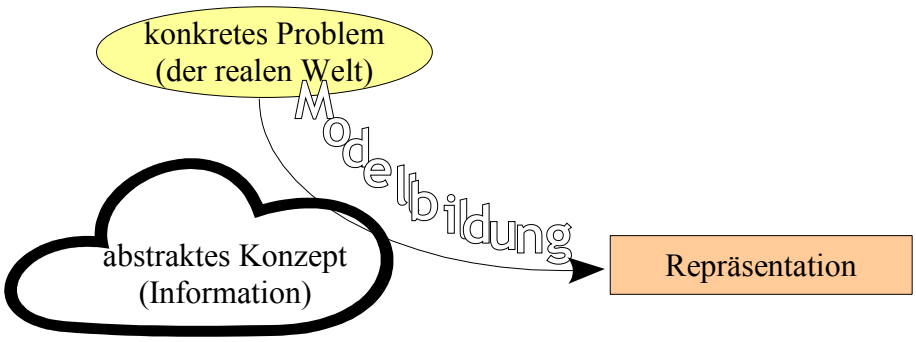


Die Geschichte der Naturwissenschaften belegt eindrucksvoll die Bedeutung guter Repräsentationen. So war z.B. das römische Zahlensystem vor ca. 2000 Jahren ein wahrer Hemmschuh für herausragende naturwissenschaftliche Leistungen.

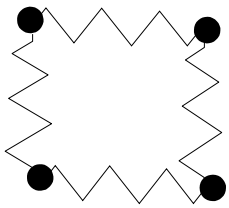
Informationen müssen vor ihrer Weiterverarbeitung dargestellt werden:



Die gewählten Repräsentationen stellen stets ein Modell der Wirklichkeit dar (siehe Abschnitt 'System und Modell').



Beispiele:

konkretes Problem	***abstraktes Konzept***	***Repräsentation***
Wie sieht ein Festkörper in seinem Inneren aus?	Kugeln mit Abstoßungs- bzw. Anziehungskräften	

konkretes Problem	*abstraktes Konzept*	*Repräsentation*
Wie lang ist die Seitenlänge eines Quadrats mit einer Fläche von 2 cm²?	Seite * Seite = 2 cm²	$a^2 = 2$ $a = \sqrt{2}$
Das zweite Beispiel zeigt, dass Verarbeitungsvorschriften, also auch Programme, ebenfalls Modelle darstellen.		

Modellierungstechniken

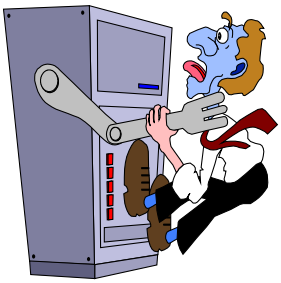
Zur Modellierung informatischer System stehen verschiedene Modellierungstechniken zur Verfügung. Aus der Vielzahl dieser Techniken und ihrer Notationsformen wollen wir uns in diesem Kurs auf einige wenige beschränken. Dabei soll als Notationsformen UML (= Unified Modelling Language) Anwendung finden. Die UML gilt heute als Standard für Analyse und Design objektorientierter Anwendungen. Die Notation der UML umfasst Diagramme für die Darstellung der verschiedenen Ansichten auf das System, vergleichbar mit Bauplänen für Häuser. Auch hier gibt es z. B. einen Grundriss, einen Lageplan, verschiedene Außenansichten und Werkpläne für die Handwerker. Für eine spezielle Aufgabe ist meist eine Diagrammart besser geeignet als die anderen.

UML 2.0 unterstützt 13 Diagrammtypen, die man in 2 Kategorien unterteilen kann *(in Klammern sind jeweils die englischen Bezeichnungen angegeben)*:

- **Strukturdiagramme**
  - → Klassendiagramm *(class diagram)*
  - → **Objektdiagramm *(object diagram)***
  - → Komponentendiagramm *(component diagram)*
  - → **Kompositionsstrukturdiagramm *(composite structure diagram)***
  - → **Verteilungsdiagramm *(deployment diagram)***
  - → **Paketdiagramm *(package diagram)***
- **Verhaltensdiagramme**
  - → **Anwendungsfalldiagramm *(use case diagram)***
  - → **Zustandsdiagramm *(state chart diagram)***
  - → **Aktivitätsdiagramm *(activity diagram)***
  - → **Sequenzdiagramm *(sequence diagram)***
  - → **Interaktionsübersichtsdiagramm *(interaction overview diagram)***
  - → **Kommunikationsdiagramm *(communiction diagram)***
  - → **Zeitverlaufsdiagramm** *(timing diagram)*

Für den schulischen Bedarf reichen in aller Regel **Klassen- und Objektdiagramm, das Anwendungsfall- und das Sequenzdiagramm** aus, wobei den Objekt- und Klassendiagrammen sicher die größte Bedeutung zukommt. Natürlich verwenden wir auch weiterhin Struktogramme und Flussdiagramme (Programmablaufpläne), die nicht der objektorientierten, sondern der algorithmischen Modellierung zugerechnet werden. In diesem Kapitel werden diese Modellierungsmöglichkeiten exemplarisch behandelt. Auch in allen andern Kapiteln sind entsprechende Diagramme eingearbeitet. Eine Notationsübersicht findet man im Internet unter http://www.oose.de/downloads/uml-2-Notationsuebersicht-oose.de.pdf.

System und Modell



Damit man auch außerhalb der Ladenöffnungszeit sich mit Nahrung bzw. Getränken versorgen kann, gibt es entsprechende Automaten. Auch in vielen Schulen ist ein Getränkeautomat aufgestellt. Jeder weiß, wie solch ein Gerät funktioniert.

Damit alles reibungslos funktioniert, müssen verschiedene Komponenten zusammenwirken. Sie bilden ein **System**. Beispielsweise kann man den

Motor eines Autos als System bezeichnen oder auch das ganze Auto. Dabei muss es sich nicht unbedingt um materielle Gegenstände oder konkrete Dinge handeln (Gleissystem der Deutschen Bahn), auch unser **Zahlensystem**, das **Zahlungs- bzw. Währungssystem** oder **Anwenderprogramme am Computer** sind Beispiele für Systeme.

Als System bezeichnen wir eine Menge von Systembestandteilen (Komponenten), die miteinander in Beziehung bzw. Wechselwirkung stehen.

Zurück zum Getränkeautomaten. Will eine entsprechende Firma ein solches Gerät neu herstellen, so wird man sich zunächst überlegen, aus welchen Einzelteilen (**Systemkomponenten**) es besteht. Damit wird zunächst auf dem Papier ein Plan entworfen, wie die einzelnen Komponenten zusammenwirken sollen. Dieser Plan stellt sozusagen ein **Modell** des herzustellenden Gesamtsystems dar. Dabei können nebensächliche Teile, die das spätere Gerät vielleicht enthalten wird, die aber für die Funktion unerheblich sind, vernachlässigt, also weggelassen werden.

Modelle werden in unserer Zeit überall in unserer Umwelt und vor allem in den verschiedenen Wissenschaftsdisziplinen angewendet. Beispielsweise erklärt man bestimmte Phänomene des Lichts mit dem **Modell des Lichtstrahls**, der Aufbau der Materie wird mit dem **Atommodell** beschrieben, das **Teilchenmodell** hilft bei der Beschreibung von Erscheinungen in der Wärmelehre. Immer dann, wenn es schwierig ist, bestimmte Sachverhalte zu erklären, verwenden wir ein Modell.

Modelle erleichtern das Verständnis komplizierter Zusammenhänge

Modelle umfassen detaillierte Pläne, aber ebenso oberflächliche Gesamtansichten eines Systems. Ein gutes Modell besteht aus Elementen, die wichtig sind für die gewählte Sichtweise und lässt jene Elemente weg, die für die momentane Betrachtungsebene nicht relevant sind. Ein System wird in der Regel also durch mehrere Modelle beschrieben, wobei jedes Modell eine eigene Sicht auf das System bildet und in sich konsistent und abgeschlossen ist.

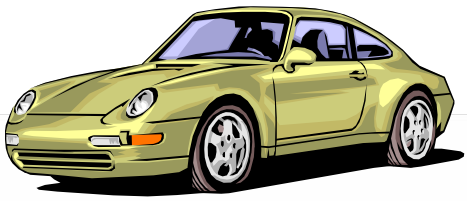
Ein Modell ist eine vereinfachte Beschreibung eines realen, geplanten oder gedachten Systems, das jedoch nicht alle Eigenschaften des Vorbilds aufweisen muss.
Das Erstellen eines Abbildes eines gedachten oder realen Objekts bzw. Systems bezeichnet man als Modellierung.

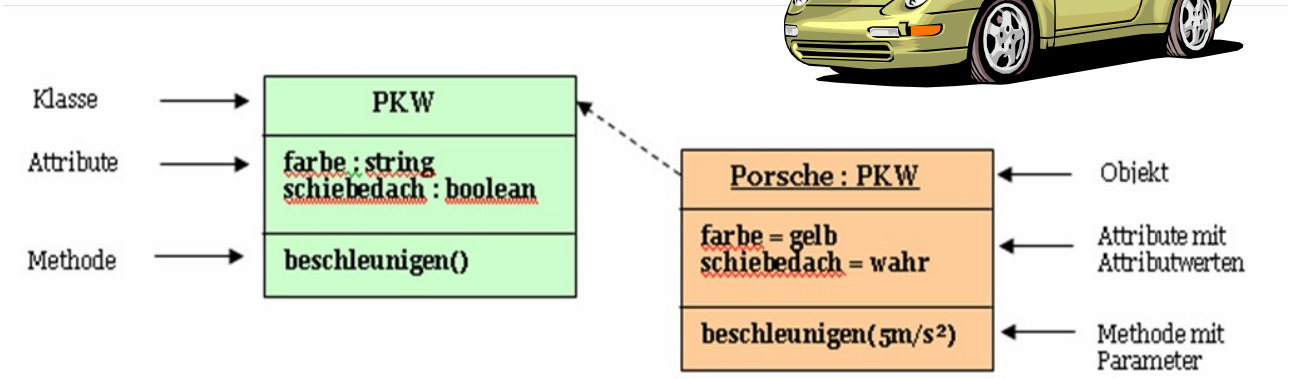
Objektorientierte Modellierung

Die objektorientierte Sichtweise kommt der natürlichen (allerdings überwiegend unbewussten) Art des Menschen, seine Umwelt wahrzunehmen und auf diese Wahrnehmungen zu reagieren, besonders nahe (siehe auch Kapitel 'Problemlösungsmethoden').

Bei der objektorientierten Sichtweise spielen die Begriffe
Klasse, Objekt, Attribut, Methode und Ereignis
die tragenden Rollen.

Bei der objektorientierten Modellierung werden Objekte identifiziert und analysiert, ihre Attribute bestimmt und ihre Methoden erarbeitet. Bei der Modellierung und Implementierung, zu der sich naturgemäß objektorientierte Programmiersprachen (nicht zu verwechseln mit lediglich visuellen Programmiersprachen) besonders eignen, ist sorgfältig zwischen Klassen (als abstrakten Beschreibungen) und Instanzen/Objekten (als konkreten Ausprägungen) von Klassen zu unterscheiden. Anhand eines einfachen Beispiels soll dies demonstriert werden.

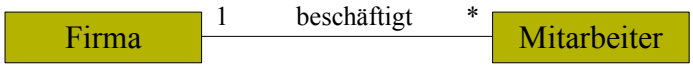




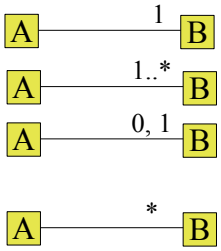
Sowohl beim **Objekt- als auch beim Klassendiagramm** verwendet man ein dreigeteiltes Rechteck, bei dem im oberen Teil der Objekt- bzw. Klassenname steht. Der mittlere Teil enthält die für das jeweilige Modell notwendigen Attribute, der dritte Teil die betrachteten Methoden (Operationen). Im Objektdiagramm wird der Objektname (einschließlich der Klasse) unterstrichen, den Attributen werden Attributwerte zugeordnet, die Methoden werden meistens nicht mit aufgenommen. Auch ohne Parameter werden Methoden durch das Hinzufügen von Klammern gekennzeichnet, um sie so leicht von Attributen unterscheiden zu können. Attribute und Methoden werden klein geschrieben.

Verzichtet wurde auf die Möglichkeit, bei den Attributen und Methoden das sog. Sichtbarkeitskennzeichen (- für private, + für public und # für protected) voranzustellen, da sie für den Gebrauch in der Realschule von untergeordneter Bedeutung sind.

Stehen mehrere Klassen bzw. deren Objekte in Beziehung zueinander, so spricht man von einer **Assoziation**. Dargestellt wird sie durch eine verbindende Linie zwischen den beteiligten Klassen. Im einfachsten Fall sind nur 2 Klassen beteiligt:



Jeder Beziehung kann – wie in diesem Beispiel – eine **Kardinalität** (auch **Multiplizität** oder **Vielfachheit** genannt) für die an der Beziehung beteiligten Klassen mitgegeben werden. Für die Kardinalitäten gelten folgende Schreibweisen:



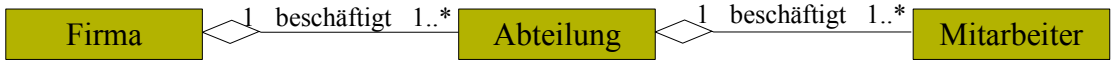
Genau ein B-Objekt steht in Beziehung zum A-Objekt

Mindestens ein B-Objekt steht in Beziehung zum A-Objekt

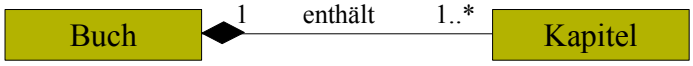
Kein oder genau ein B-Objekt steht in Beziehung zum A-Objekt

Keine Einschränkung bezüglich der Anzahl der B-Objekte

Ist eine Beziehung so geartet, dass die eine Klasse Teil einer anderen Klasse ist, so spricht man von **Aggregation**. Hierbei besitzt die Verbindungslinie auf der Seite des Ganzen eine offene Raute:



Außerdem gibt es noch eine besonders strenge Form der Aggregation, die man **Komposition** nennt. Hier ist die Beziehung existenziell, d. h. die Teile können nur existieren, wenn das Ganze vorhanden ist. Bei dieser Art der Assoziation wird eine gefüllte Raute auf der Seite des Ganzen verwendet.



An dieser Stelle sei auch noch einmal auf die Begriffe **Vererbung und Polymorphismus** als wichtige Bausteine der objektorientierten Sichtweise verwiesen:

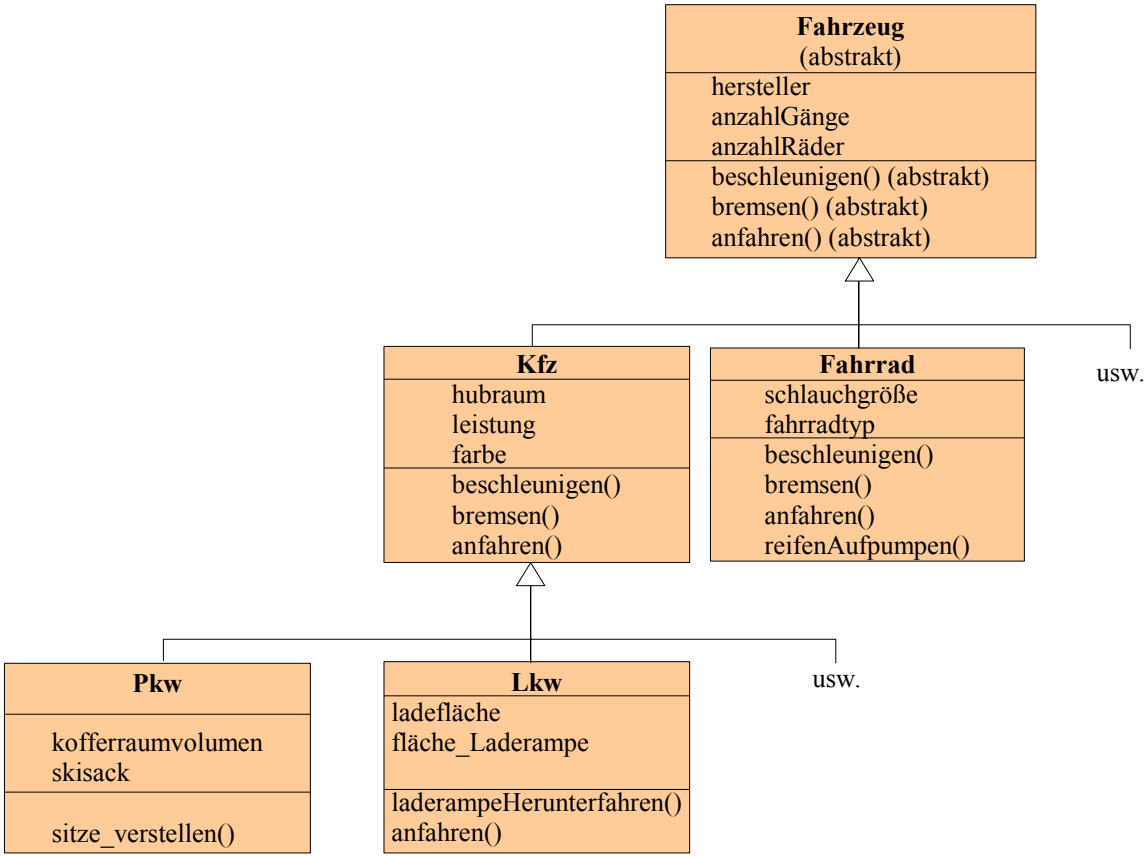
Übernimmt eine Klasse B (Unterklasse) die Attribute und Operationen aus einer Klasse A (Oberklasse), so nennt man dies Vererbung. Bei der Mehrfachvererbung erbt eine Unterklasse aus mehreren Oberklassen. Bei der Unterklasse können neue Attribute und Operationen hinzugefügt oder überschrieben werden, d.h. die Unterklasse wird spezialisiert.

Der Vererbungspfeil zeigt von der Unterklasse zur Oberklasse, in den Unterklassen werden neue Attribute und neue bzw. überschriebene Operationen notiert.

Abstrakte Klasse: Von einer abstrakten Klasse können keine Instanzen gebildet werden. Die Definition einer solchen Klasse dient lediglich zur Festlegung gemeinsamer Attribute und Operationen der Unterklassen. Eine abstrakte Methode wird in der Oberklasse nur definiert und erst in den Unterklassen mit Anweisungen versehen.

Der **Polymorphismus** macht es möglich, dass verschiedene Unterklassen dieselbe Botschaft verstehen, obwohl die technische Umsetzung als Reaktion auf diese Botschaft völlig unterschiedlich sein kann. Auf die Botschaft 'anfahren' können sowohl Objekte vom Typ 'Pkw' als auch vom Typ 'Lokomotive' reagieren, und das Resultat ist bei beiden vergleichbar: sie setzen sich in Bewegung. Allerdings sind die Handgriffe, die hierzu erforderlich sind, bei beiden Objekten sehr unterschiedlich. Wenn Objekte zu verschiedenen Klassen gehören, werden sie auf die gleiche Nachricht unterschiedlich reagieren.

Beispiel: Vererbung aus der abstrakten Klasse 'Fahrzeug'

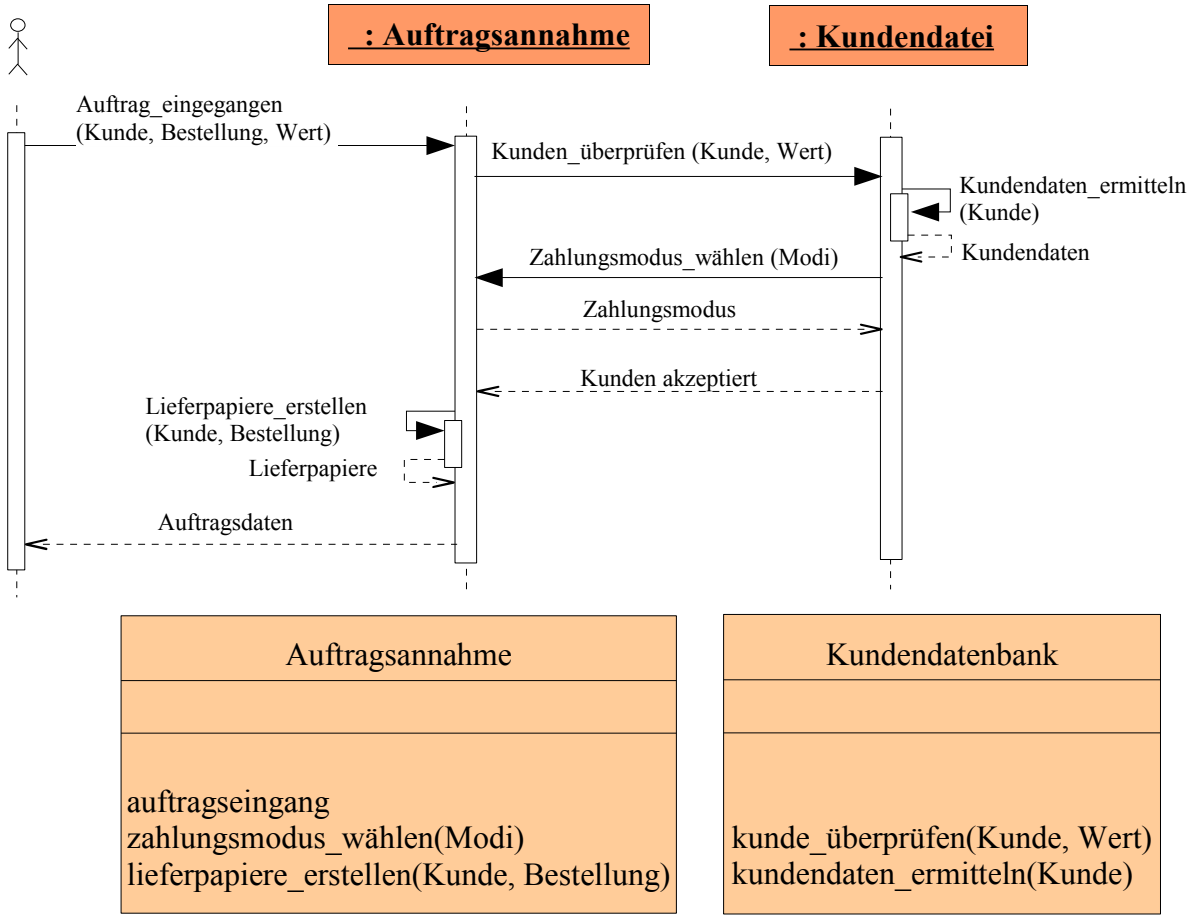


Interaktionsbasierte Modellierung

Um das dynamische Verhalten eines Systems zu beschreiben, wird häufig das Sequenzdiagramme verwendet. Ein **Sequenzdiagramm** zeigt die Interaktionen zwischen ausgewählten Objekten in einem abgegrenzten Anwendungsfall unter Betonung der Abfolge der Nachrichten, die dabei zwischen den Objekten ausgetauscht werden. Da es zeigt, wie Objekte im zeitlichen Ablauf zusammenarbeiten, stellen Sequenzdiagramme eine logistische Erweiterung von Objekt- bzw. Klassendiagrammen dar. Im Beispiel unten sind aus diesem Grund auch die beteiligten Klassendiagramme mit den verwendeten Methoden dargestellt.

Ein Sequenzdiagramm hat zwei Dimensionen: Die vertikale Dimension repräsentiert die Zeit und die horizontale Dimension die Objekte sowie den Benutzer. Die Zeitlinie verläuft senkrecht von oben nach unten, die Objekte werden durch senkrechte Lebenslinien beschrieben und die gesendeten Nachrichten waagerecht entsprechend ihres zeitlichen Auftretens eingetragen. Ein Objekt-Bezeichner muss also mindestens aus einem Doppelpunkt gefolgt von einem Klassennamen bestehen, um das Objekt eindeutig als Instanz dieser Klasse zu kennzeichnen. Optional ist die Vergabe eines Variablennamens für diese Instanz. Diese Möglichkeit ist insbesondere nützlich, wenn mehrere Instanzen derselben Klasse in einem Sequenzdiagramm verwendet werden.

Als Beispiel soll die Auftragsannahme bei der Firma Mustermann dienen. Folgende Ereignisse finden statt: *Auftragseingang – Prüfen ob Kunde bereits vorhanden – Kundendaten ermitteln – Zahlungsmodus festlegen – Lieferpapiere erstellen*.



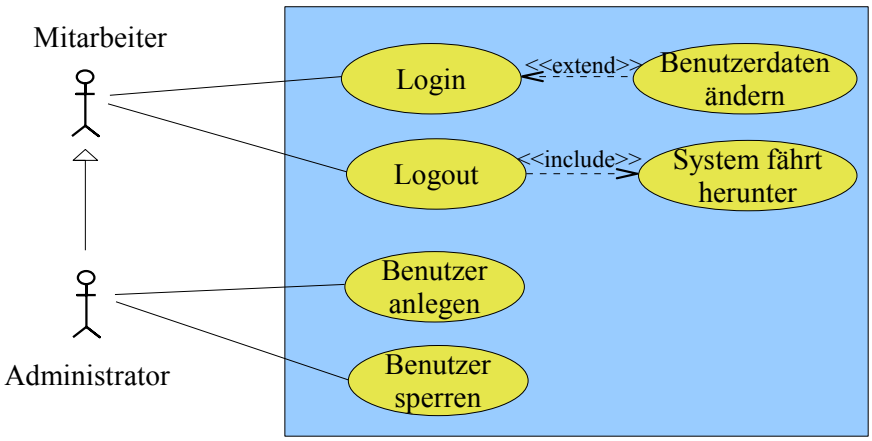
Die Dauer der Existenz eines Objekts wird im Sequenzdiagramm als senkrechte gestrichelte Linie unterhalb des Objekt-Symbols dargestellt. Die auf der Lebenslinie eines Objektes platzierten Rechtecke kennzeichnen Perioden, in denen dieses Objekt aufgrund einer empfangenen Nachricht eine Aktion ausführt - entweder direkt oder durch eine an ein anderes Objekt verschickte Nachricht. Es kann für jedes Objekt in einem Sequenzdiagramm mehrere solcher Aktivitätsperioden geben. Der Benutzer als Strichmännchen links oben im Diagramm repräsentiert. Er stößt die Abarbeitung der Aktionen des Anwendungsfalls durch eine Nachricht an das direkt rechts von ihm stehende Objekt an (in der Abbildung die Instanz der Klasse 'Auftragsannahme'), das wiederum die gewünschten Aktionen entweder selbst ausführt oder per Nachricht an andere Objekte veranlasst. Der Benutzer

interagiert nur mit diesem Objekt, die restlichen Objekte und Nachrichten bleiben vor ihm verborgen. Der Austausch von Nachrichten (Methoden) zwischen Objekten bedeutet im Grunde nur, dass ein Objekt eine Operation eines anderen Objektes aufruft. Dieser **Vorgang** wird als **durchgezogener waagerechter Pfeil mit ausgefüllter Spitze** von der Lebenslinie des sendenden Objekts zu der des empfangenden Objekts gezeichnet, wobei Objekte auch Nachrichten an sich selbst senden können, wie etwa 'Lieferpapiere_erstellen' in obiger Abbildung. Nachdem die durch die Nachricht veranlasste Aktion ausgeführt wurde, kennzeichnet ein **gestrichelter Rückpfeil mit offener Spitze** die **Rückkehr** zum aufrufenden Objekt. Bei der Rückkehr kann ggf. ein Wert an den Urheber der Nachricht zurückgegeben werden, so etwa 'Zahlungsmodus' im Beispiel. Schließlich besteht noch die Möglichkeit, Kommentare in Sequenzdiagrammen zu platzieren, die die Modellierung näher erläutern. Diese Kommentare können frei in natürlicher Sprache formuliert werden und beliebige Länge haben.

Mit **Anwendungsfall-Diagrammen** kann das externe Systemverhalten aus Anwendersicht modelliert werden. Anwendungsfälle (Use Cases) werden als Ellipsen in einem Kasten, dem System, dargestellt und mit dem dazugehörigen Akteur (Strichmännchen), der dieses Anwendungsfall auslöst, verbunden. Man verwendet Use Cases, um das geplante Verhalten eines Systems darzustellen, ohne näher darauf eingehen zu müssen, wie das System intern mit den einzelnen Anforderungen umgeht. Ein Anwendungsfall führt normalerweise eine Reihe von Arbeiten aus, wie zum Beispiel das Berechnen von Werten, das Erzeugen von Objekten oder das Verändern von Zuständen in Objekten. Es kann auch sein, dass ein Akteur ein eigenes System darstellt. Zu jedem Anwendungsfall gibt es eine Beschreibung in Textform. Die Beschreibung kann ausführlich oder stichpunktartig erfolgen.

Ein **Anwendungsfall** ist eine abgeschlossene, zusammenhängende Einheit (nicht zu klein und nicht zu groß) und sollte eine logische, zusammengehörige, wiederkehrende Anwendung innerhalb eines Systems darstellen.

Beispiel: Im einem System gibt es 2 verschiedene User (Mitarbeiter und Administrator). Beide können sich im System ein- und ausloggen. Nach dem Einloggen können beide ihre persönlichen Benutzerdaten ändern. Nach dem Ausloggen wird das System heruntergefahren. Der Administrator hat außerdem noch die Möglichkeit, Benutzer zu sperren und neue Benutzer anzulegen.



Das Rechteck stellt das geplante System dar. Es ist quasi der Container für die Anwendungsfälle. Eine Linie stellt eine **Assoziation** zwischen einem Akteur und einem Anwendungsfall dar. Sie beschreibt den Zugriff des Akteurs auf die Funktionalität, die das System in diesem Fall zur Verfügung stellt bzw. eine Antwort des Systems an einen Akteur.

Zwischen zwei Anwendungsfällen kann es auch Beziehungen geben:

Die ***include***-Beziehung definiert einen Anwendungsfall, der die Funktionalität, die ein anderer Use Case zur Verfügung stellt, importiert. Der Anwendungsfall 1 importiert die Funktionalität des Anwendungsfalls 2. Sie wird gestrichelt dargestellt und mit <<include>> gekennzeichnet.

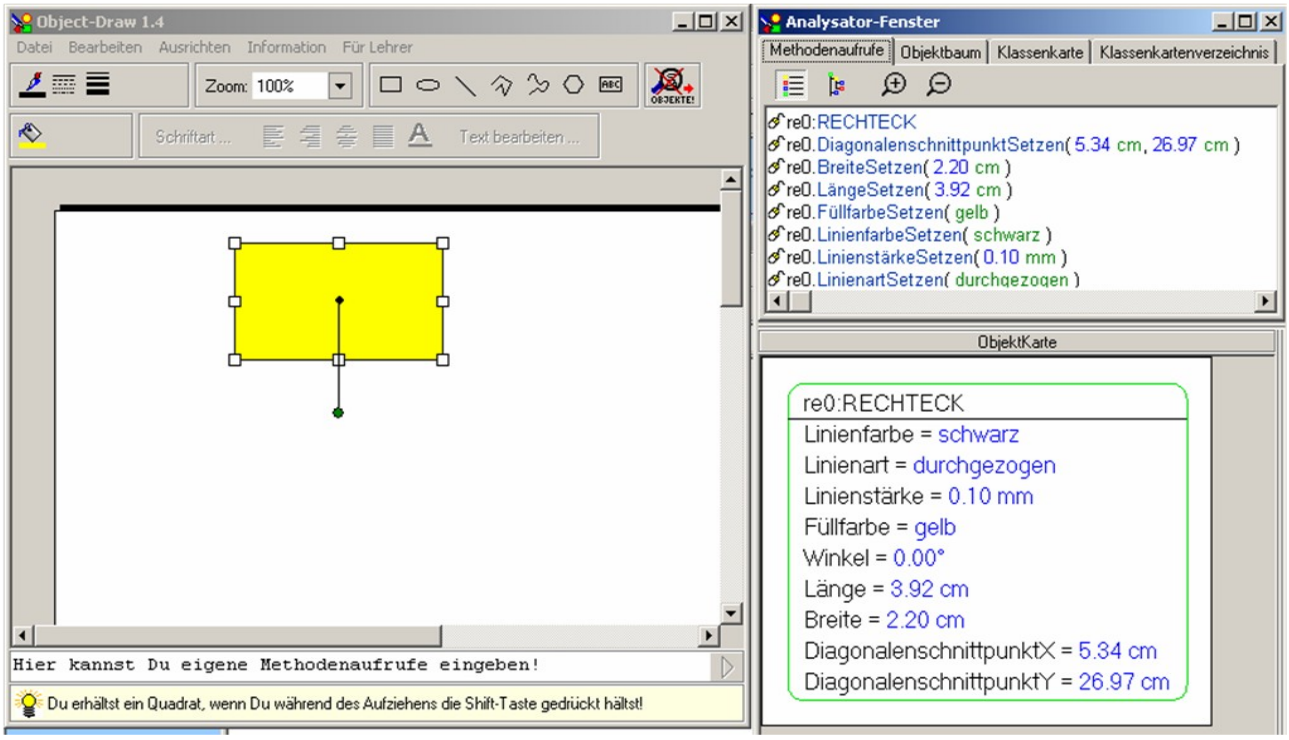
Die ***extend***-Beziehung bezeichnet einen Anwendungsfall, dessen Verhalten durch einen weiteren Anwendungsfall erweitert werden kann. Der Anwendungsfall 1 erweitert den Anwendungsfall 2. Die Beziehung zwischen den Fällen zeigt von dem erweiternden zu dem erweiterten. Sie wird ebenfalls gestrichelt dargestellt und mit <<extend>> gekennzeichnet.

In Anwendungsfall-Diagrammen ist die **Vererbungsbeziehung** erlaubt. Sie kann verwendet werden um Verhalten zwischen Use Cases zu vererben. Sie kann aber auch (wie in obigem Beispiel) verwendet werden, um Vererbungsbeziehungen zwischen Akteuren aufzubauen.

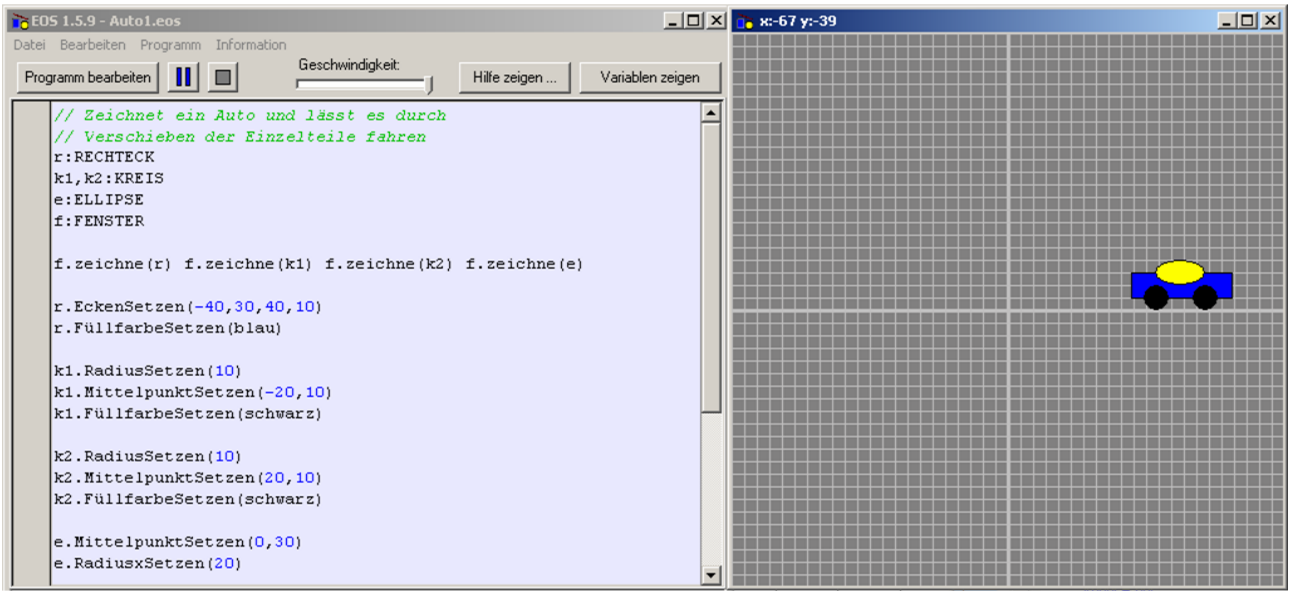
Programme zur Vermittlung objektorientierter Grundbegriffe

Die Programmierwerkzeuge ObjectDraw und EOS sind von M. Papst für den unterrichtlichen Einsatz in der Sekundarstufe I entwickelt worden und stehen kostenlos zum Download zur Verfügung. Beide Programme ermöglichen es, Klassen- und Objektdiagramme zu interpretieren und mithilfe von Methoden Attributwerte zu verändern. Dabei wird besonders Wert auf die Punktnotation gelegt.

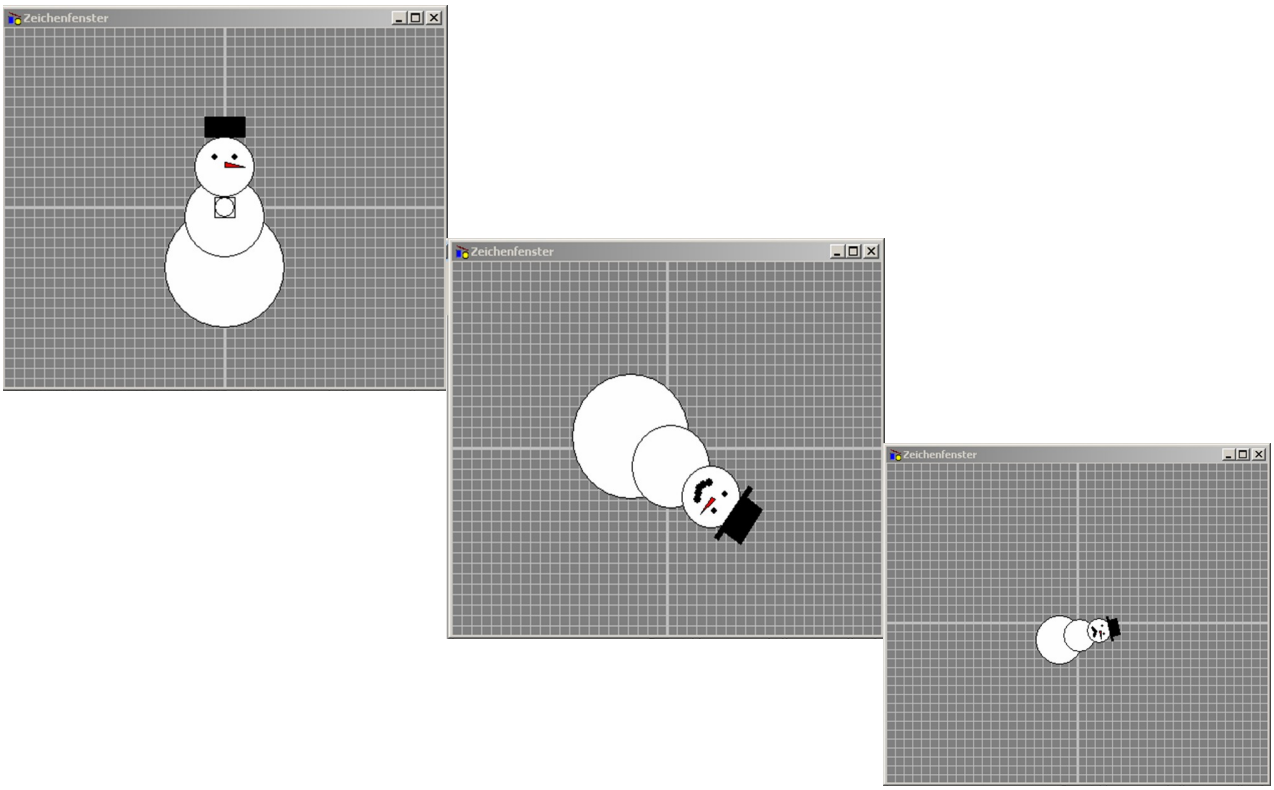
Beispiel ObjectDraw:



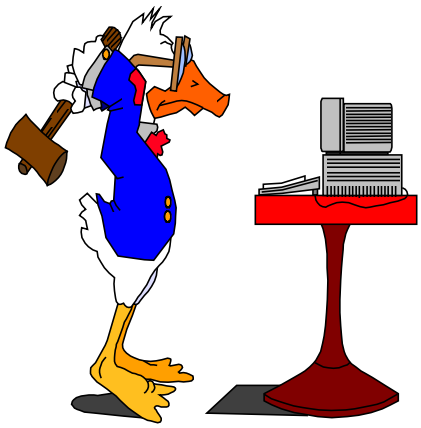
Beispiele EOS:



Die Einführung objektorientierter Grundbegriffe sollte anhand einfacher Vektorgrafiken geschehen. (siehe auch Seite 34!). Herr Prof. Peter Hubwieser von der TU München empfiehlt, EOS erst nach der Einführung algorithmischer Kontrollstrukturen (Bedingte Anweisung, Wiederholung) einzusetzen (siehe nachfolgendes Kapitel). Dann kann EOS zur Vertiefung der Algorithmik, z. B. zur Animation von Grafiken verwendet werden.



Algorithmische Modellierung



Unter einem Algorithmus versteht man eine Verarbeitungsvorschrift, die so präzise formuliert ist, dass sie von einem mechanisch oder elektronisch arbeitenden Gerät durchgeführt werden kann. Der Begriff geht auf den persischen Mathematiker Al-Chowarizmi (ca. 780 - 840 n. Chr.) zurück, der ein bedeutendes Werk über Algebra verfasste.

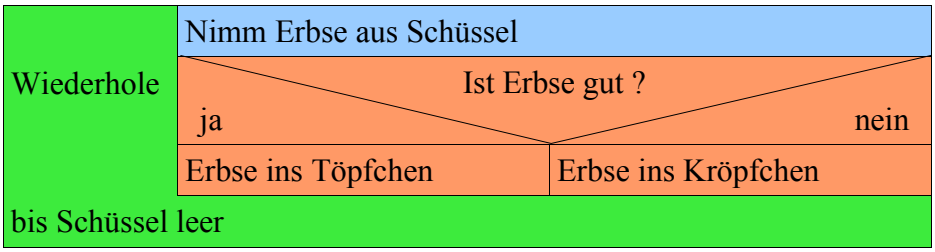
Eigenschaften von Algorithmen

- Im allgemeinen löst ein Algorithmus eine **Klasse von Problemen**. Die Auswahl eines Problems erfolgt über Parameter.
- Algorithmen sind **determiniert**, d. h. wird ein Algorithmus mit den gleichen Eingangswerten wiederholt, so liefert er auch stets das gleiche Ergebnis.
- Die Beschreibung eines Algorithmus besitzt eine **endliche Länge**.
- Für die Praxis sind meist nur solche Algorithmen von Bedeutung, die nach **endlich vielen Schritten ein Ergebnis** liefern.
- Ein Algorithmus heißt **deterministisch**, wenn zu jedem Zeitpunkt seiner Ausführung höchstens eine Möglichkeit der Fortsetzung besteht.

Algorithmische Modellierung lässt sich vorzugsweise mit Nassi-Shneiderman-Struktogrammen und Programmablaufplänen (Flussdiagramme) umsetzen.

Bevor wir zu informatischen Anwendungen kommen, hier ein märchenhaftes Beispiel, das jedoch schon alle Strukturelemente enthält, die für eine Programmierung notwendig sind:

'Die guten ins Töpfchen, die schlechten ins Kröpfchen'



Im nächsten Beispiel soll mit Hilfe des Euklidschen Algorithmus der größten gemeinsame Teiler (ggT) von zwei Zahlen berechnet werden. Nach Euklid wird von den beiden Zahlen die größere durch die kleinere dividiert. Geht diese Division ohne Rest auf, so ist der Divisor der ggT. Bleibt dagegen ein Rest, so wird nun der vorherige Divisor zum Dividenden, der jetzt durch den entstandenen Rest geteilt wird. Dies wird so lange wiederholt, bis die Division ohne Rest aufgeht.

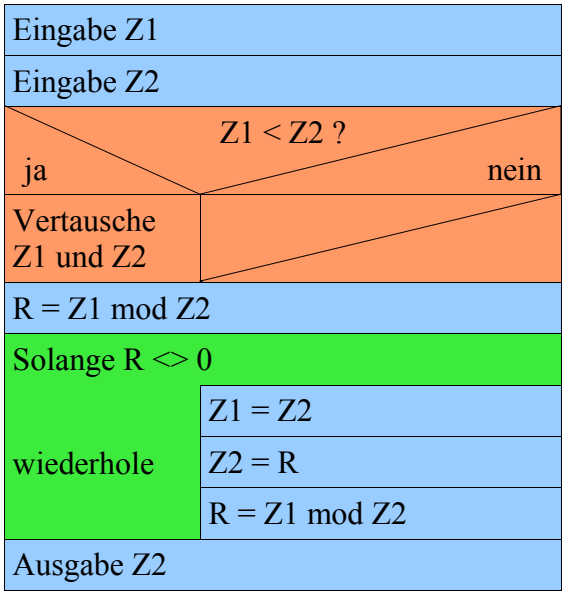
Der letzte Rest ≠ 0 ist der größte gemeinsame Teiler.

Zum Verständnis des Algorithmus sollte man zunächst einen Schreibtischtest durchführen (Abb. rechts).

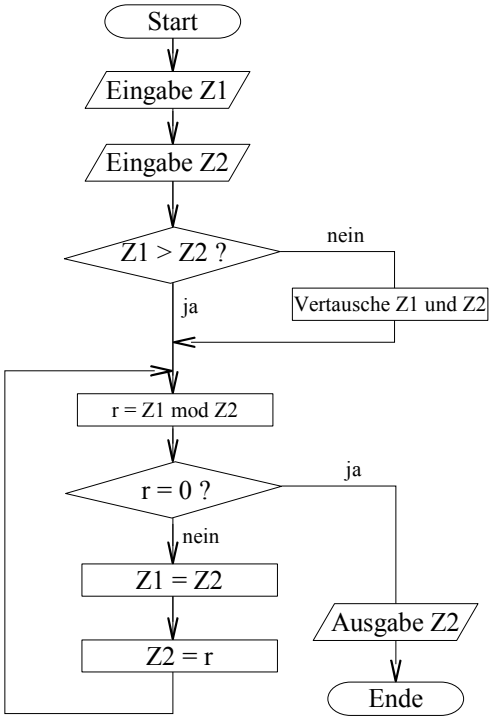
```
210 : 154 = 1 Rest 56
154 :  56 = 2 Rest 42
 56 :  42 = 1 Rest 14
 42 :  14 = 3 Rest 0

==> ggT(154, 210) = 14
```

Struktogramm

Euklidscher Algorithmus



Programmablaufplan

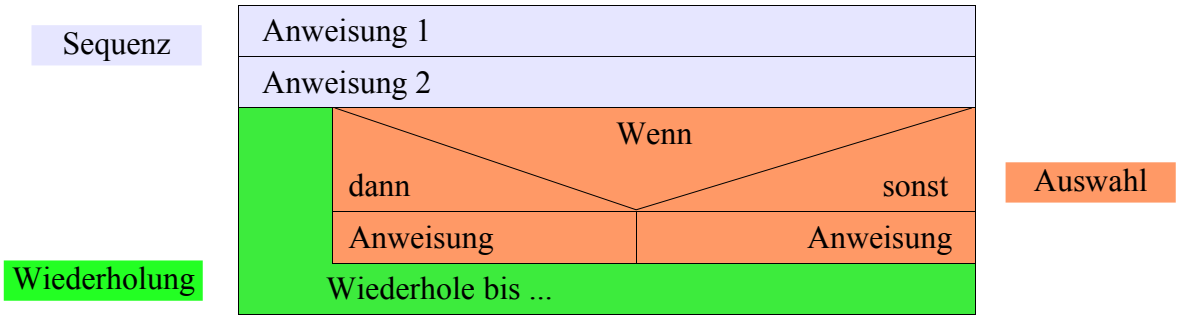


Für die Strukturierung solcher Aufgabenstellungen gibt es im Wesentlichen drei Strukturelemente:

- Sequenz
- Auswahlstrukturen
- Wiederholungsstrukturen

Programmablaufpläne verwenden dabei folgende Elemente:

Im Struktogramm sehen die Strukturelemente Sequenz, Auswahl und Wiederholung so aus:



Hinweis: Struktogramm und Programmablaufplan sind nicht Bestandteile von UML, da sie jedoch den Datenfluss zeigen, können sie als funktionales Modell aufgefasst werden.

Systematisches Lösen von Problemen

Da auf die algorithmische Sichtweise in der Informatik der Realschule nicht verzichtet werden kann, hier zunächst die Schritte des Problemlösens aus dieser Sichtweise. Zur Lösung von Aufgabenstellungen mittels des Computers empfiehlt es sich, **strukturiert und systematisch** vorzugehen; d.h. das Lösen des Problems in einzelne Arbeitsschritte zu zerlegen:

- **Problemanalyse**
- **Erstellung eines Lösungsentwurfs**
- **Übertragung auf den Rechner**
- **Erprobung**
- **Erstellung einer Dokumentation**
- **Programmwartung**

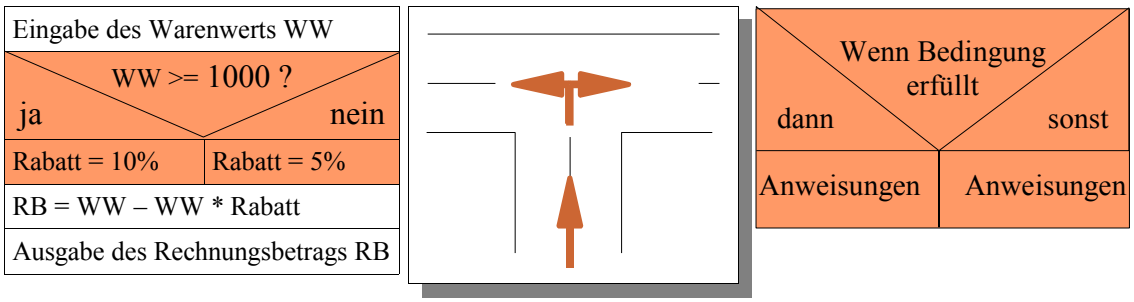
Bei der **Problemanalyse** prüft man, ob die Aufgabe in mehrere Teilaufgaben zerlegt werden kann. Zu jedem Teilproblem wird dann ein geeigneter Algorithmus erstellt.
Auch hier findet das **EVA-Prinzip** seine Anwendung; d.h. man legt zunächst die Ein- und Ausgabedaten fest. Ebenfalls zu definieren ist der bei den Konstanten und Variablen verwendete Datentyp. Dadurch wird einerseits die Größe des benötigten Speicherplatzes (siehe oben 'Datentypen') bestimmt, andererseits werden auch die für den Datentyp möglichen **Operatoren (arithmetische: + - * / , logische: AND OR NOT, vergleichende: = < >)** festgelegt.

> Der Begriff Datentyp beschreibt den Wertebereich von Daten, in dem es ganz bestimmte Operationen gibt, die man auf alle Daten diesen Typs anwenden kann.

Im Wesentlichen unterscheidet man nummerische Daten wie ganze Zahlen (integer/longinteger) und Fließkommazahlen (single/double), Zeichen- und Textdaten (character/string), Wahrheitswerte (boolean) und Datums- bzw. Zeitwerte (date/time).

Hier eine Reihe von Beispielaufgaben mit algorithmischer Modellierung im Struktogramm. Die Beispiele können mit den unterschiedlichsten Werkzeugen umgesetzt werden:

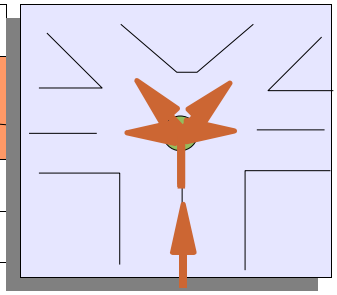
- **Auswahlstrukturen**
  - Ein Kaufhaus gewährt ab einem Warenwert von 1000 € einen Rabatt von 10%, liegt der Warenwert darunter, so erhält der Kunde nur 5%.



  Da man hier die Auswahl aus zwei Alternativen hat, spricht man von **zweiseitiger Auswahl**.
  - Existieren mehr als zwei Wahlmöglichkeiten, so liegt eine **mehrseitige Auswahl** vor:

Nach Eingabe der bei einer Stegreifaufgabe erreichten Punktzahl soll die zugehörige Note ausgegeben werden (Notenskala: 14-13, 12-11, 10-9, 8-7, 6-5, 4-0).

Eingabe der Punktzahl					
Wenn Punktzahl =					
0 – 4	5 – 6	7 – 8	9 – 10	11 – 12	13 – 14
Note = 6	Note = 5	Note = 4	Note = 3	Note = 2	Note = 1
Ausgabe der Note					



- Es gibt auch die Möglichkeit, mehrere Blöcke einer zweiseitigen oder mehrseitigen Auswahl ineinander zu schachteln. Man spricht bei dieser Programmstruktur dann von einer **mehrstufigen oder geschachtelten Auswahl:**
  Ein Elektronikversand verlangt für eine integrierte Schaltung folgende Preise: 1 Stück 0,60 €, ab 10 Stück 0,55 €, ab 25 Stück 0,50 €, ab 100 Stück 0,45 €. Aus der eingegebenen Stückzahl soll der Gesamtpreis berechnet werden.

<table>
<tr><td colspan="4">Eingabe der Stückzahl SZ</td></tr>
<tr><td colspan="4">Wenn SZ < 10 ?
dann sonst</td></tr>
<tr><td rowspan="3">G = SZ * 0,60</td><td colspan="3">Wenn SZ < 25 ?
dann sonst</td></tr>
<tr><td rowspan="2">G = SZ * 0,55</td><td colspan="2">Wenn SZ < 100 ?
dann sonst</td></tr>
<tr><td>G = SZ * 0,50</td><td>G = SZ * 0,45</td></tr>
<tr><td colspan="4">Ausgabe des Gesamtpreises G</td></tr>
</table>

Selbstverständlich kann man diese Aufgabenstellung auch mit Hilfe der deutlich übersichtlicheren mehrseitigen Auswahl lösen.

- Zur **Wiederholungsstruktur** folgende Besispiele:
  Ein bestimmtes Kapital soll verzinst werden, wobei der Zins dem jeweiligen Kapital zugeschlagen wird (Zinseszins). Es soll die Kapitalentwicklung angezeigt werden. Neben dem Struktogramm werden bei diesen Beispielen noch Klassendiagramme angegeben.
  Dabei sind folgende Aufgabenstellungen möglich:
  - Das Kapital soll bei einer festen Laufzeit verzinst werden.

<table>
<tr><td colspan="2">Eingabe des Kapitals</td></tr>
<tr><td colspan="2">Eingabe des Zinssatzes</td></tr>
<tr><td colspan="2">Eingabe der Laufzeit</td></tr>
<tr><td colspan="2">Für Jahr von 1 bis 6</td></tr>
<tr><td rowspan="3">wieder-
hole</td><td>Zinsen = Kapital * Zinssatz / 100</td></tr>
<tr><td>Kapital = Kapital + Zinsen</td></tr>
<tr><td>Ausgabe von Jahr, Zinsen, Kapital</td></tr>
</table>

  - Das Kapital soll so solange angelegt werden bis es mindestens eine Million beträgt.

Eingabe des Anfangskapitals	
Eingabe des Zinssatzes	
Endkapital = 1 000 000	
Solange Kapital < Endkapital	
wieder-hole	Zinsen = Kapital * Zinssatz / 100
	Kapital = Kapital + Zinsen
	Jahr = Jahr + 1
Ausgabe von Jahr	

- Das Kapital soll sich bei Auszahlung mindestens verdoppelt haben.

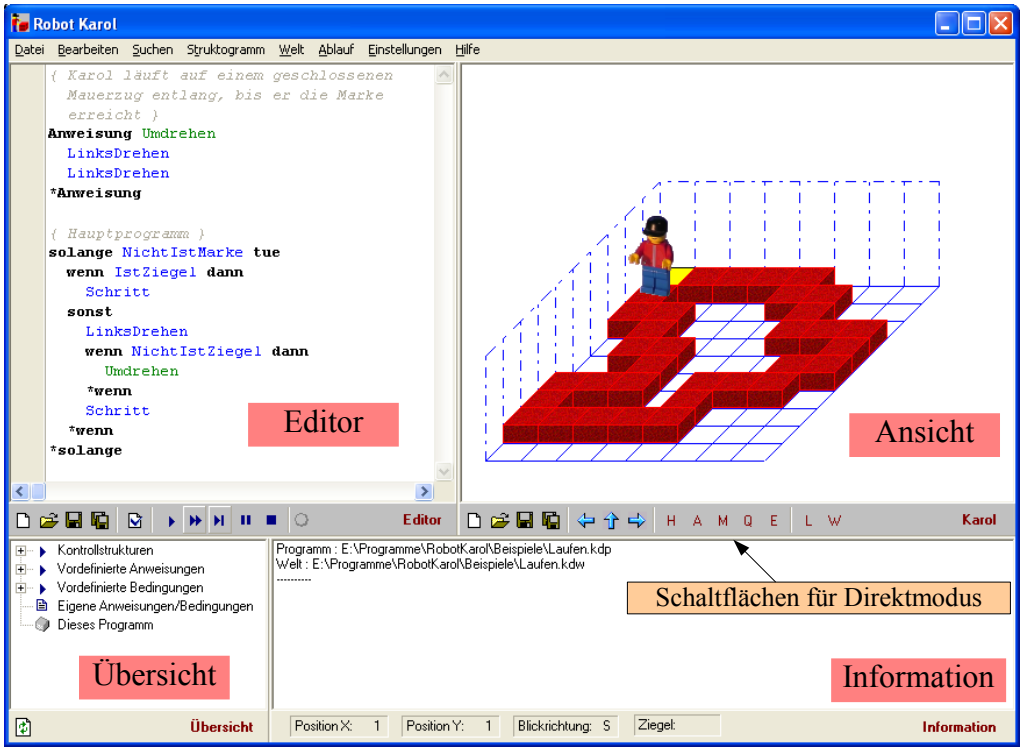
Eingabe des Anfangskapitals	
Eingabe des Zinssatzes	
Kapital = Anfangskapital	
Jahr = 0	
wieder-hole	Zinsen = Kapital * Zinssatz / 100
	Kapital = Kapital + Zinsen
	Jahr = Jahr + 1
bis Kapital = 2 * Anfangskapital	
Ausgabe von Jahr	

- Hier noch einige zusammenfassende **Aufgaben zum Üben**:
  - Ein Student hat sich folgende, täglich wiederkehrende Prozedur zu eigen gemacht: Abends, bevor er zu Bett geht, durchsucht er seine Taschen nach Kleingeld. Wird er fündig, so kommt alles in ein Sparschwein. Da er fünf Sparschweine besitzt, geht er nach folgendem Schema vor: er stellt den Betrag seines Kleingeldes in Euro-Cent fest und zählt die Sparschweine der Reihe nach ab. Dabei kehrt er am Ende der Sparschweinreihe die Zählrichtung um, ohne das letzte Sparschwein doppelt zu zählen. Hat er den Betrag erreicht, gibt er das gesamte Geld genau in dieses Schwein. Im Laufe der Zeit wurde ihm das Abzählen der Sparschweine zu mühsam und er beschließt, ein Computerprogramm zu schreiben, das das richtige Sparschwein ermittelte.
  - Ein Programm soll das Würfeln mit einem normalen sechsseitigen Würfel simulieren und die Ergebnisse anzeigen. Dabei soll solange gewürfelt werden bis eine 6 erscheint.
  - In einem Parkhaus kostet jede angefangen Stunde Parkzeit 0,50 €. Der Geldautomat nimmt auch Papiergeld entgegen, gibt aber nur Münzen zurück. Berechne das Rückgeld, wenn mit einem Schein bezahlt wird. Dabei soll auch angegeben werden, wie viele von jeder Münzsorte (2 €, 1 €, 50 Cent) ausgeworfen werden.
  - Zwei Schüler spielen folgendes kleine Spiel: der eine denkt sich eine zweistellige Zahl aus, der andere muss so lange raten bis er die gedachte Zahl errät. Dabei erhält er bei jedem Rateversuch einen Hinweis darüber, ob die geratene Zahl stimmt, zu klein oder zu groß ist.

Roboter Karol

Ein ganz anders geartetes Beispiel gestattet die Umsetzung des Lehrplans in Bezug sowohl auf Modellierung als auch 'strukturierte Programmierung'. Ohne die Tücken einer umfassenden Programmiersprache kann man auf relative einfache Art und Weise strukturierte Programmierung und Modellierung unter Einbeziehung des Werkzeugs Computer verwirklichen: der Roboter Karol. Aus didaktischen Gründen eignet sich der Roboter Karol sehr gut zur Einführung in die algorithmische Denkweise. Seine Attribute und Fähigkeiten sind gut überschaubar und eine vorgegebene Aufgabenstellung kann zunächst im Direktmodus gelöst werden. So gelingt es mit Schülern sehr rasch und gut, die algorithmischen Strukturen zu erschließen.

Die Programmoberfläche ist in vier Bereiche unterteilt.



Karol ist ein Objekt der Klasse **ROBOTER**. Seine 'Welt' ist ein rechteckiges Feld aus Planquadraten. Er kann sich von Planquadrat zu Planquadrat bewegen und mit Ziegeln bauen. Das folgende Klassendiagramm zeigt einige Attribute und Methoden der beiden Klassen.

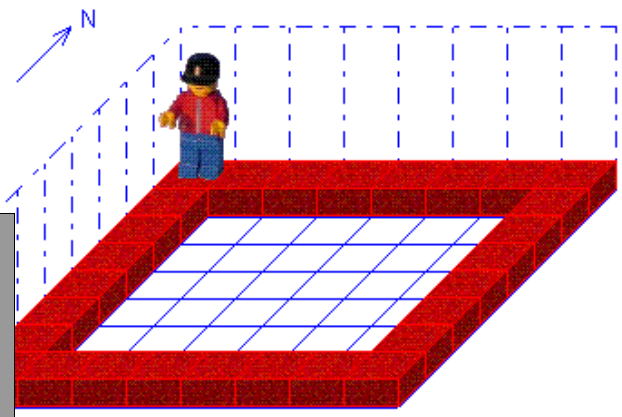
Roboter	*Klasse*	*Welt*
x-Position y-Position blickrichtung ziegelImRucksack	Attribute ⟷	x-Größe y-Größe z-Größe 3D-Modus
schritt() linksDrehen(90°) rechtsDrehen(90°) hinlegen() aufheben()	Methoden ⟷	kopieren() speichern() zurücksetzen() markeSetzen() markeEntfernen()

Im Unterricht sollte man zunächst mit einigen einfachen Aufgaben im Direktmodus beginnen, z.B. Karol soll 3 Schritte machen und anschließend einen Ziegelstein hinlegen. Dasselbe kann man nun in den Editor eingeben und so die sequentielle Struktur und den Programmmodus einführen.

Es folgen Beispielsaufgaben für Auswahl und Wiederholung. Lässt man diese Aufgaben zunächst ebenfalls wieder im Direktmodus lösen, wird man feststellen, dass die Schüler sehr schnell den Pseudocode für das zugehörige Programm herausfinden. Der Schritt zum Code von Karol ist minimal, da man die Sprache von Karol im Prinzip dem Pseudocode entspricht.

Es empfiehlt sich bei Karol zunächst die Wiederholung und dann die Auswahl einzuführen.

Karol befindet sich beim Start auf Position (X1/Y1) mit Blickrichtung Süden. Seine Welt hat die Größe 8 x 8. Er soll entlang dem Rand seiner Welt Ziegeln auslegen.
In der Programmiersprache von Karol sieht die Lösung wie in den Abbildungen aus.



wiederhole 4 mal
wiederhole 7 mal
hinlegen
Schritt
linksdrehen

```
wiederhole 4 mal
   wiederhole 7 mal
      hinlegen
      Schritt
   *wiederhole
   linksdrehen
*wiederhole
```

Die Programmiersprache von Karol umfasst eine gut überschaubare Anzahl von Befehlen und ist ins Deutsche übertragen worden. Auch das ist ein Vorteil dieses Werkzeugs, den man nicht übersehen sollte.

wiederhole 2 mal
Solange NichtIstWand
Schritt
linksdrehen
wiederhole 4 mal
Solange NichtIstWand
hinlegen
Schritt
linksdrehen

Die Aufgabe kann nun so angewandelt werden, dass der Algorithmus für eine beliebige Welt gelten soll.

wiederhole 4 mal
solange NichtIstWand
hinlegen
Schritt
linksdrehen

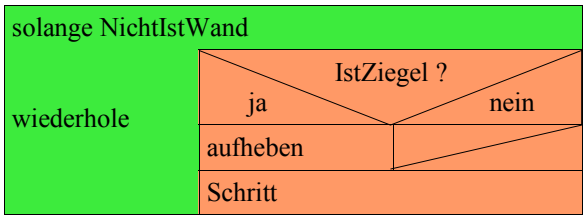
Im weiteren kann man nun die Bedingung hinzufügen, dass die Ausgangsposition und die Blickrichtung von Karol beliebig sein sollen.

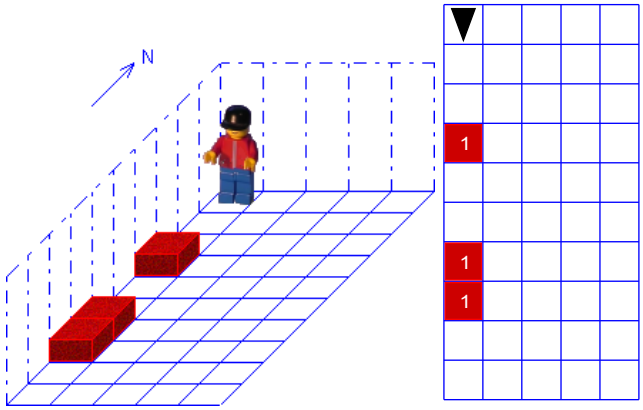
Das Einstiegsbeispiel für die Auswahl wäre folgende Aufgabenstellung denkbar:

Karol befindet sich wieder in seiner normalen Ausgangsstellung (X1/Y1) mit Blickrichtung Süden. Auf seinem geraden Weg zur gegenüberliegenden Wand befinden sich einige wenige Ziegelsteine, die er einsammeln soll.

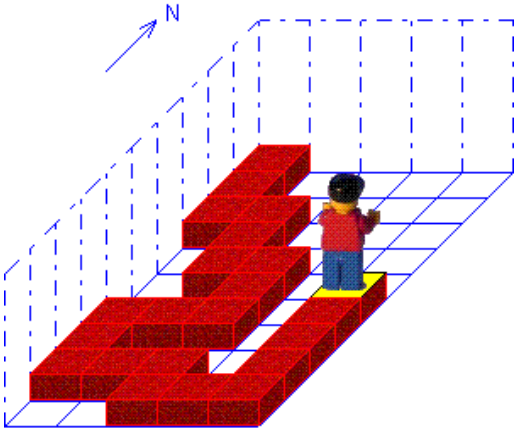
Vor Beginn des Programmlaufs müssen die Ziegelsteine platziert werden. Dazu muss man in den ***2D-Modus*** umschalten und die ***Welt direkt festlegen***. Dies erfolgt im Menüpunkt 'Welt'.

Karol muss zunächst entscheiden, ob vor ihm ein Ziegelstein liegt oder nicht. Das Struktogramm kann so aussehen:

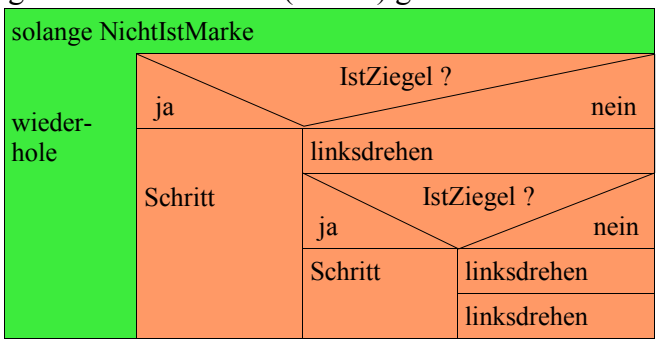




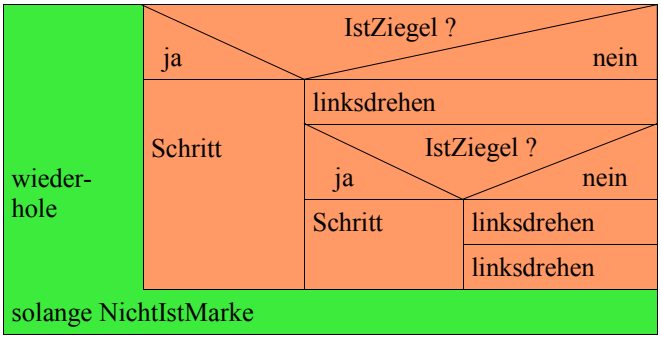
Auch hier sind Ergänzungen der Aufgabenstellung gut möglich: Dieselbe Aufgabenstellung wie eben, jedoch können Ziegeln entlang dem Rand seiner Welt liegen...

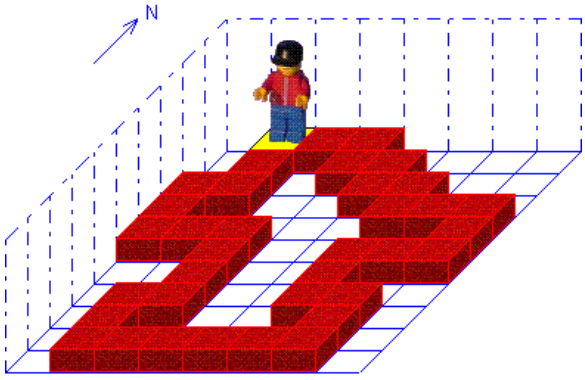


Oder: Karol soll auf einem durch Ziegelsteinen markierten Weg bis zu dessen Ende (Marke) gehen.



Baut man einen 'Rundkurs', bei dem die Marke Start und Ziel symbolisiert, muss man die Wiederholung mit Endbedingung wählen, da Karol sonst gar nicht erst losläuft.





Siehe hierzu auch das leicht veränderte Beispiel weiter oben, das in der Abbildung 'Programmoberfläche' dargestellt ist.

Das zugehörige Programm findet man im Internet unter: www.schule.bayern.de/karol/

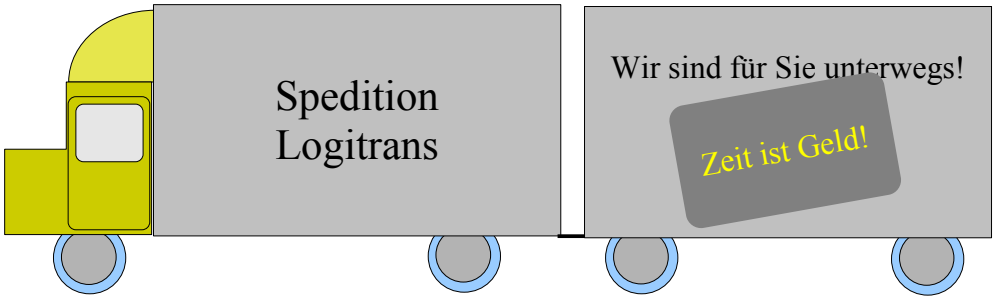
Ein Handbuch und eine Reihe von Beispieldateien werden ebenfalls mitgeliefert.

Umgang mit digitalen Daten

Die Objektstruktur bei Grafiksystemen

Grafiken sind sehr gut zur objektorientierten Analyse geeignet. Zunächst werden die **Objekte** identifiziert. Für jedes dieser Objekte sind bestimmte **Eigenschaften (Attribute)** zu nennen, denen wir aktuelle **Attributwerte** zuordnen können. Daneben besitzen die Objekte gewisse Fähigkeiten, die **Methoden**, mit denen sie mit ihrer Umgebung kommunizieren. Objekte mit gleichen Attributen und Methoden werden zu einer **Klasse** zusammengefasst.

Wir wollen eine einfache Zeichnung erstellen wie z.B. den abgebildeten LKW. Dazu verwenden wir ein vektororientiertes Grafikprogramm. Wir erkennen, dass die Zeichnung aus einer Reihe von **Objekten (Rechteck1, Polygon1, Kreis1, Kreisteil1, Linie1, Linie2, Textobjekt1, etc.)** besteht.



Bei einer Vektorgrafik können die Objekte beliebig auf dem Zeichenblatt verschoben werden. Es sind voneinander unabhängige Objekte. Einige sind vom gleichen Typ oder von derselben Klasse: Rechteck oder Kreis.

Die Objekte sind die **Instanzen** aus verschiedenen **Klassen**. **Die Klasse stellt sozusagen den Bauplan für die einzelnen Instanzen dar.**

Schreibweise in Punktnotation: **Anhänger[Rechteck].**

Objektname — Klassenbezeichnung

Beispiele für Klassen und Instanzen:

Klasse	*Name der Instanz*
Rechteck	Anhänger
Kreis	Rad
Linie	Anhängerkupplung

Instanzen derselben Klasse haben dieselben Attribute (Eigenschaften), sie unterscheiden sich aber in den Attributwerten (Eigenschaftswerten). Die Instanzen Fahrertür und Anhänger der Klasse Rechteck unterscheiden sich in mehreren Merkmalen: in Größe, Farbe und der Eckenrundung.

Punktnotation: **Anhänger[Rechteck].Füllfarbe = grau**

Rad[Kreis].Linienstil = durchgängig

Weitere Attribute sind z.B. Länge, Breite, Eckenrundung, Text usw.

Instanzen verschiedener Klassen unterscheiden sich in ihren Attributen; z.B. hat ein Rechteck das Attribut Breite, eine Linie besitzt dieses Attribut nicht. Der Zusammenhang zwischen Instanzen und ihren Attributwerten lässt sich auch als Objektdiagramm darstellen:

Attribut	*Attributwert*
Breite	3,80 cm
Füllfarbe	grau
Linienstil	durchgezogen

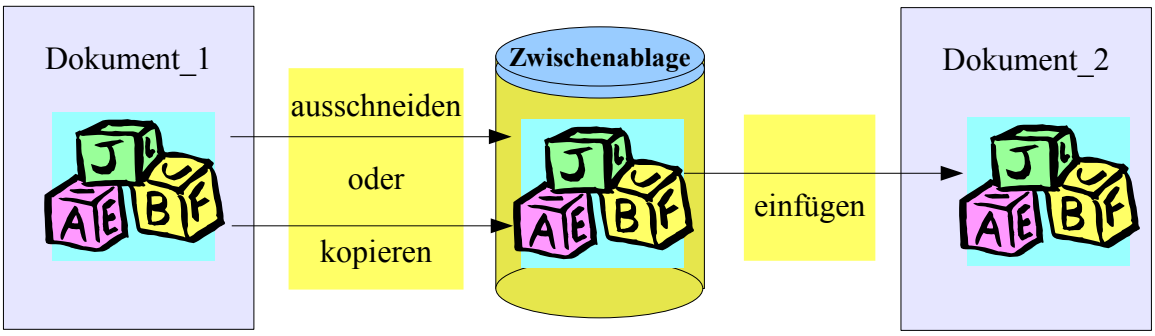
Bei der Arbeit an der Zeichnung beschränkte sich unser Tun nicht allein darauf, den Objekten Attributwerte zuzuordnen. Gleich aussehende Objekte lassen sich auch durch die **Methode** 'Kopieren' erzeugen oder platzierte Objekte durch 'Verschieben' an eine andere Position bringen. Die Methoden werden auch hier durch **Botschaften (Ereignisse)** aufgerufen: z.B. Mausklick, Aufruf von Menübefehlen, Drücken eines Bildsymbols.

Beispiele für Methoden in einem Grafikprogramm:
'sich erschaffen', 'markieren', 'drehen', 'kopieren', 'einfügen', usw.

Schreibweise: **Rad[Kreis].kopieren()**

Die Klammern dienen der Unterscheidung von etwa gleichnamigen Attributen.

Einige Methoden erfordern zusätzliche Hilfsmittel. Beim Ausschneiden oder Kopieren beispielsweise wird ein Behälter benötigt, der die Objekte bis zur Wiederverwendung (Einfügen) aufbewahrt: die Zwischenablage. Sie ist in der Regel nicht sichtbar. Dabei können die beiden betroffenen Dokumente zu verschiedenen Anwenderprogrammen gehören, die Zwischenablage ist ein Windowsbestandteil, also Teil des Betriebssystems.



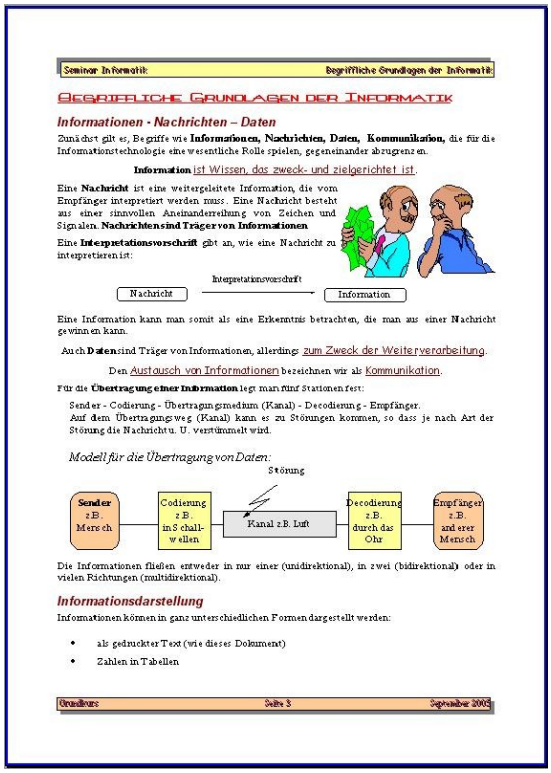
Notation: **Würfel[Grafik].kopieren()** **Würfel[Grafik].einfügen()**

Was passiert dabei im einzelnen?

Methode	*Wirkung*	*Aufruf*
Ausschneiden 	Die markierten Objekte werden aus dem Dokument_1 entfernt und in der Zwischenablage abgelegt. Der vorherige Inhalt der Zwischenablage wird gelöscht.	Menü Bearbeiten_Auschneiden *oder* Kontextmenü (rechte Maustaste) *oder* durch Klick auf das Symbol (Icon)
Kopieren 	Eine Kopie der markierten Objekte wird in der Zwischenablage gespeichert. Der vorherige Inhalt der Zwischenablage wird überschrieben.	Menü Bearbeiten_Kopieren *oder* Kontextmenü *oder* Klick auf das Symbol
Einfügen 	Die in der Zwischenablage gespeicherten Objekte werden in das Dokument_2 kopiert. Der Inhalt der Zwischenablage bleibt erhalten.	Menü Bearbeiten_Einfügen *oder* Kontextmenü *oder* Klick auf das Symbol

Andere Methoden erfordern zusätzliche Angaben, wie das Drehen, bei dem man den Drehwinkel angeben muss. Diese Angaben werden dann in die Klammern geschrieben.

Eine interessante Methode ist das 'Gruppieren'. Hier werden mehrere Objekte durch Einrahmen mit dem Mauszeiger und Aufruf dieser Methode zu einem einzigen neuen Objekt zusammengeführt. Allerdings besitzt nun dieses Objekt nicht mehr alle Attribute und Methoden der Einzelobjekte.



Seminar Informatik: Begriffliche Grundlagen der Informatik

BEGRIFFLICHE GRUNDLAGEN DER INFORMATIK

Informationen - Nachrichten – Daten

Zunächst gilt es, Begriffe wie **Informationen, Nachrichten, Daten, Kommunikation,** die für die Informationstechnologie eine wesentliche Rolle spielen, gegeneinander abzugrenzen.

Information ist Wissen, das zweck- und zielgerichtet ist.

Eine **Nachricht** ist eine weitergeleitete Information, die vom Empfänger interpretiert werden muss. Eine Nachricht besteht aus einer sinnvollen Aneinanderreihung von Zeichen und Signalen. **Nachrichten sind Träger von Informationen.**

Eine **Interpretationsvorschrift** gibt an, wie eine Nachricht zu interpretieren ist:

Nachricht → Interpretationsvorschrift → Information

Eine Information kann man somit als eine Erkenntnis betrachten, die man aus einer Nachricht gewinnen kann.

Auch **Daten** sind Träger von Informationen, allerdings zum Zweck der Weiterverarbeitung.

Den Austausch von Informationen bezeichnen wir als Kommunikation.

Für die **Übertragung einer Information** legt man fünf Stationen fest:

Sender - Codierung - Übertragungsmedium (Kanal) - Decodierung - Empfänger.

Auf dem Übertragungsweg (Kanal) kann es zu Störungen kommen, so dass je nach Art der Störung die Nachricht u. U. verstümmelt wird.

Modell für die Übertragung von Daten:

Störung

Sender z.B. Mensch → Codierung z.B. in Schallwellen → Kanal z.B. Luft → Decodierung z.B. durch das Ohr → Empfänger z.B. anderer Mensch

Die Informationen fließen entweder in nur einer (unidirektional), in zwei (bidirektional) oder in vielen Richtungen (multidirektional).

Informationsdarstellung

Informationen können in ganz unterschiedlichen Formen dargestellt werden:

- als gedruckter Text (wie dieses Dokument)
- Zahlen in Tabellen

Grundkurs Seite 3 September 2008

Die Objektstruktur bei Textverarbeitungssystemen

Die Begriffe der objektorientierten Sichtweise übertragen wir nun auf ein Textverarbeitungssystem. Starten wir das Programm, so wird uns eine Instanz der Klasse 'Seite' angezeigt, die über gewisse Standardeigenschaften verfügt. Diese werden aus einer vordefinierten Standardvorlage übernommen und können nun verändert werden.

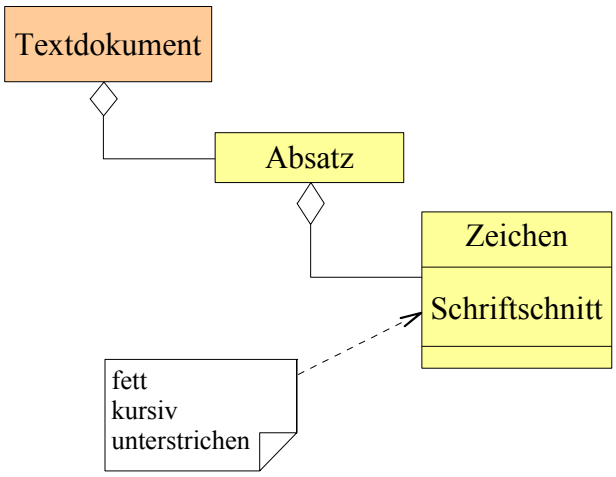
Klasse	**Attribute (Eigenschaften)**	**Methoden**
ZEICHEN	Schriftart Schriftstil Schriftgrad Farbe Unterstreichung	markieren() kopieren() einfügen() setzeSchriftart(Arial) setzeEffekt(Hochgestellt)

Klasse	Attribute (Eigenschaften)	Methoden
ABSATZ	Ausrichtung Einzug_Links Einzug_Rechts Abstand_davor Abstand_danach Zeilenabstand Tabstopps Rahmen Hintergrundfarbe	markieren() kopieren() einfügen() setzeAusrichtung(zentriert) setzeEinzug_Links(1 cm) setzeZeilenabstand(12 pt)
DOKUMENT	Dateiname Dateigröße Dokumentvorlage	speichern_unter(…) Wörter_zählen()

Beispiele für die Schreibweise von Attributen und Methoden:

- Buchstabe1[ZEICHEN].markieren()
- Buchstabe5[ZEICHEN].Schriftgrad = 12 pt
- Überschrift[ABSATZ].Ausrichtung = zentriert
- Absatz1[ABSATZ].Abstand_danach = 6 pt
- Seite2[SEITE].setzeSeitenrand_Links(2,5 cm)
- Arbeitsblatt[DOKUMENT].Dateiname = Blatt_1.doc

Zwischen den einzelnen Klassen bestehen Beziehungen, wie **'enthält'** oder **'ist Teil von'**. Das Diagramm unten zeigt den in der Tabelle beschriebenen Ausschnitt aus dem Beziehungsgeflecht. Es ist als Klassendiagramm mit Aggregation dargestellt.



Die Arbeiten an einem Dokument beschränken sich aber nicht auf das Texterfassen oder Festlegen der Eigenschaften.

Jede Klasse legt für ihre Instanzen neben den typischen Attributen auch eine Reihe von Methoden (Operationen) fest.

Operationen wie 'Markieren', 'Kopieren' oder 'Ausschneiden' sind **Methoden eines Textverarbeitungssystems**. Die Methoden werden durch **Botschaften (Ereignisse)** aufgerufen: z.B. Mausklick, Aufruf von Menübefehlen, Drücken eines Bildsymbols (Icon).

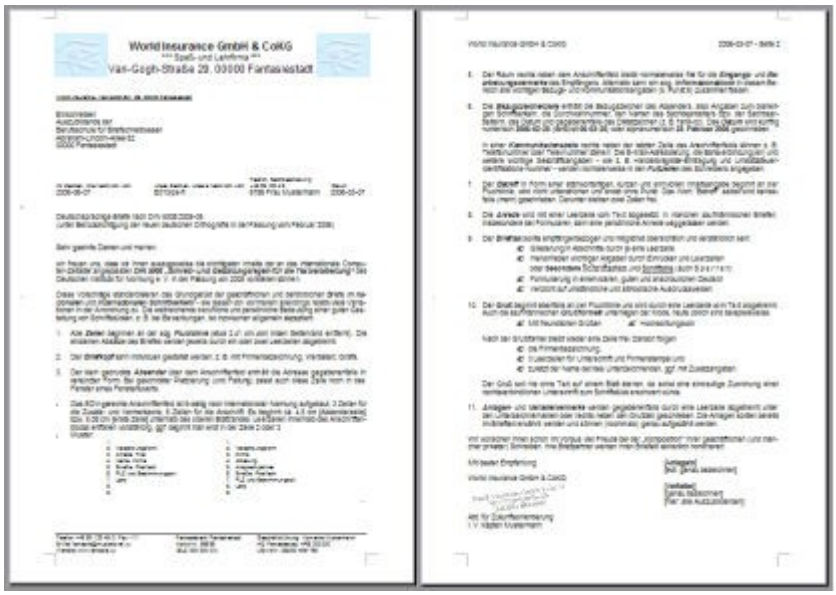
Weitere Beispiele für Methoden:
Seitenansicht, Dokument öffnen, Dokument speichern, Absatz ausschneiden, Silbentrennung, Spracherkennung ...

Über welche Eigenschaften und Methoden ein Objekt verfügt, wird durch die verwendete Software festgelegt.

In den meisten Windows-Anwendungen lassen sich **sowohl Eigenschaften als auch Methoden durch Drücken der rechten Maustaste auswählen**. Es öffnet sich das sog. **Kontextmenü** (siehe Abb.), aus dem man die für die gewählte Klasse gewünschten Eigenschaften bzw. Methoden auswählt.

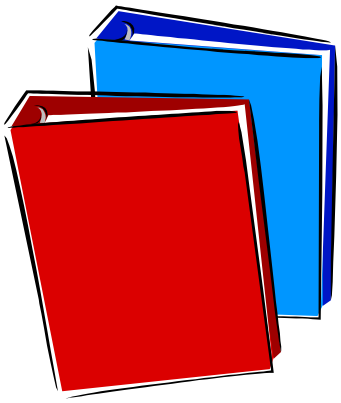
Wie das Programm vorgeht, um die aufgerufene Methode umzusetzen, muss den Anwender nicht kümmern. Beim Autofahren interessiert es den Fahrer auch nicht, was im Motorraum genau vor sich geht, wenn er das Gaspedal tritt. Er weiß aber, dass das Fahrzeug beschleunigt.

An dieser Stelle noch ein Hinweis auf den Textverarbeitungsteil innerhalb des IT-Unterrichts: Neben dem Einüben des Tastschreibens ist auch der Privat- und Geschäftsbrief Inhalt des Lehrplans. Dabei gilt es, die Vorschriften nach der DIN-Norm 5008 zu berücksichtigen. Näheres zu dieser Norm können dem Internet entnommen werden. Dazu eignet sich beispielsweise die entsprechende Seite der Wikipedia-Enzyklopädie:
http://de.wikipedia.org/wiki/DIN_5008_%E2%80%93_Musterbrief



Das Ordner- und Dateisystem

Die Methode 'Dokument speichern' führt uns zum Begriff 'Datei'. Solange wir noch am Dokument arbeiten und es noch nicht gesichert haben, führt es den vom System vorgegebenen Namen 'Dokument1'. Beim Speichern legen wir eine Kopie unseres DokuDments auf dem Datenträger ab.



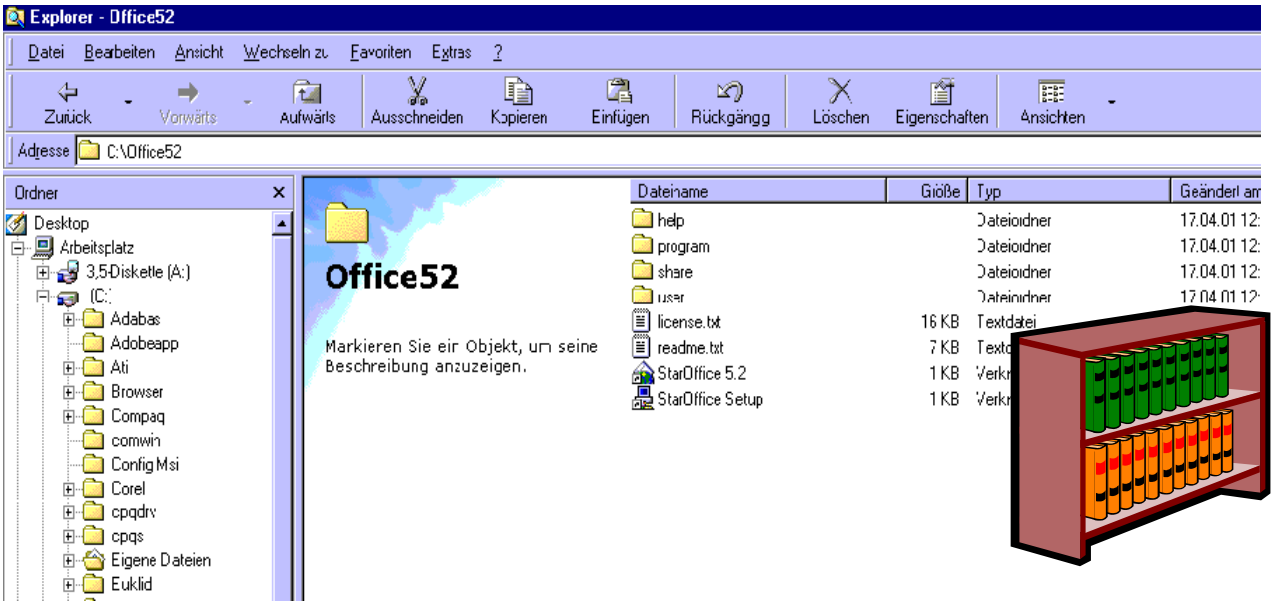
Dadurch erhalten wir eine **Datei**, die ihrerseits in der Regel in einem Ordner untergebracht ist. Dass die Unterscheidung zwischen Dokument und Datei Sinn macht, verdeutlicht folgender Sachverhalt: Wenn wir ein neues Dokument erstellen und es noch nicht gespeichert haben, existiert zwar dieses Dokument und kann bearbeitet werden, ohne dass eine Datei dazu existiert. Bei einem Systemabsturz (Stromausfall) führt das zum Totalverlust der bisherigen Arbeit. Haben wir dagegen eine Datei angelegt, so ist zumindest der Teil der Arbeit gerettet, den wir als letztes gesichert haben. Die Datei ist also ein Behälter für unser Dokument. Ohne Computer entspricht das etwa folgender Analogie: Dem Dokument entsprechen die ausgedruckten Seiten, diese werden in eine Klarsichthülle gegeben (Datei), die wiederum in einem Ordner aufbewahrt wird. Auch Dateien sind Objekte und besitzen Attribute. Die für ein Textverarbeitungssystem bedeutenden Methoden 'Öffnen' und 'Speichern' führen uns zum Dateisystem

Die Methode '**Öffnen**' einer Datei erzeugt in der aufrufenden Software ein neues Dokument aus dem Inhalt der Datei.

Beim Aufruf der Methode '**Speichern**' eines Dokumentes wird auf dem Speichermedium eine **neue Datei erzeugt** bzw. **eine vorhandene überschrieben**.
Dadurch entsteht ein Objekt der **Klasse DATEI**.
Die Methode '**Speichern unter**' bietet die Möglichkeit, der Datei einen neuen Namen zu geben. Beim '**Speichern' eines neuen Dokumentes**, wird automatisch '**Speichern unter**' aufgerufen.

auf der Festplatte und deren Struktur, die für das Wiederfinden gespeicherter Dokumente so wichtig ist. Bei unserer Arbeit müssen wir uns vor dem Speichern von Dokumenten Gedanken machen, wie das Dateisystem in unserem Betriebssystem auf dem Datenträger aussieht bzw. aussehen soll. Dazu zunächst noch einmal folgende Feststellungen:
Dabei sollten wir berücksichtigen, dass die Dateiformate der verschiedenen Textverarbeitungssysteme unterschiedlich und nicht kompatibel sind; d.h. ein in 'Word' geschriebenes Dokument hat als Datei ein Format, das ein anderes Programm i.a. nicht öffnen kann. Es gibt aber durchaus Formate, auf die aus allen Textprogrammen zugegriffen werden kann: das ASCII(Ansi)-Format (Dateierweiterung *.txt) oder das Rich-Text-Format (Dateierweiterung *.rtf). Allerdings gehen beim Speichern in diesen Formaten eine Reihe von Attributwerten verloren.

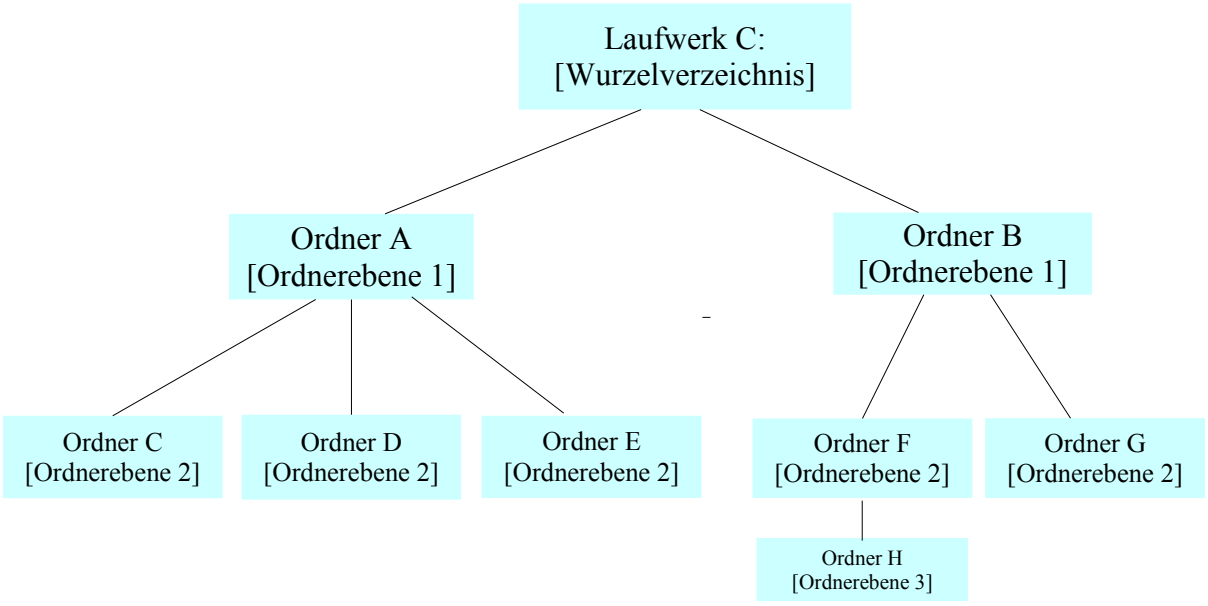
Das Ordner- und Dateiensystem wird im Betriebssystem Windows im **Explorer** dargestellt.



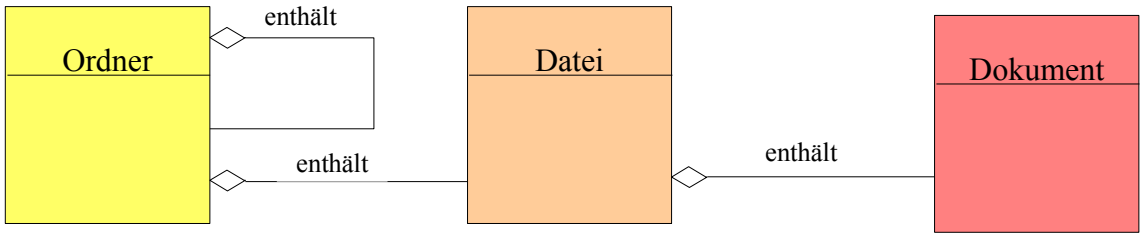
In der Abbildung sehen wir im linken Teil die Ordnerstruktur, während im rechten Teil die im 'geöffneten' Ordner enthaltenen Unterordner bzw. Dateien angezeigt werden. Klickt man auf einen Ordner im linken Fensterausschnitt, so wird der Inhalt des Ordners im rechten Ausschnitt angezeigt. Das Pluszeichen neben dem Ordnersymbol gibt an, dass in diesem Ordner weitere Unterordner existieren. Um sie anzuzeigen, klickt man auf das Pluszeichen. Über den Menübefehl Ansicht kann man zwischen verschiedenen Darstellungsformen von Ordnern und Dateien wechseln. Man erkennt die **hierarchische Struktur des Ordnersystems.**

Ordner können Dateien und/oder andere Ordner enthalten. Jede Datei und jeder Ordner ist aber genau in einem Ordner enthalten. So entsteht ein '**Ordnerbaum**', der genau eine '**Wurzel**' ('Hauptverzeichnis', 'Stammverzeichnis') hat.

Es kann folgendermaßen dargestellt werden (alle Beziehungen sind vom Typ „enthält“):



Klassendiagramm zum System Ordner-Datei-Dokument:



Da Objekte der Klasse ORDNER wiederum Objekte derselben Klasse enthalten können, weist obiges Klassendiagramm eine „enthält" - Beziehung der Klasse ORDNER mit sich selbst auf.

In diesem Zusammenhang sollte noch der Begriff **Pfad** erwähnt werden. **Ein Pfad stellt gewissermaßen den Weg vom Wurzelverzeichnis bis zur Datei dar.** Jede mögliche Verzweigungsstelle wird dabei durch einen 'Backslash' dargestellt.

Beispiel (s.o.): **C:\Freizeit\Sport\Skifahren\Wettkämpfe\Mürren\Strecke.htm**

Auch auf das Ordner- und Dateiensystem lassen sich die Begriffe **Klasse, Objekt, Attribut, Ereignis und Methode** anwenden.

Klasse	*Attribut*	*Attributwert*
Ordner	Name	Freizeit
	Erstellungsdatum	01.01.2000
Datei	Name	Strecke
	Typ	HTML Document
	Änderungsdatum	25.10.00

Beispiele für **Methoden** sind in diesem Zusammenhang Kopieren, Verschieben, Umbenennen, Senden_an oder Öffnen.

SEMI-OOS – Einführung

Zum weiteren Verständnis objektorientierter Modellierung und Programmierung hat Robert Pütterich die Entwicklungsumgebung SEM-OOS entwickelt und wesentliches dazu in einem Skript festgehalten.
Im folgenden sind Teile dieses Skriptes übernommen.

Objektorientierte Softwareentwicklung (mit SEMI-OOS)

Stand: April 2008

1. Vier Phasen der Softwareentwicklung

Analyse

Bei der Analyse steht die Frage im Vordergrund, WAS die Software leisten soll.

Zur Darstellung der Inhalte werden verschiedene Modelle verwendet. Zu einem sehr verbreiteten Standard hat sich die UML (Unified Modeling Language) entwickelt.

Entwurf (Design)

Diese Phase stellt die Frage in den Mittelpunkt, WIE die Problemlösung konkret erreicht werden soll, d.h. es ist festzulegen, wie beispielsweise die technische Umsetzung erfolgen soll bzw. welche Klassen und Algorithmen nötig sind, um die Aufgabe zu bewältigen. Die Darstellung der konkreten Lösungswege erfolgt wiederum mit Modellen.

Implementierung

Umsetzung der Modelle der Entwurfsphase beispielsweise in ein Programm der gewählten Programmiersprache.

Entwicklung / Evaluation

Aus Erfahrungen mit dem System werden Verbesserungsvorschläge erarbeitet und umgesetzt.

2. Die betriebssystemunabhängige Programmiersprache Java

Compiler verschiedener Sprachen wie C++, Pascal oder Basic arbeiteten bisher nach dem Prinzip, dass sie das Programm in höherer Programmiersprache in einen Maschinencode übersetzten, den der Computer dann abarbeiten kann. In der Regel steht dieser Maschinencode in einer Datei mit der Endung „.exe“ und funktioniert nur unter dem Betriebssystem, für das er erzeugt wurde.

Soll nun beispielsweise im Internet ein Programm wie ein Chatfenster zur Verfügung gestellt werden, ist nicht bekannt, welches Betriebssystem der Benutzer hat, d.h. das im ersten Absatz beschriebene Prinzip funktioniert nicht.

Auf diesem Problem basiert die Idee, welche mit Java realisiert wurde: Der Java-Compiler erzeugt aus dem Programm in der höheren Programmiersprache Java einen Zwischencode, den sogenannten Bytecode, der in Dateien mit der Endung „.class“ abgelegt wird. Dieser Bytecode

kann von einem Java-Interpreter, den jeder Benutzer passend zu seinem Betriebssystem kostenlos aus dem Internet beziehen und installieren kann, ausgeführt werden.

Bei Java existiert zudem die Möglichkeit, alle „class“-Dateien zu einer mit dem Java-Interpreter ausführbaren Java-Archiv-Datei (Endung „.jar“) zusammen zu fassen und zu komprimieren. Die Kompression entspricht dem „zip“-Format, sodass „jar“-Dateien auch mit einem entsprechenden „zip“-Programm geöffnet werden können.

Objektorientierung - Grundbegriffe

Die Objektorientierung ist eine Sichtweise, die hilft, Objekte dieser Welt nach einem bestimmten Muster zu schematisieren. Dabei wird jedem Objekt ein bestimmter Typ (Klasse) zugeordnet.

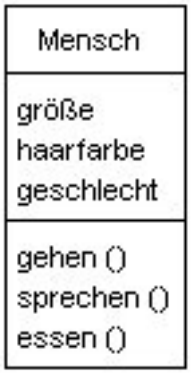
Beispiel: Die Objekte Frau Müller und Herr Meier gehören dem Typ (der Klasse) Mensch an. Sie sind also Objekte (oder Instanzen) der Klasse Mensch.

Oft lassen sich auch speziellere Klassen finden: So könnte Frau Müller als ein Objekt der Klasse Frau und Herr Meier als ein Objekt der Klasse Mann gesehen werden.

Klassen werden beschrieben, indem man Attribute (Eigenschaften) und Methoden (mögliche Handlungen) angibt, wobei eine schrittweise beschriebene Handlung als Algorithmus bezeichnet wird.

So könnten für die Klasse Mensch sicherlich die Attribute größe, haarfarbe und geschlecht und die Methoden gehen(), sprechen() und essen() gefunden werden.

Das UML-Modell dieser Klasse sieht folgendermaßen aus:

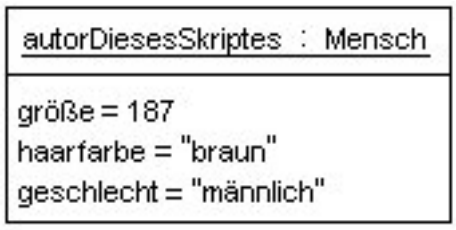


Ein Objekt einer Klasse besitzt zu jedem Attribut einen individuellen Attributwert.

Eine bestimmte Person, wie beispielsweise der Autor dieses Skriptes, besitzt die Attributwerte 1,87m, braun und männlich zu den oben genannten Attributen größe, haarfarbe und geschlecht.

Attributwerte sind Momentaufnahmen, die sich mit der Zeit ändern können.

Das Objekt autorDiesesSkriptes wird in UML wie folgt modelliert:



Im oberen Rechteck wird dabei der Objektname mit Angabe der Klasse unterstrichen und zu jedem Attribut der individuelle Attributwert angegeben. Die Methoden werden beim Objektdiagramm in der Regel weg gelassen, da sie ja bei jedem Objekt äquivalent zur Angabe bei der Klasse sind.

Bei Methoden ist es möglich, Parameter in den runden Klammern anzugeben.

Ein Beispiel hierfür wäre die Methode berechneSumme(p_summand1, p_summand2) mit den Parametern p_summand1 und p_summand2, welche aus zwei ganzen Zahlen die Summe berechnen soll. Um Verwechslungen mit anderen Variablen oder Attributen zu vermeiden, wird empfohlen, jede Parameterbezeichnung mit „p_“ zu beginnen.

Somit könnten in einem Programm dann beispielsweise die Methodenaufrufe **berechneSumme(12, 14)** oder auch **berechneSumme(zahl1, zahl2)**, wenn zahl1 und zahl2 entsprechende Variablen sind, erfolgen. Im ersten Fall ist p_summand1 = 12 und p_summand2 = 14, im zweiten Fall ist p_summand1 = zahl1 und p_summand2 = zahl2. Der nachfolgend aufgeführte Algorithmus (Notation gemäß SEMI-OOS-Syntax, siehe folgende Kapitel dieses Skripts) der Methode berechnet dann aus den Parametern p_summand1 und p_summand2 die Summe und gibt sie aus:

berechneSumme(p_summand1, p_summand2)

```
DefiniereVariable s  als  GanzeZahl
Setze s := p_summand1 + p_summand2
BM.ausgabeText("Die Summe ist " + BM.text_Aus_GanzeZahl(s))
```

Die Methode berechneSumme(p_summand1, p_summand2) könnte aber auch so gestaltet werden, dass sie einen Wert zurück gibt, der dann beispielsweise gleich einer Variable gesetzt wird. Eine entsprechende Anweisung könnte dann **Setze summe := berechneSumme(5, 8)** lauten.

Bei diesem Aufruf ist p_summand1 gleich 5 und p_summand2 gleich 8. Die Methode berechnet daraus die Summe und gibt den Wert 13 zurück, der dann gleich der Variablen summe gesetzt wird. Der Algorithmus würde dann wie folgt aussehen:

berechneSumme(p_summand1, p_summand2) mit Rückgabewert

```
DefiniereVariable s  als  GanzeZahl
Setze s := p_summand1 + p_summand2
methodeBeendenMitRückgabe(s)
```

Anmerkung: Die lokale Variable s ist nur innerhalb dieser Methode gültig und nicht zu verwechseln mit der Variable summe beim Aufruf **Setze summe := berechneSumme(5, 8).**

Eine Methode muss aber keinen Wert zurückgeben. Die Methode ausgabeDerZahlenVon1Bis10() soll beispielsweise lediglich die Zahlen von 1 bis 10 ausgeben, eine Wertrückgabe erscheint nicht sinnvoll.

Konventionen zur Schreibweise

Klassennamen werden immer groß geschrieben, Methoden, Attribute und Parameter beginnen mit einem kleinen Buchstaben.

Setzt sich die Bezeichnung aus mehreren Worten zusammen, so wird alles zusammen geschrieben und ab dem 2. Wort die Teilwörter groß.

Beispiele:

Klassen: Mensch, LandInEuropa

Attribute: größe, haarfarbe, anzahlDerEinwohnerInMillionen

Methoden: gehen(), sprechen(), insAutoSteigen(), demokratischeWahlDurchführen()

Vererbung

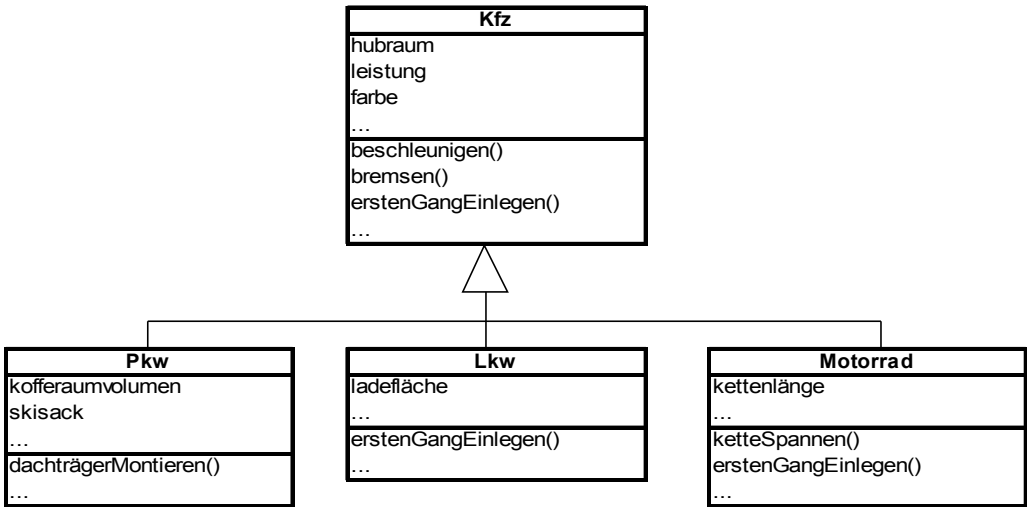
Übernimmt eine Klasse (Unterklasse) alle Attribute und Methoden einer anderen Klasse (Oberklasse), so nennt man dies Vererbung.

Methoden der Oberklasse können dabei überschrieben werden (Polymorphismus: „gleiche Methodenbezeichnung, unterschiedliche Handlung“).

In der Regel ist die Unterklasse eine Spezialisierung der Oberklasse bzw. die Oberklasse eine Generalisierung der Unterklasse.

Beispiel: Oberklasse Kfz, Unterklassen Pkw, Lkw und Motorrad

UML-Darstellung mit Attributen und Methoden



Softwareentwicklung mit SEMI-OOS

http://www.semioos.de

Allgemeines

Für die Softwareentwicklung hat sich die objektorientierte Sichtweise als sehr praktikabel erwiesen, sodass sie heute eine nicht wegzudenkende Rolle spielt.

Sie ermöglicht die Programmierung in sehr realitätsnahen Strukturen und in kleinen überschaubaren Einheiten, welche wieder verwendet und durch die Möglichkeit der Vererbung erweitert werden können.

Grundsätzliche Vorgehensweise

Zunächst werden Klassen (d.h. deren Attribute und Methoden), die als Baupläne für Objekte dienen, definiert. Zu jeder Methode ist zudem ein Algorithmus festzulegen, der die einzelnen Arbeitsschritte, die beim Ausführen einer Methode abzuarbeiten sind, fixiert.

Gemäß den definierten Klassen können dann Objekte erzeugt (instantiiert) werden. Aus dem Zusammenspiel der Objekte mit gegenseitigen Methodenaufrufen ergibt sich der Programmablauf.

Bei der Erstellung objektorientierter Software sind die im vorhergehenden Abschnitt genannten Definitionen in einer objektorientierten Programmiersprache wie z.B. Java zu formulieren, wobei darauf geachtet werden muss, dass die Syntax (Schreibweise) der Programmiersprache exakt eingehalten wird.

Da dies gerade im Unterricht mit großen Schwierigkeiten verbunden ist, wurde SEMI-OOS so konstruiert, dass lediglich die inhaltliche Struktur (Semantik) des Programms einzugeben ist, die Formulierung in der Programmiersprache erfolgt dann automatisch.

Bei Oberflächen, wie sie in der Softwareentwicklung heutzutage üblich sind, ergibt sich der Programmablauf in der Regel ereignisgesteuert. Wird beispielsweise eine Befehlsschaltfläche gedrückt (Ereignis!), so hat dies die Ausführung einer bestimmten Methode zur Folge.

Prinzipiell gibt es eine Vielzahl von Ereignissen, die eintreten können. Neben dem Drücken einer Befehlsschaltfläche sind beispielsweise auch das Drücken einer bestimmten Maustaste oder Mausbewegungen Ereignisse, welche Methodenaufrufe zur Folge haben können.

Im Rahmen der didaktischen Reduktion ist bei SEMI-OOS nur das Ereignis „Drücken einer Befehlsschaltfläche“ zugelassen.

Datentypen

Attributen, Parametern, lokalen Variablen, die innerhalb einer Methode definiert werden können, und Methoden mit Rückgabewert muss ein bestimmter Datentyp zugewiesen werden.

Wir unterscheiden zwischen „primitiven“ Datentypen und Referenzdatentypen.

Unter SEMI-OOS sind folgende „primitive“ Datentypen verwendbar:

Text

Wahrheitswert wahr / falsch

GanzeZahl	liegt im Intervall [-2^{31} ; $2^{31}-1$]	wobei 2^{31} : ca 2,147 Mrd.
GanzeZahlGroß	liegt im Intervall [-2^{63} ; $2^{63}-1$]	
Kommazahl	ca. 15 Stellen Genauigkeit	

Referenzdatentypen verweisen auf Objekte, ihr Datentyp ist die Klasse des Objektes. Aus diesem Grund ist jede Klasse, nachdem sie gespeichert wurde, auch als Datentyp verfügbar. Die Bezeichnung einer Referenzvariable könnte als Objektname gesehen werden, so wie der Name einer Person einen bestimmten Menschen, also ein Objekt der Klasse Mensch, referenziert.

Beispiel:

Der Datentyp einer Referenzvariable autorDiesesSkriptes ist Mensch, autorDiesesSkriptes verweist auf mich, ein Objekt der Klasse Mensch. Damit wäre eine Anweisung **autorDiesesSkriptes.sprechen()**, welche bei mir die Methode sprechen() ausführen würde, möglich.

Praktische Beispiele mit SEMI-OOS

(Ein- und Ausgabe über die Konsole)

Bitte zuerst unter „Einstellungen“ auf „Einsteiger“ und „Deutsch“ stellen.

Konto

Projekt Konto.semi

Wir definieren eine Klasse Konto mit dem Attribut kontostand und den Methoden einzahlen() und auszahlen().

Nach dem Start von SEMI-OOS erscheint die Oberfläche zur Eingabe der Klassenbestandteile, die auch über die Befehlschaltfläche „Klasse“ der Steuerungskonsole im oberen Bereich erreichbar ist.

Hier geben wir nun die Klasse „Konto“, das Attribut „kontostand“ und die Methoden „einzahlen“ und „auszahlen“ ein. Die Bedienung sollte selbsterklärend sein. Bei Unklarheiten hilft auch die knapp gehaltene Anleitung „Kurzreferenz SEMI-OOS“ im Anhang.

Der Einfachheit halber wählen wir für das Attribut kontostand den Datentyp „GanzeZahl“ und führen demnach das Konto in ganzen Euro. Bei den Methoden sind keine Parameter anzugeben, da wir den jeweiligen Betrag bei der Methodenausführung vom Benutzer erfragen wollen. Ein Rückgabewert macht ebenso keinen Sinn, daher belassen wir die Einstellung „kein Rückgabewert“.

Die UML-Darstellung (im Arbeitsbereich rechts) der Klasse sieht nun folgendermaßen aus:

Nun sind noch die Algorithmen für die beiden Methoden (Befehlsschaltfläche „Methode“ der Steuerungskonsole) und eine Startanweisung (Befehlsschaltfläche „Startanweisung“ der Steuerungskonsole) zu definieren.

Die Festlegung der Algorithmen der Methoden erfolgt mit dem Struktogramm nach Nassi-Shneiderman, einer (programmiersprachenunabhängigen) Modellierungstechnik für Algorithmen.

Bei SEMI-OOS wird eine minimale aber effiziente Auswahl an algorithmischen Strukturen angeboten, welche auf der linken Seite angewählt (orange Markierung) und per rechtem Mausklick und dem Menüpunkt „Struktogrammelement einfügen“ ein- bzw. angefügt werden. Klickt man mit der linken Maustaste auf ein Element des Struktogramms, so öffnet sich für dieses Element der Editor im unteren Fensterbereich.

Beim Editieren der Anweisungen bzw. Bedingungen von Struktogrammelementen werden Textbausteine angeboten. Die Bausteine beinhalten auch eine Menge sehr nützlicher Basismethoden einer in SEMI-OOS integrierten Klasse BM (siehe Anhang).

Im Editor gelangt man mit der Cursor-Taste „nach-oben“ in das Auswahlfeld, die Eingabetaste fügt den markierten Baustein ein und mit der Esc-Taste kann das Auswahlfeld stets verlassen werden.

Wird mit dem Eintippen eines Bausteines begonnen, so erfüllt das Auswahlfeld die Aufgabe der Textvervollständigung, welche mit der Eingabetaste dann abgeschlossen werden kann.

Werden nur einstellige Bausteine eingegeben, z.B. bei der Eingabe eines Variablennamens und ist der Baustein eindeutig, so wird automatisch nach dem Tippen des Zeichens der Baustein übernommen.

Wurde ein korrekter und vollständiger Ausdruck zusammengesetzt, so wird die Befehlsschaltfläche „übernehmen“ aktiv, welche dann gedrückt werden kann. Ist „übernehmen“ aktiv, so wird sie beim Fortfahren z.B. mit der Bearbeitung eines anderen Struktogrammelementes automatisch gedrückt. Ist „übernehmen“ nicht aktiv, so werden aktuelle Änderungen beim Fortfahren verworfen. Der Algorithmus für die Methode einzahlen() stellt sich wie folgt dar:

einzahlen()

DefiniereVariable betrag als GanzeZahl
BM.ausgabeTextZU("")
BM.ausgabeText("Einzahlungsbetrag in Euro: ")
Setze betrag := BM.eingabeGanzeZahl()
Setze kontostand := kontostand + betrag
BM.ausgabeTextZU("Aktueller Kontostand: " + BM.text_Aus_GanzeZahl(kontostand) + " Euro")
auszahlen()

Erläuterung:

Die erste Anweisung definiert eine lokale Variable betrag vom Datentyp GanzeZahl.

ausgabeTextZU(p_text) und ausgabeText(p_text) sind sogenannte Basismethoden einer in SEMI-OOS integrierten Klasse BM, welche auf der SEMI-OOS-Konsole mit oder ohne Zeilenumbruch den Text ausgeben, der als Parameter angegeben wird. Die Anweisung BM.ausgabeTextZU(“”)

gibt also eine Leerzeile aus.

In Anweisung vier wird betrag ein Wert zugewiesen. Die Basismethode eingabeGanzeZahl() erfragt dabei den einzuzahlenden Betrag auf der Konsole.

Anweisung fünf setzt den Wert von kontostand auf kontostand (alt) plus den Wert von betrag.

In Anweisung sechs wird dann der neue Kontostand ausgegeben. Anzumerken ist hier, dass Texte mit einem „+“-Zeichen verkettet werden und der Kontostand zur Ausgabe mit der entsprechenden Basismethode zu einem Text verwandelt werden muss.

Nach jeder Einzahlung soll eine Auszahlung folgen, daher wird zum Abschluss die Methode auszahlen() aufgerufen.

Die Struktur des Algorithmus ist denkbar einfach, da nur aufeinander folgende Anweisungen ausgeführt werden.

Der Algorithmus für die Methode auszahlen() stellt sich sehr ähnlich dar:

auszahlen()

```
DefiniereVariable betrag  als  GanzeZahl
BM.ausgabeTextZU("")
BM.ausgabeText("Auszahlungsbetrag in Euro: ")
Setze betrag := BM.eingabeGanzeZahl()
Setze kontostand := kontostand - betrag
BM.ausgabeTextZU("Aktueller Kontostand: " + BM.text_Aus_GanzeZahl(kontostand) + " Euro")
einzahlen()
```

Als Startanweisung geben wir an: **erzeugeNeuesObjekt_Konto().einzahlen()**

Das bedeutet, dass beim Start unseres Programms ein Objekt der Klasse Konto erzeugt (instantiiert) und die Methode einzahlen() ausgeführt wird. Der restliche Programmablauf ergibt sich dann aus den Anweisungen der beiden Methoden, die sich abwechselnd gegenseitig aufrufen.

Drücken wir nun die Befehlsschaltfläche „Implementierung“ in der Steuerungskonsole, können wir die entsprechenden Formulierungen unserer Eingaben in schülergemäßer SEMI-OOS-Sprache oder in Java anzeigen lassen.

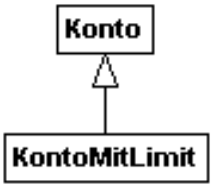
Zur Erläuterung der weiteren Möglichkeiten im Arbeitsbereich „Implementierung“ verweist Herr Pütterich ich an dieser Stelle auf die Kurzreferenz von SEMI-OOS.

KontoMitLimit

Projekt KontoMitLimit.semi

Wir erstellen eine Klasse KontoMitLimit, welche von der Klasse Konto erbt und beim Auszahlen ein Limit berücksichtigen soll.

Dazu drücken wir nach der Definition der Klasse KontoMitLimit die Befehlsschaltfläche „Vererbungsbeziehung“ der Steuerungskonsole und erstellen die entsprechende Beziehung, welche dann in UML wie folgt dargestellt wird:



Für die Klasse KontoMitLimit ist lediglich ein Attribut limit zu definieren und die Methode auszahlen(), die nun das Limit berücksichtigen soll, zu überschreiben (Polymorphismus). Zum Festlegen des Limits und anschließendem Aufruf von einzahlen() wird zudem die Methode initialisiere() erstellt, welche in der Startanweisung ausgeführt werden soll.

Die Methode initialisiere() liest vom Benutzer das limit ein und ruft dann einzahlen() auf:

initialisiere()

BM.ausgabeText("Bitte Kontolimit in Euro festlegen: ")
Setze limit := BM.eingabeGanzeZahl()
einzahlen()

Die Methode auszahlen() benötigt nun eine bedingte Anweisung mit Alternative:

auszahlen()

DefiniereVariable betrag als GanzeZahl
BM.ausgabeTextZU("")
BM.ausgabeText("Auszahlungsbetrag in Euro: ")
Setze betrag := BM.eingabeGanzeZahl()
kontostand - betrag < limit
w / f
BM.ausgabeTextZU("Limit von " + BM.text_Aus_GanzeZahl(limit) + " Euro wird unterschritten. Keine Auszahlung!") / Setze kontostand := kontostand - betrag
BM.ausgabeTextZU("Aktueller Kontostand: " + BM.text_Aus_GanzeZahl(kontostand) + " Euro")
einzahlen()

Als Startanweisung geben wir an: **erzeugeNeuesObjekt_KontoMitLimit().initialisiere()**

EinMalEinsTrainer

Projekt EinMalEinsTrainer.semi

Wir erstellen zunächst folgende Klasse:

EinMalEinsTrainer
aktuellerFaktor1 : GanzeZahl aktuellerFaktor2 : GanzeZahl aktuelleAntwort : GanzeZahl
stelleAufgabeUndLiesAntwort () bewerteAntwort ()

Die Methode stelleAufgabeUndLiesAntwort() ermittelt die beiden Faktoren über die Basismethode ganzeZufallszahl(p_minimum, p_maximum):

stelleAufgabeUndLiesAntwort()

Setze aktuellerFaktor1 := BM.ganzeZufallszahl(1, 20)
Setze aktuellerFaktor2 := BM.ganzeZufallszahl(1, 20)
DefiniereVariable aufgabeText als Text
Setze aufgabeText := BM.text_Aus_GanzeZahl(aktuellerFaktor1) + " * " + BM.text_Aus_GanzeZahl(aktuellerFaktor2) + " = "
BM.ausgabeText(aufgabeText)
Setze aktuelleAntwort := BM.eingabeGanzeZahl()
bewerteAntwort()

Die Methode bewerteAntwort() prüft die Antwort und ruft dann wieder stelleAufgabeUndLiesAntwort() auf:

bewerteAntwort()

aktuelleAntwort = aktuellerFaktor1 * aktuellerFaktor2	
w	f
BM.ausgabeTextZU("Korrekt !")	BM.ausgabeTextZU("Leider falsch !")
	BM.ausgabeText("Die korrekte Antwort lautet: ")
	BM.ausgabeTextZU(BM.text_Aus_GanzeZahl(aktuellerFaktor1 * aktuellerFaktor2))
stelleAufgabeUndLiesAntwort()	

Als Startanweisung geben wir an:

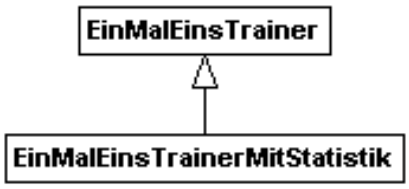
erzeugeNeuesObjekt_EinMalEinsTrainer().stelleAufgabeUndLiesAntwort()

EinMalEinsTrainerMitStatistik

Projekt EinMalEinsTrainerMitStatistik.semi

Die Klasse EinMalEinsTrainerMitStatistik erbt von der Klasse EinMalEinsTrainer, wobei die Attribute nummerDerAufgabe und korrekteAntworten bei der Unterklasse hinzugefügt werden.

Da wir nun eine feste Anzahl an Aufgaben stellen wollen, wird die Methode stelleZehnAufgaben() definiert. bewerteAntwort() wird überschrieben, da wir beim Bewerten ja nun auch eine kleine Statistik ausgeben wollen.



Bei der Methode stelleZehnAufgaben() verwenden wir dabei die algorithmische Struktur „Wiederholung mit Bedingungsprüfung am Anfang“.

stelleZehnAufgaben()

Setze korrekteAntworten := 0
Setze nummerDerAufgabe := 1
wiederhole solange nummerDerAufgabe <= 10
BM.ausgabeTextZU("Aufgabe " + BM.text_Aus_GanzeZahl(nummerDerAufgabe))
stelleAufgabeUndLiesAntwort()
Setze nummerDerAufgabe := nummerDerAufgabe + 1

Hinweis:

Einziger Unterschied bei den in SEMI-OOS angebotenen Wiederholungsvarianten ist, dass bei der Bedingungsprüfung am Ende zunächst der Wiederholungsblock durchgeführt und dann erst die Bedingung geprüft wird, d.h. der Wiederholungsblock wird in jedem Fall mindestens einmal durchgeführt. Bei der Wiederholung mit Bedingungsprüfung am Anfang kann es sein, dass der

Die Methode bewerteAntwort() ermittelt nun auch eine Statistik: Wiederholungsblock gar nicht ausgeführt wird, wenn die Bedingung von Anfang an falsch ist.

bewerteAntwort()

aktuelleAntwort = aktuellerFaktor1 * aktuellerFaktor2	
w	f
BM.ausgabeTextZU("Korrekt !")	BM.ausgabeTextZU("Leider falsch !")
Setze korrekteAntworten := korrekteAntworten + 1	BM.ausgabeText("Die korrekte Antwort lautet: ")
	BM.ausgabeTextZU(BM.text_Aus_GanzeZahl(aktuellerFaktor1 * aktuellerFaktor2))
BM.ausgabeTextZU("Korrekte Antworten: " + BM.text_Aus_GanzeZahl(korrekteAntworten) + " / " + BM.text_Aus_GanzeZahl(nummerDerAufgabe))	
BM.ausgabeTextZU("")	

Als Startanweisung geben wir an:

erzeugeNeuesObjekt_EinMalEinsTrainerMitStatistik().stelleZehnAufgaben()

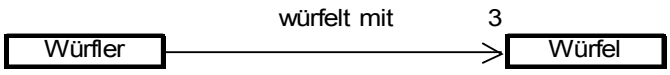
Würfler und Würfel

Projekt Wuerfel.semi

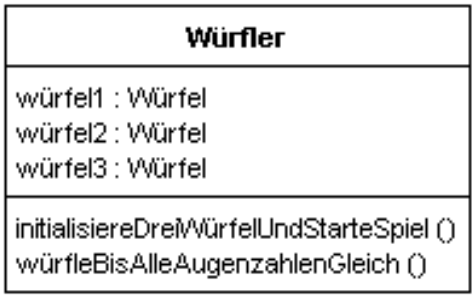
Wir erstellen bei diesem Beispiel zwei Klassen, wobei von der Klasse Würfler ein Objekt und von der Klasse Würfel drei Objekte instantiiert werden.

Der Würfler soll drei Würfel solange werfen, bis ihre Augenzahl identisch ist.

Da der Würfler Zugriff auf die Würfel haben soll, die Würfel aber nicht auf den Würfler (unidirektionale Assoziation), sieht das handschriftlich zu erstellende Klassendiagramm wie folgt aus:

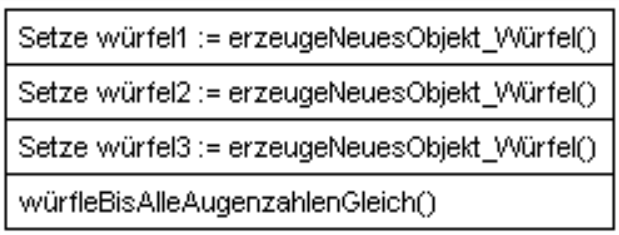


Zur Implementierung der Assoziation erhält die Klasse Würfler drei Attribute vom Typ Würfel, die als Referenzvariablen für die drei Würfel-Objekte dienen:



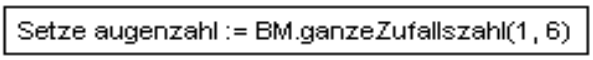
In der Methode initialisiereDreiWürfelUndStarteSpiel() werden nun die drei Würfelobjekte erzeugt und den Referenzattributen zugewiesen:

initialisiereDreiWürfelUndStarteSpiel()



Die Methode würfle() der Klasse Würfel ermittelt lediglich einen Attributwert zum Attribut augenzahl:

würfle()



ermittleAugenzahl() liefert die aktuelle Augenzahl, um zu vermeiden, dass die Klasse Würfler direkt auf das klassenfremde Attribut zugreift.

ermittleAugenzahl()

Hinweis:

Gemäß Kapselungsprinzip sollen auf die Attribute einer Klasse grundsätzlich immer nur klasseneigene Methoden zugreifen. Würde nämlich bei einer Klasse ein unzulässiger Attributwert gesetzt (z.B. bei einer Klasse Bruch das Attribut nenner auf den Wert 0), so ist der Fehler immer nur in der eigenen Klasse und nicht im ganzen Programm zu suchen.

Bei der Klasse Bruch müsste man also beispielsweise dafür sorgen, dass nur klasseneigene Methoden wie z.B. setzeNenner(p_nenner) auf das Attribut nenner zugreifen und dass diese Methoden dafür sorgen, dass der Attributwert auf keinen Fall 0 wird.

In der Methode würfleBisAlleAugenzahlenGleich() wird zunächst die lokale Variable wurf definiert und auf 0 gesetzt. Wir haben hier nun einen typischen Fall vorliegen, um eine Wiederholung mit Bedingungsprüfung am Ende einzusetzen, da auf jeden Fall zuerst einmal gewürfelt werden soll, bevor geprüft wird, ob die drei Würfel die gleiche Augenzahl haben. Besitzen sie die gleiche Augenzahl, so wird die Wiederholung und damit das Spiel beendet.

würfleBisAlleAugenzahlenGleich()

```
DefiniereVariable wurf  als  GanzeZahl
Setze wurf := 0
    Setze wurf := wurf + 1
    BM.ausgabeTextZU(BM.text_Aus_GanzeZahl(wurf) + ". Wurf")
    würfel1.würfle()
    würfel2.würfle()
    würfel3.würfle()
    BM.ausgabeTextZU("Würfel 1: " + BM.text_Aus_GanzeZahl(würfel1.ermittleAugenzahl()))
    BM.ausgabeTextZU("Würfel 2: " + BM.text_Aus_GanzeZahl(würfel2.ermittleAugenzahl()))
    BM.ausgabeTextZU("Würfel 3: " + BM.text_Aus_GanzeZahl(würfel3.ermittleAugenzahl()))
    BM.ausgabeTextZU("")
wiederhole solange  würfel1.ermittleAugenzahl() <> würfel2.ermittleAugenzahl() ODER würfel2.ermittleAugenzahl() <> würfel3.ermittleAugenzahl()
BM.ausgabeTextZU("Spiel beendet!")
```

Als Startanweisung geben wir an:

erzeugeNeuesObjekt_Würfler().initialisiereDreiWürfelUndStarteSpiel()

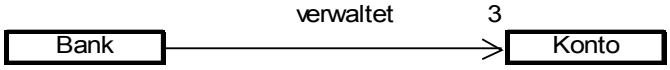
Bank und Konten

Projekt BankKonto.semi

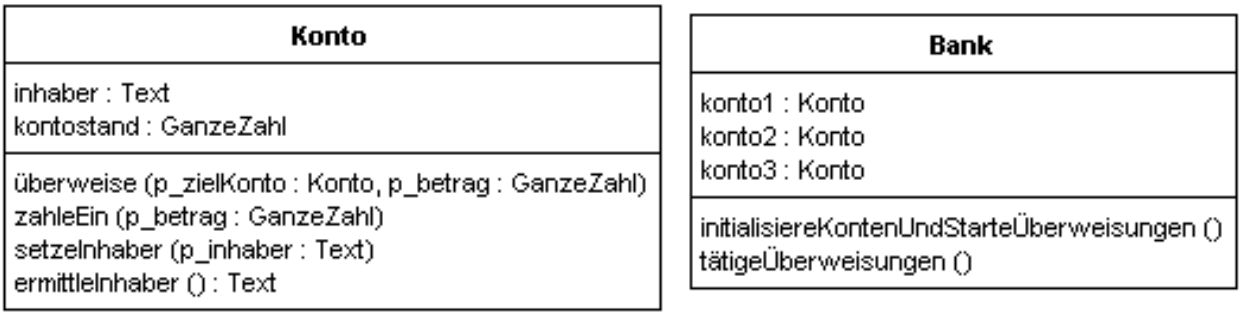
Ähnlich dem vorhergehenden Beispiel erstellen wir wieder zwei Klassen, wobei von der Klasse Bank erneut ein Objekt und von der Klasse Konto drei Objekte instantiiert werden.

Die Bank veranlasst bei den Konten untereinander Überweisungen.

Da die Bank Zugriff auf die Konten haben soll, sieht das Klassendiagramm wie folgt aus:



Die Klassen werden wie folgt definiert:



In der Methode initialisiereKontenUndStarteÜberweisungen() werden drei Kontoobjekte erzeugt und den Referenzattributen zugewiesen. Zugleich werden die Inhaber festgelegt und ein Startguthaben eingezahlt:

initialisiereKontenUndStarteÜberweisungen()

```
BM.ausgabeTextZU("Startguthaben: ")
Setze konto1 := erzeugeNeuesObjekt_Konto()
konto1.setzeInhaber("Franz Meier")
konto1.zahleEin(1000)
Setze konto2 := erzeugeNeuesObjekt_Konto()
konto2.setzeInhaber("Fritz Müller")
konto2.zahleEin(200)
Setze konto3 := erzeugeNeuesObjekt_Konto()
konto3.setzeInhaber("Hansi Huber")
konto3.zahleEin(5000)
BM.ausgabeTextZU("")
tätigeÜberweisungen()
```

Die Methode tätigeÜberweisungen() ruft bei den Konten eine beliebig gewählte Folge der Methode überweise(p_zielkonto, p_betrag) auf:

tätigeÜberweisungen()

```
konto1.überweise(konto2, 500)
konto2.überweise(konto3, 5000)
konto1.überweise(konto3, 500)
konto3.überweise(konto2, 40000)
konto2.überweise(konto1, 2000)
```

Die Methode überweise(p_zielkonto, p_betrag) gibt die Überweisungsdaten aus, zieht den Betrag vom eigenen Konto ab und zahlt es auf dem als Parameter gegebenen Zielkonto ein, indem es die Methode zahleEin(p_betrag) bei diesem Konto aufruft:

überweise(p_zielkonto, p_betrag)

```
BM.ausgabeTextZU(inhaber + " überweist an " + p_zielKonto.ermittleInhaber() + " " + BM.text_Aus_GanzeZahl(p_betrag) + " Euro.")
Setze kontostand := kontostand - p_betrag
BM.ausgabeTextZU("Neuer Kontostand: " + inhaber + " " + BM.text_Aus_GanzeZahl(kontostand) + " Euro.")
p_zielKonto.zahleEin(p_betrag)
BM.ausgabeTextZU("")
```

Die Methode zahleEin(p_betrag) erhöht den Kontostand entsprechend und gibt den neuen Kontostand aus:

zahleEin(p_betrag)

Setze kontostand := kontostand + p_betrag
BM.ausgabeTextZU("Neuer Kontostand: " + inhaber + " " + BM.text_Aus_GanzeZahl(kontostand) + " Euro.")

Die Methoden setzeInhaber(p_inhaber) und ermittleInhaber() wurden lediglich wieder dazu definiert, dass gemäß Kapselungsprinizip die Klasse Bank nicht direkt auf fremde Attribute der Klasse Konto zugreift.

setzeInhaber(p_inhaber)

Setze inhaber := p_inhaber

ermittleInhaber()

methodeBeendenMitRückgabe(inhaber)

Als Startanweisung geben wir an:

erzeugeNeuesObjekt_Bank().initialisiereKontenUndStarteÜberweisungen()

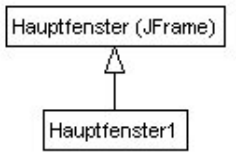
Ein- und Ausgaben mit SEMI-OOS über Oberflächen

Objektorientierte Struktur einer Oberfläche

Objektorientierte Oberflächen sind in der heutigen Softwareentwicklung nicht mehr wegzudenken. Aus diesem Grund ist bei SEMI-OOS ein entsprechender Editor integriert, der über die Befehlsschaltfläche „Oberflächen-Klasse“ der Steuerungskonsole aktiviert wird.

Für jede Oberfläche erstellt SEMI-OOS automatisch eine Klasse. Diese Klasse erbt von einer bereits existenten Klasse Hauptfenster, welche unter Java mit „JFrame“ bezeichnet ist. In der Klasse Hauptfenster (JFrame) sind alle nötigen Attribute und Methoden für ein solches Fenster bereits vorhanden. Durch die Vererbung übernehmen wir diese in unsere neue Fensterklasse.

Da die Unterklasse zunächst den Standardnamen „Hauptfenster1“ erhält, wird nach der Erstellung eines Fensters unter „Vererbungsbeziehung“ Folgendes angezeigt:

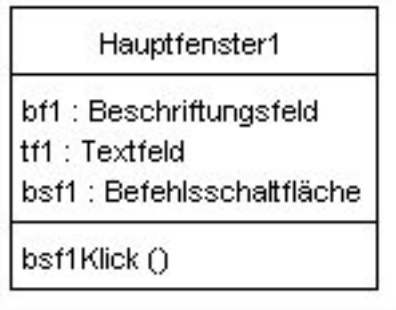


Zeichnen wir in unser Hauptfenster Oberflächenelemente wie beispielsweise Textfelder, Beschriftungsfelder oder Befehlsschaltflächen (dies sind Objekte von Klassen, die unter Java wie die Klasse JFrame bereits vorhanden sind), so wird automatisch für jedes gezeichnete Oberflächenobjekt ein Attribut, d.h. eine Referenzvariable für dieses Objekt, unserer Fensterklasse angelegt. Diese

Attribute erhalten dabei abgekürzte Namen, die jederzeit noch geändert werden können. Textfelder werden beispielsweise tf1, tf2… , Befehlsschaltflächen bsf1, bsf2… und Beschriftungsfelder bf1, bf2… benannt. Für jede Befehlsschaltfläche wird zudem automatisch eine „Klick"-Methode erstellt, welche beim Programmablauf durch das Drücken der entsprechenden Befehlsschaltfläche ausgeführt wird.

Ändert man nachträglich diese Namen ab, so ist zu beachten, dass identische Namen für verschiedene Elemente nicht erlaubt sind und SEMI-OOS daher die letzte Tastatureingabe, die zu einem solchen Fehler führen würde, verwirft.

Wird beispielsweise mit dem Editor folgendes Fenster erstellt, welches ein Beschriftungsfeld, ein Textfeld und eine Befehlsschaltfläche enthält, so entsteht dabei automatisch, verbunden mit der entsprechenden Vererbungsbeziehung, die aufgeführte Klasse:



Da jedes Oberflächenelement Methoden besitzt, die entsprechend im Editor als Textbausteine angeboten werden, können In der Methode dieser Klasse beispielsweise Anweisungen wie bf1.setzeBeschriftung(„hallo") oder tf1.ermittleText() erfolgen.

Java Applikationen und Java Applets

Programme mit Oberflächen können unter „Implementierung" als Applikation („Java-Archiv (*.JAR) erzeugen und ausführen"), d.h. als ein unabhängig von SEMI-OOS lauffähiges Programm, oder aber auch als Java-Applet (Java-Applet-Archiv (*.JAR) und HTML-Gerüst erzeugen) implementiert werden.

Bei der Implementierung als Applet muss genau eine Oberflächenklasse im Projekt vorhanden sein und die Startanweisung mit „erzeugeNeuesObjekt_NameDerFensterklasse" beginnen. Um alle weiteren technischen Details kümmert sich SEMI-OOS.

Ein Applet läuft innerhalb einer HTML-Internetseite. Ein Gerüst für diese HTML-Seite, welche nur die Einbindung des Applets beinhaltet, wird von SEMI-OOS erzeugt und in dem Verzeichnis, in dem sich auch die Projektdatei befindet, abgelegt. Diese Seite kann beliebig mit einem entsprechenden HTML-Editor erweitert werden.

Beispiele

Multiplikator

Projekt: Multiplikator_Oberflaeche.semi

Unter „Oberflächen-Klasse“ wird dazu folgende Oberfläche erstellt und die Bezeichnungen der Oberflächenelemente auf tf_faktor1, tf_faktor2, bsf_produktBerechnen und bf_ergebnis umbenannt. Der Name der Klasse wird ebenfalls von Hauptfenster1 auf MultiplikatorOberfläche geändert:

MultiplikatorOberfläche
tf_faktor1 : Textfeld tf_faktor2 : Textfeld bsf_produktBerechnen : Befehlsschaltfläche bf_ergebnis : Beschriftungsfeld
bsf_produktBerechnenKlick ()

Der Algorithmus der Methode definiert zunächst die lokalen Variablen faktor1, faktor2 und ergebnis:

bsf_produktBerechnenKlick()

DefiniereVariable faktor1 als GanzeZahl
DefiniereVariable faktor2 als GanzeZahl
DefiniereVariable ergebnis als GanzeZahl
Setze faktor1 := BM.ganzeZahl_Aus_Text(tf_faktor1.ermittleText())
Setze faktor2 := BM.ganzeZahl_Aus_Text(tf_faktor2.ermittleText())
Setze ergebnis := faktor1 * faktor2
bf_ergebnis.setzeBeschriftung("Ergebnis: " + BM.text_Aus_GanzeZahl(ergebnis))

Anschließend wird faktor1 gleich dem Textinhalt des Textfeldes tf_faktor1 und faktor2 gleich dem Textinhalt von tf_faktor2 gesetzt. Der jeweilige Textinhalt wird mit der Methode ermittleText() ausgelesen und muss dann noch zur ganzen Zahl konvertiert werden.

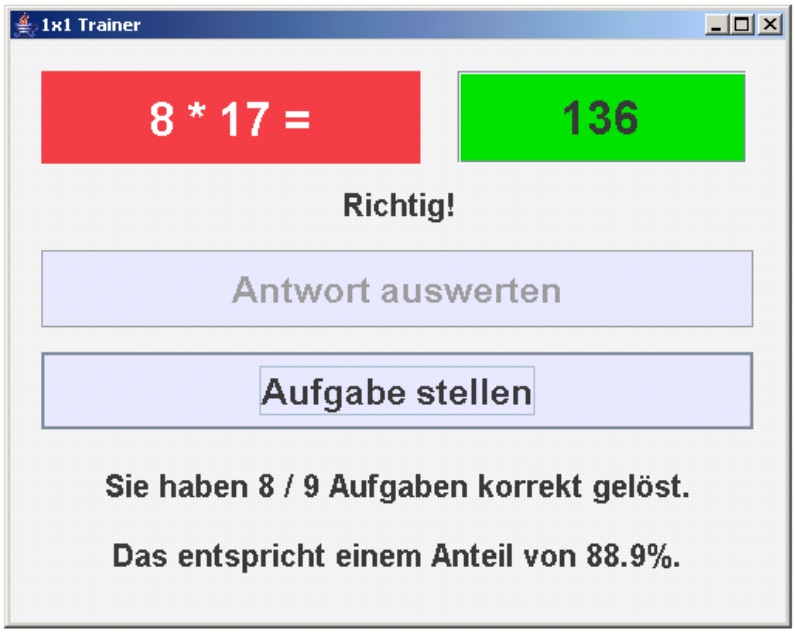
Die Methode setzeBeschriftung(…) erwartet als Parameter einen Text, welcher aus „Ergebnis: “ und dem zum Text gewandelten Ergebnis zusammengesetzt wird.

Die Startanweisung lautet: **erzeugeNeuesObjekt_MultiplikatorOberfläche()**

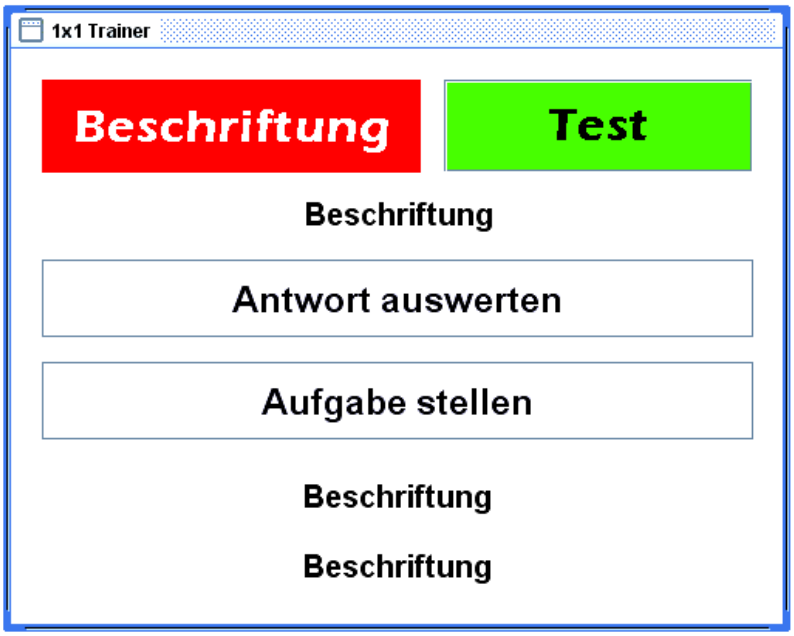
EinMalEinsTrainer mit Oberfläche

Projekt EinMalEinsTrainer_Oberflaeche.semi

Das Endprodukt soll folgendes Erscheinungsbild besitzen:



Dazu wird eine entsprechende Oberfläche mit zugehöriger Klasse erstellt:



Die Attribute aktuellerFaktor1, aktuellerFaktor2, aktuelleAntwort, korrekteAntworten und anzahlAufgaben wurden dabei mit dem Klasseneditor (Befehlsschaltfläche „Klasse“ der Steuerungskonsole) ergänzt, die anderen Attribute entstanden durch das Zeichnen der Oberfläche im Editor.

Ebenso wurden die Methoden initialisiere(), aufgabeStellen() und statistikAusgeben() ergänzt und die Methoden bsf_antwortAuswertenKlick() und bsf_aufgabeStellenKlick() durch das Zeichnen erstellt.

Die ergänzten Attribute und Methoden sind beim Klassendiagramm durch eine etwas größere Lücke von den durch die Oberflächenerstellung automatisch erzeugten abgesetzt.

Die Startanweisung **erzeugeNeuesObjekt_EinMalEinsTrainerOberfläche().initialisiere** erzeugt ein Objekt der Klasse EinMalEinsTrainerOberfläche und ruft initialisiere() auf.

Im Folgenden sind alle Struktogramme der definierten Methoden aufgelistet.

initialisiere()

bf_aufgabe.setzeBeschriftung("")
bf_statistik1.setzeBeschriftung("")
bf_statistik2.setzeBeschriftung("")
bf_kommentar.setzeBeschriftung("")
tf_ergebnis.setzeText("")
aufgabeStellen()

aufgabeStellen()

Setze aktuellerFaktor1 := BM.ganzeZufallszahl(1, 20)
Setze aktuellerFaktor2 := BM.ganzeZufallszahl(1, 20)
bf_aufgabe.setzeBeschriftung(BM.text_Aus_GanzeZahl(aktuellerFaktor1) + " * " + BM.text_Aus_GanzeZahl(aktuellerFaktor2) + " =")
bsf_aufgabeStellen.setzeAktiviert(falsch)
bsf_antwortAuswerten.setzeAktiviert(wahr)
setzeStandardBefehlsschaltfläche(bsf_antwortAuswerten)
tf_ergebnis.setzeText("")
bf_kommentar.setzeBeschriftung("")

bsf_aufgabeStellenKlick()

aufgabeStellen()

bsf_antwortAuswertenKlick()

Setze aktuelleAntwort := BM.ganzeZahl_Aus_Text(tf_ergebnis.ermittleText())	
aktuelleAntwort = aktuellerFaktor1 * aktuellerFaktor2	
w	f
Setze korrekteAntworten := korrekteAntworten + 1	bf_kommentar.setzeBeschriftung("Leider falsch! Das Ergebnis lautet: " + BM.text_Aus_GanzeZahl(aktuellerFaktor1 * aktuellerFaktor2))
bf_kommentar.setzeBeschriftung("Richtig!")	
Setze anzahlAufgaben := anzahlAufgaben + 1	
statistikAusgeben()	
bsf_aufgabeStellen.setzeAktiviert(wahr)	
bsf_antwortAuswerten.setzeAktiviert(falsch)	
setzeStandardBefehlsschaltfläche(bsf_aufgabeStellen)	

statistikAusgeben()

```
bf_statistik1.setzeBeschriftung("Sie haben " + BM.text_Aus_GanzeZahl(korrekteAntworten) + " / " + BM.text_Aus_GanzeZahl(anzahlAufgaben) + " Aufgaben korrekt gelöst.")
DefiniereVariable prozent  als  Kommazahl
DefiniereVariable prozentGerundet  als  Kommazahl
Setze prozent := 100 * BM.kommazahl_Aus_GanzeZahl(korrekteAntworten) / BM.kommazahl_Aus_GanzeZahl(anzahlAufgaben)
Setze prozentGerundet := BM.kommazahl_Aus_GanzeZahl(BM.ganzeZahl_Aus_Kommazahl(prozent * 10 + 0.5)) / 10
bf_statistik2.setzeBeschriftung("Das entspricht einem Anteil von " + BM.text_Aus_Kommazahl(prozentGerundet) + "%.")
```

Multiplikator mit separater Steuerungsklasse

Projekt: Multiplikator_Oberflaeche_SeparateSteuerungsklasse.semi

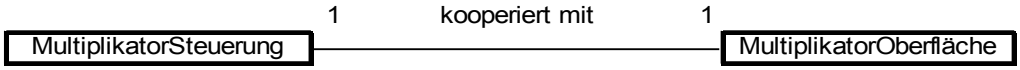
MVC (Model – View – Controller)

Bei der Erstellung von Software sollte man sich eigentlich stets an das Prinzip halten, Model (Datenmodell), View (Oberfläche) und Controller (Steuerung) in separate Programmteile, d.h. in separate Klassen, aufzuspalten.

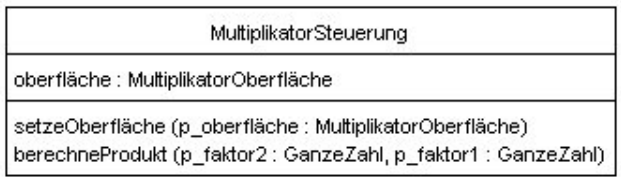
Die Berücksichtigung dieses Prinzips soll nun am einfachen Beispiel des Multiplikators aufgezeigt werden:

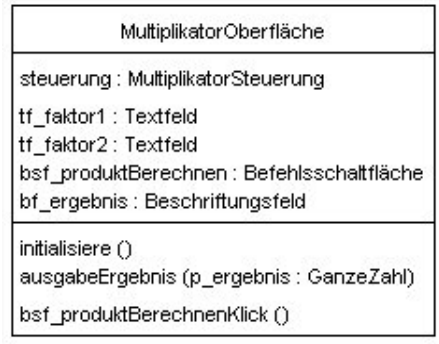
Da die Daten beim Multiplikator vernachlässigbar sind, verzichten wir auf die Dreiteilung. Eine Trennung in eine Steuerungsklasse (MultiplikatorSteuerung) und eine Oberflächenklasse (MultiplikatorOberfläche) erscheint jedoch sinnvoll.

Wir haben dann zwei Klassen und eine bidirektionale 1:1 Assoziation, da jedes Steuerungsobjekt genau auf ein Oberflächenobjekt zugreift und jedes Oberflächenobjekt genau ein Steuerungsobjekt mit eingegebenen Daten versorgen soll:



mplementiert wird diese Assoziation mit einem Attribut steuerung (Datentyp MultiplikatorSteuerung) der Oberflächenklasse und einem Attribut oberfläche (Datentyp MultiplikatorOberfläche) der Steuerungsklasse, d.h. jede Klasse besitzt eine Referenz auf ein Objekt der anderen Klasse.





Man hat beim Programmstart dann nur noch dafür zu sorgen, dass beide Attribute tatsächlich auf ein Objekt der Gegenseite verweisen und nicht auf ein „nullObjekt“, d.h. ins Leere.

Dies geschieht folgendermaßen:

Die Startanweisung **erzeugeNeuesObjekt_MultiplikatorOberfläche().initialisiere()** erzeugt ein Objekt der Klasse MultiplikatorOberfläche und führt dessen nachfolgend aufgeführte Methode initialisiere() aus:

initialisiere()

Setze steuerung := erzeugeNeuesObjekt_MultiplikatorSteuerung()
steuerung.setzeOberfläche(diesesObjekt)

Der Algorithmus der Methode setzeOberfläche(p_oberfläche : MultiplikatorOberfläche) der Steuerungsklasse besteht nur aus einer Anweisung und setzt den erhaltenen Parameter gleich ihrem Attribut oberfläche:

setzeOberfläche(p_oberfläche)

Setze oberfläche := p_oberfläche

Damit ist die Assoziation realisiert.

Die jeweiligen Methoden erledigen die Tätigkeiten, welche in der jeweiligen Klasse als Aufgabenbereich gesehen werden:

Die nachfolgend aufgeführte Methode bsf_ produktBerechnenKlick () der Oberflächenklasse liest die Textfelder aus, die Berechnung ist die Aufgabe der Steuerung:

bsf_produktBerechnenKlick()

DefiniereVariable faktor1 als GanzeZahl
DefiniereVariable faktor2 als GanzeZahl
Setze faktor1 := BM.ganzeZahl_Aus_Text(tf_faktor1.ermittleText())
Setze faktor2 := BM.ganzeZahl_Aus_Text(tf_faktor2.ermittleText())
steuerung.berechneProdukt(faktor1, faktor2)

Die Methode berechneProdukt() der Steuerungsklasse berechnet das Ergebnis, wobei dessen Ausgabe wiederum Aufgabe der Oberfläche ist:

berechneProdukt()

```
DefiniereVariable ergebnis  als  GanzeZahl
Setze ergebnis := p_faktor1 * p_faktor2
oberfläche.ausgabeErgebnis(ergebnis)
```

Die Oberfläche gibt das Ergebnis mit der Methode ausgabeErgebnis(p_ergebnis : GanzeZahl) auf dem Beschriftungsfeld bf_ergebnis aus:

ausgabeErgebnis(p_ergebnis)

```
bf_ergebnis.setzeBeschriftung("Ergebnis: " + BM.text_Aus_GanzeZahl(p_ergebnis))
```

Die Aufteilung in zwei Klassen mag bei diesem Beispiel lächerlich erscheinen. Die Einhaltung dieses Prinzips bringt aber bei komplexeren Programmen wesentliche Vorteile da kleinere und sinngemäß getrennte Einheiten insbesondere im Rahmen der Vererbung Vorteile u.a. durch mehr Flexibilität bringen.

Weitere Beispiele mit Oberflächen

Weitere Beispiele mit Oberflächen und separaten Steuerungsklassen finden sich im Internet unter http://www.semioos.de bei den „Applets“.

Tabellenkalkulation

Objekte in der Tabellenkalkulation

Tabellenkalkulationsprogramme gehören zu den sogenannten Standardprogrammen. Die zwei Begriffe **Tabelle** und **Kalkulation**, weisen auf die zwei grundsätzlichen Möglichkeiten dieser Programme hin: Daten können übersichtlich und geordnet dargestellt werden und es sind Berechnungen mit diesen Daten möglich. Grundsätzlich verwendet man ein Tabellenkalkulationsprogramm für Aufgaben, die man ohne Computer auch auf einem Rechenblatt lösen würde. Sie eignen sich besonders für die Lösung von Problemen, die ein Durchrechnen eines Lösungsweges bei Planungs- und Kalkulationsaufgaben mit unterschiedlichen Eingangsdaten erfordern.

Beim Start eines Tabellenkalkulationsprogramms wird stets eine leere Arbeitsmappe mit mehreren **Arbeitsblättern** geöffnet. Das Arbeitsblatt besteht aus **Zellen**, die in **Zeilen** (untereinander) und **Spalten** (nebeneinander) angeordnet sind. Mehrere Zellen können zu **Bereichen zusammengefasst** werden. Zeilen werden in allen Programmen durch Ziffern gekennzeichnet. Für Spalten gibt es, abhängig vom jeweiligen Programm, zwei unterschiedliche Lösungen: entweder werden sie mit Buchstaben bezeichnet oder ebenfalls durch Zahlen.

	A	B
1		
2		
3		

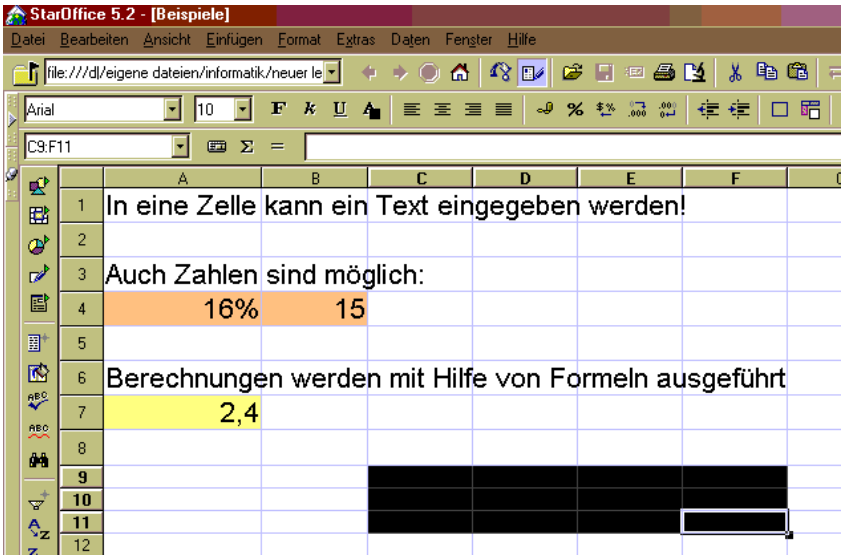
	1	2
1		
2		
3		

In jede **Zelle** können **Texte, Zahlen oder Formeln** eingegeben werden. Man erkennt, dass Zahlen und Texte in den Zellen unterschiedlich angezeigt werden: Texte linksbündig, Zahlen rechtsbündig. Das bedeutet, das Programm erkennt weitgehend selbstständig, welcher Datentyp vorliegt. Das ist auch sehr wichtig, denn schließlich hängen die Bearbeitungsmöglichkeiten von Daten von ihrem Typ ab. Mit Zahlen kann man Berechnungen durchführen, während Texte z.B. aneinander gehängt werden können. Auch das Datum oder die Uhrzeit stellen einen eigenen Datentyp dar.

Auch auf die Tabellenkalkulation lassen sich die Begriffe **Objekte, Eigenschaften und Operationen (Methoden)** anwenden.

Objekte (Zellen, Zeilen, Spalten, Bereiche, Rechenblätter, Arbeitsmappen) sind die Elemente, die man mit dem Programm bearbeiten kann.

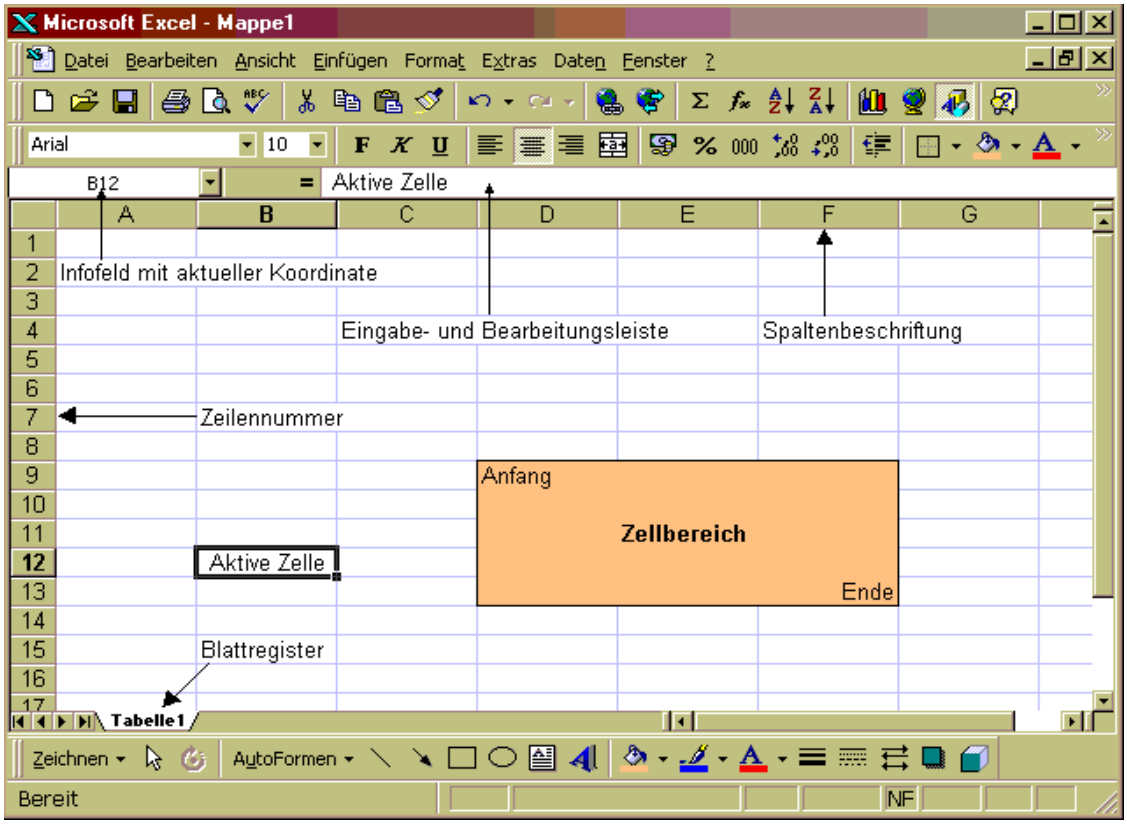
Attribute (Format, Ausrichtung, Schriftart, -größe, -stil, Farben) können den ausgewählten Objekten zugewiesen werden.



Durch **Operationen (Zellen ändern, löschen, schützen; Spaltenbreite ändern; Zeilen oder Spalten einfügen, löschen; Zeilen sortieren; Zellen und Bereiche kopieren, bewegen, benennen; Rechenblätter laden, speichern, drucken)** können die Objekte verändert werden.

In allen Tabellenkalkulationen besteht die Möglichkeit **Formeln** (Funktionen) zur Berechnung einzusetzen. Zur Formulierung sind Zahlen, Zell- und Bereichsbezüge, Klammern, Operatoren und Standardfunktionen zugelassen. Um die Unterschiede zwischen Text-, Zahlen- und Formeleingaben zu erkennen, probieren wir es einfach einmal aus. Erkennungszeichen für eine Formeleingabe ist das vorangestellte Gleichheitszeichen. Danach steht entweder ein Zellbezug (Zelladresse) oder eine Funktion mit in Klammern eingeschlossenem Argument.

Aufbau eines Rechenblatts:



Beispiele:

- Nun soll ein Angebotsvergleich durchgeführt werden. Die Texte bzw. Zahlenwerte können der nachfolgenden Abbildung entnommen werden. Die Tabelle soll sinnvoll formatiert und das günstigste Angebot ermittelt werden.

	A	B	C	D
1	Angebotsvergleich			
2				
3		Angebot 1	Angebot 2	Angebot 3
4	Pos 1	98,4	91,78	103,86
5	Pos 2	23,66	25,77	24
6	Pos 3	155	156,88	147,5
7	Pos 4	44,7	46,9	43,8
8	Pos 5	4,95	5,77	5,55
9	Summe			
10				

- Jemand besitzt drei Sparbücher mit unterschiedlichen Kapitaleinlagen und Zinssätzen. Mit Hilfe eines Rechenblattes soll nach Eingabe der Guthaben und Zinssätze berechnet werden, welche Zinsen nach einem Jahr anfallen, wie groß das jeweilige Endkapital ist und welches Gesamtkapital entstanden ist.

	A	B	C	D
1	Kapital	1.000,00 €	1.500,00 €	5.000,00 €
2	Zinssatz	3,75%	4,00%	4,50%
3	Zinsen			
4	Endkapital			
5				

- Es soll ein Tabellenblatt (s.u.) erstellt werden, bei dem aus dem neu einzugebenden Tachostand und dem Benzinverbrauch in Liter die gefahrenen Kilometer sowie der Verbrauch pro 100 km berechnet werden.

	A	B	C	D	E	F	G
1	Benzinverbrauch						29.08.01
2							
3	**Datum**	**km – alt**	**km – neu**	**Gefahrene km**	**Benzin-menge in l**	**Verbrauch pro 100 km**	**Gesamtdurch-schnitt**
4	01.05.01	32645	32978	333	29,5	8,86	8,86
5	04.05.01	32978	33567	589	50	8,49	8,62
6	06.05.01	33567	33967	400	34,7	8,68	8,64
7	07.05.01	33967	34487	520	46,9	9,02	8,75
8	10.05.01	34487	34921	434	40,3	9,29	8,85
9	13.05.01	34921	35555	634	54,8	8,64	8,80

Bei allen obigen Beispielen bietet ein Tabellenkalkulationsprogramm den Vorteil, dass eine Formel nur einmal eingegeben werden muss. In die benachbarten Zellen, in denen im Prinzip gleich aufgebaute Formeln benötigt werden, kann die erste Formel kopiert werden. Die sog. Zellbezüge (Zelladressen) werden dabei automatisch angepasst. Man spricht von relativer Adressierung. Insbesondere bei größeren Tabellen ist das eine ungeheuere Vereinfachung.

Bei der **relativen Adressierung** wird die Lage einer Zelle relativ zu einer Bezugszelle angegeben und beim Kopieren werden die Zellbezüge automatisch angepasst.

Vorteile dieser Adressierungsart sind:

- Formeln können leicht kopiert werden
- Bereiche können verschoben werden
- Zeilen und Spalten können eingefügt werden, ohne dass sich die Formeln ändern.

Im folgenden Beispiel benötigen wir jedoch eine weitere Adressierungsart:

- Für mehrere Nettobeträge soll die Mehrwertsteuer berechnet und anschließend der Bruttobetrag ermittelt werden. Damit man bei einer eventuellen Änderung des Mehrwertsteuersatzes diesen nicht in allen verwendeten Formeln ändern muss, gibt man ihn einmal in einer Extrazelle an.

	A	B	C	D
1	Mehrwertsteuersatz	16%		
2				
3	Nettobetrag	1.000,00 €	1.500,00 €	5.000,00 €
4	MWST			
5	Bruttobetrag			

Bei der **absoluten Adressierung** wird eine Verbindung zu einer bestimmten **festen Zelle** hergestellt. Es besteht folgende Bezeichnervereinbarung:

relative Adressierung: **B1** **absolute** Adressierung: **B1**

Eine weitere Adressierungsart ist die **Adressierung mit Namen.** Hier werden Zellen bzw. Bereichen **Namen zugeordnet**, die dann z.B. in Formeln verwendet werden können. Verwendet man aussagekräftige Namen, so ist diese Adressierungsart oft den anderen vorzuziehen!

Die wichtigsten Funktionen in einer Tabellenkalkulation:

- **ANZAHL**(*Bereich*) zählt, in wie vielen Zellen Zahlenwerte eingetragen sind.
- **GANZZAHL**(*Wert*) liefert den ganzzahligen Anteil von Wert.
- **MAX**(*Bereich*) liefert die größte Zahl aus dem angegebenen *Bereich*.
- **MIN**(*Bereich*) liefert die kleinste Zahl aus dem angegebenen *Bereich*.
- **MITTELWERT**(*Bereich*) berechnet den Durchschnittswert der im *Bereich* eingetragenen Zahlenwerte.
- **RUNDEN** (*Wert;2*) rundet Wert auf zwei Nachkommastellen.
- **SUMME**(*Bereich*) liefert als Ergebnis die Summe aller Zahlen des angegebenen *Bereichs*.
- **WENN**(*Bedingung;Dann-Wert;Sonst-Wert*) liefert bei erfüllter Bedingung den *Dann-Wert*, bei Nichterfüllung den *Sonst-Wert*.

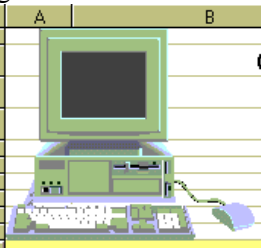
Problemlösungsstrukturen bei der Tabellenkalkulation

Bei der Lösung von Aufgabenstellungen mit Hilfe eines Tabellenkalkulationsprogramms kommen auch hier die drei Strukturelemente zur Anwendung:

- **Sequenz**
- **Auswahl**
- **Wiederholung**

Das Beispiel Rechnung ist geeignet, sowohl die Sequenzstruktur als auch die Auswahl mit den Schülern anhand der Tabellenkalkulation zu erarbeiten.

Berechne die Summe der Einzelbeträge
Subtrahiere von Summe S den Rabatt R
Addiere die Mehrwertsteuer
Subtrahiere den Skontobetrag
Gib das Endergebnis aus



Computerhandelsgesellschaft

E. Müller & Söhne

Rechnung

		für Felix Krull	Datum:	24.02.02
Anzahl		Artikel	Einzelpreis	Gesamtpreis
		Computer Typ 1	644,00 €	0,00 €
	1	Computer Typ 2	875,00 €	875,00 €
		Monitor 17"	159,00 €	0,00 €
	1	Monitor 19"	398,00 €	398,00 €
		DVD-Laufwerk	75,00 €	0,00 €
	1	Brenner	119,00 €	119,00 €
	50	CD-R	0,69 €	34,50 €
	10	CD-R-RW	1,49 €	14,90 €
				1.441,40 €
		Rabatt	5%	72,07 €
				1.369,33 €
		MWST	16%	219,09 €
		Rechnungsbetrag		1.588,42 €
		Bei Zahlung bis zum	04.03.02	
		können Sie noch 2% Skonto =	31,77 €	
		abziehen. Zu zahlen sind dann	1.556,65 €	

Sequenz: Zur Berechnung des zu zahlenden Endbetrags wird der Reihe nach zunächst die Summe der Einzelbeträge gebildet, dann der Rabatt abgezogen, die Mehrwertsteuer addiert und zum Schluss noch das Skonto subtrahiert.

Im Unterricht sollte man hier noch herausstellen, dass, nachdem alle Formeln eingegeben sind, die Berechnung der einzelnen Zellwerte so schnell erfolgt, dass dabei die Sequenzstruktur nicht mehr direkt erkennbar ist.

Auswahl: Die Auswahlstruktur kommt zur Anwendung, wenn man die Aufgabenstellung folgendermaßen abändert: Ist die Summe der Einzelposten 1000,-- € oder höher, beträgt der Rabatt 10%, ansonsten nur 5%.

S >= 1000 ?	
ja	nein
Rabatt = 10%	Rabatt = 5%
Subtrahiere Rabatt von Summe	
:	

Die Formel in Zelle C25:

= Wenn(D24 >= 1000; 10%; 5%)

Ähnlich könnte man für die beiden letzten Posten folgende Bedingung formulieren: z.B. kostet eine CD-R 0,79 €, bei Abnahme von mindestens 50 Stück nur 0,69 €.

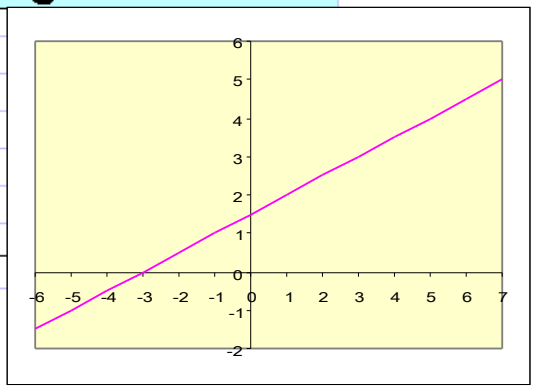
Wiederholung: Bei diesem Beispiel wird auf ein Bankkonto jährlich ein bestimmter Betrag eingezahlt und am Ende des Jahres mit einem gleichbleibenden Zinssatz verzinst. Die Wiederholungen stehen in diesem Beispiel in unmittelbar untereinander liegenden Zeilen und können durch Kopieren in diese übertragen werden.

	A	B	C
1	Ratensparen mit jährlicher Sparsumme		
2			
3		Eingaben	Ergebnisse
4	Rate pro Jahr	1.200,00 DM	Endsumme
5	Zinssatz	4%	8.277,95 DM
6	Beginn	2001	Zinssumme
7	Laufzeit	6 Jahre	1.077,95 DM
8			
9	Jahr	Kapital	Zinsen
10	2001	1.200,00 DM	48,00 DM
11	2002	2.448,00 DM	97,92 DM
12	2003	3.745,92 DM	149,84 DM
13	2004	5.095,76 DM	203,83 DM
14	2005	6.499,59 DM	259,98 DM
15	2006	7.959,57 DM	318,38 DM
16			

RateProJahr = 1200 €	
Kapital = 0 €	
Zinssatz = 4%	
Wiederhole für Zähler von 1 bis 6	
	Kapital = Kapital + RateProJahr
	Zinsen = Kapital * Zinssatz
	Kapital = Kapital + Zinsen

Auch Termwertberechnungen, wie sie im Mathematikunterricht durchgeführt werden sind in diesem Zusammenhang ein Beispiel für die Wiederholungsstruktur. Hier kann man auch überlegen, ob es auf einem Tabellenblatt nicht günstiger ist die Wertetabelle senkrecht anzulegen. Auch ist dieses Beispiel gut geeignet, eine grafische Darstellung der Termwerte hinzuzufügen.

	A	B	C	D	E	F	G	H	I	J
1	Termwertberechnung									
2										
3			T(x)	=	0,5	* x +	1,5			
4										
5			Startwert		-6		Schrittweite			1
6										
7										
8	x	-6	-5	-4	-3	-2	-1	0	1	2
9	T(x)	-1,5	-1	-0,5	0	0,5	1	1,5	2	2,5
10										

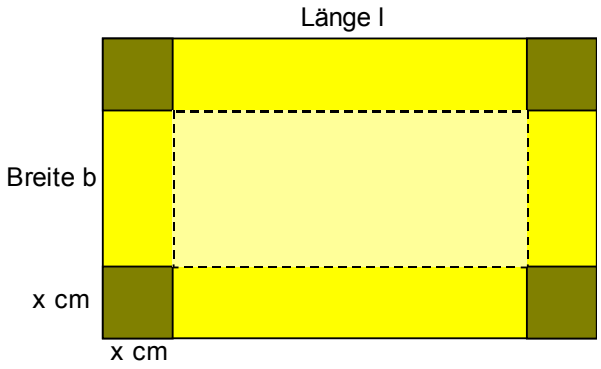


Noch eine etwas anspruchsvollere Aufgabe: Aus einem rechteckigen Karton wird eine Schachtel gefaltet (vgl. Abb.). Mit Hilfe einer Wertetabelle soll herausgefunden werden, bei welchem ganzzahligen Wert für die Schachtelhöhe (x cm) der Karton mit dem größtmöglichen Volumen entsteht.

Zunächst überlegen wir uns den Algorithmus:

$$V = (l - 2x)(b - 2x)x$$

Außerdem muss x kleiner sein als die halbe Breite des Kartons (vorausgesetzt, die Breite ist kleiner als die Länge!). Nun bereiten wir eine Tabelle wie abgebildet vor und überlegen uns die notwendigen Formeln.



	A	B	C	D
1	**Schachteln falten**			
2				
3	Kartonmaße:	Länge l:	40	cm
4		Breite b:	30	cm
5				
6	Volumengrößte Schachtel:		3024	cm³
7				
8	**Schachtelhöhe in cm**	**Volumen in cm³**		
9	0	0		
10	1	1064		
11	2	1872		
12	3	2448		
13	4	2816		
14	5	3000		
15	6	**3024**		
16	7	2912		
17	8	2688		
18	9	2376		
19	10	2000		
20	11	1584		
21	12	1152		
22	13	728		
23	14	336		
24	15	0		
25				

Dass es sich um eine Modellierung bzw. Simulation handelt, sei hier nur nebenbei erwähnt. Auch der weiter oben angesprochene Getränkeautomat lässt sich mit einem Tabellenkalkulationsprogramm umsetzen (siehe Kapitel 'Modellierung').

Graphisches Darstellen von Tabellen

Auf Rechenblättern einer Tabellenkalkulation lassen sich Daten übersichtlich anordnen. Eine noch anschaulichere Form der Datendarstellung lässt sich durch Diagramme realisieren. Hierfür gibt es so genannte Präsentationsgrafikprogramme bzw. Geschäftsgrafikprogramme. Sie sind in Tabellenkalkulationsprogrammen meist schon integriert, sodass aus den vorliegenden Daten direkt die zugehörigen Diagramme erstellt werden können.

Bei ihrer Erstellung hilft der Diagrammassistent. Am einfachsten erstellt man zunächst die Tabelle mit den Zahlenwerten, markiert den entsprechenden Bereich und ruft den Diagrammassistenten durch einen Mausklick auf das entsprechende Symbol auf. Die nun folgende Vorgehensweise ergibt sich durch den Assistenten weitgehend von selbst.

Die verschiedenen Diagrammarten lassen sich in folgende Klassen einteilen:

- **Säulendiagramme** eignen sich für die Darstellung von Mengenzuordnungen einer überschaubaren Anzahl von Wertepaaren, die stufenweise oder in zeitlicher Abhängigkeit erfolgen. Als unabhängige Größe wählt man im allgemeinen die waagerechte Achse, während die abhängige Größe senkrecht abgetragen wird. Es existieren bei der Darstellung keine Zwischengrößen. Der Aussageschwerpunkt dieser Diagramme liegt im **Vergleich**.

  Anwendungsbeispiel: Darstellung der monatlichen Niederschläge, Mengenvergleiche

- Das **Balkendiagramm** ist ein um 90° gedrehtes Säulendiagramm.

  Anwendungsbeispiel: wie Säulendiagramm

- Für die Darstellung kontinuierlich ablaufender Sachverhalte verwendet man **Liniendiagramme**. Bei Veränderung von Größen im Zeitablauf existieren bei dieser Darstellung auch Zwischengrößen. Sie dienen auch der **Darstellung von Änderungen und Trends**.

  Anwendungsbeispiel: Darstellung der monatlichen Durchschnittstemperatur oder (zeitlicher) Verlauf einer Messgröße.

- Füllt man die Fläche unter der Linie aus, erhält man ein **Flächendiagramm**. Gebräuchlich sind auch Kombinationen verschiedener Diagrammarten zum **Vergleichen** unterschiedlicher Sachverhalte.

  Anwendungsbeispiel: ähnlich wie Liniendiagramm

- Sollen gemessene Werte dargestellt werden, eignet sich zur Darstellung besonders gut ein **Punktdiagramm**. Die Diagrammart wird vor allem bei der **Darstellung von Zahlenpaaren** bevorzugt.

  Anwendungsbeispiel: ähnlich wie Liniendiagramm

- Will man die absolute oder prozentuale Verteilung vorgegebener es Größen darstellen, verwendet man ein **Kreisdiagramm**, das die **Teile des Ganzen** anschaulich wiedergibt. Manche Tabellenkalkulationsprogramme geben die Prozentwerte automatisch mit aus.

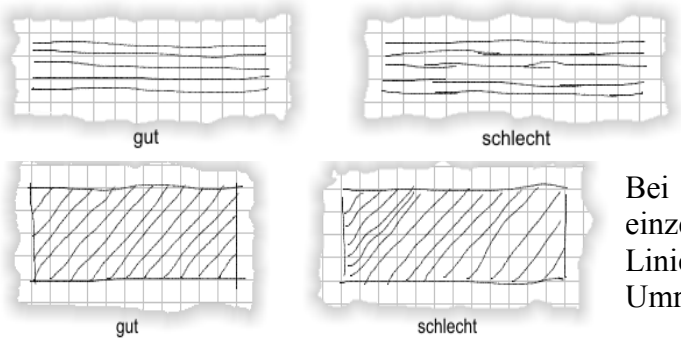
  Anwendungsbeispiel: Stimmenverhältnisse bei einer Wahl oder Anteile an einem Ganzen.

Geometrisches Konstruieren

Grundlagen des Geometrischen Zeichnens

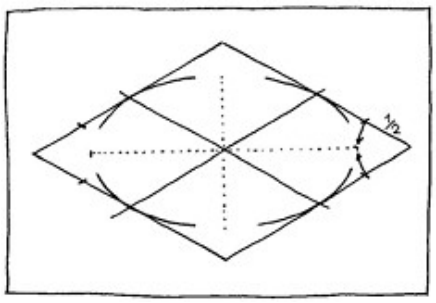
(Hinweis: Viele Abbildungen und Hinweise sind den Fortbildungsunterlagen von RSL Otmar Wagner, Realschule Kronach I mit dessen Einverständnis entnommen!)

Von Beginn an ist die **Freihandskizze** ein durchgängiges Unterrichtsprinzip in diesem Teil des IT-Unterrichts und muss natürlich besonders zu Beginn intensiv geübt werden. Dazu eignen sich besonders Bleistifte bzw. Feinminenstifte auf Blanko- oder Karopapier. Insbesondere ist am Anfang Wert auf Parallelität und ein Gefühl für Abständen zu legen. Eine erste Übung könnte so aussehen:

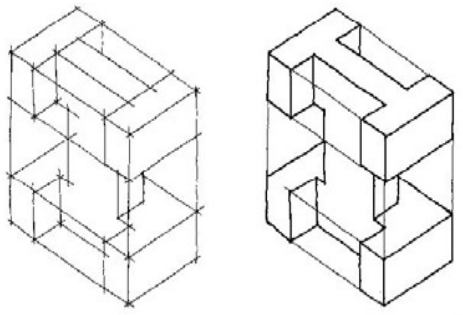


Wichtig ist es, dass die Striche locker und möglichst ohne abzusetzen durchgezogen werden. Parallele Linien sollten auch parallel bleiben.

Bei Schraffuren sollten die Abstände der einzelnen Linien möglichst gleich groß sein, die Linien sollten darüber hinaus bis zu den Umrisslinien gehen.



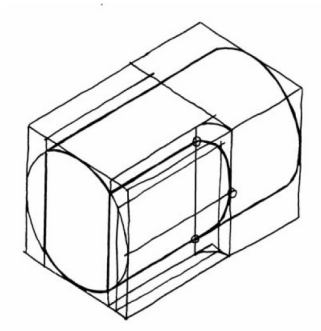
Da auch Rotationskörper einen wichtigen Baustein bilden, sollten auch hier schon Kreise und besonders Ellipsen in die Freihandzeichnungen einbezogen werden.



Auch das Gespür für dreidimensionale Abbildungen ist zu üben.

Dabei ist folgende Vorgehensweise empfehlenswert:

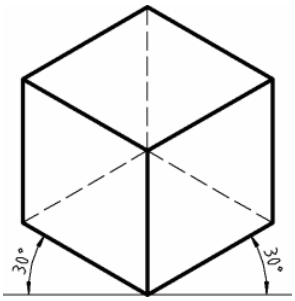
1. Hüllform dünn vorzeichnen
2. Bei lockerer Stifthaltung Linien durchziehen
3. Proportionen abschätzen
4. Linien sind parallel, waagrecht, senkrecht, mittig, usw.
5. Markante Punkte setzen
6. Mittellinien, Hilfslinien, Kanten und Kurven zeichnen
7. Ergebnislinien nachziehen



Axonometrische Projektionen

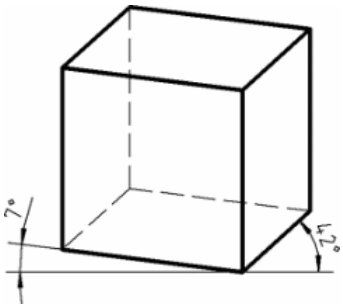
Unter Perspektive versteht man die Darstellung räumlicher Gebilde in einer ebenen Zeichenfläche. Zwei Parallelperspektiven sind genormt:

Die **isometrische Axonometrie**
(isometrisches Raumbild)



Seitenverhältnis 1 : 1

die **dimetrische Axonometrie**
(dimetrisches Raumbild)



Seitenverhältnis 1 : 2

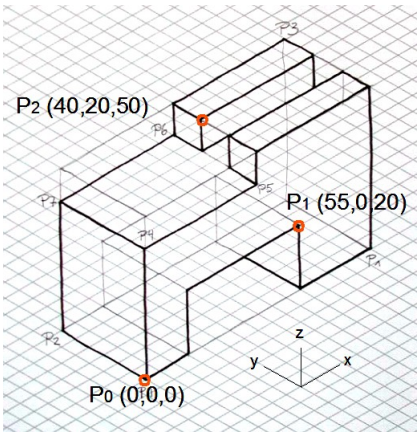
Isometrische Rasterskizze

Neben diesen Freihandskizzen bilden isometrische Raumbildskizzen auf Rasterpapier (mit Rasterpunkten oder Rasterlinien) einen fundamentalen Baustein im Einüben geometrischer Konstruktionen. Entsprechende Raumbilder können so auch mit einem CAD-Programm wie Solid Edge, MegaCAD oder AutoCad sofort parallel mit erstellt werden.

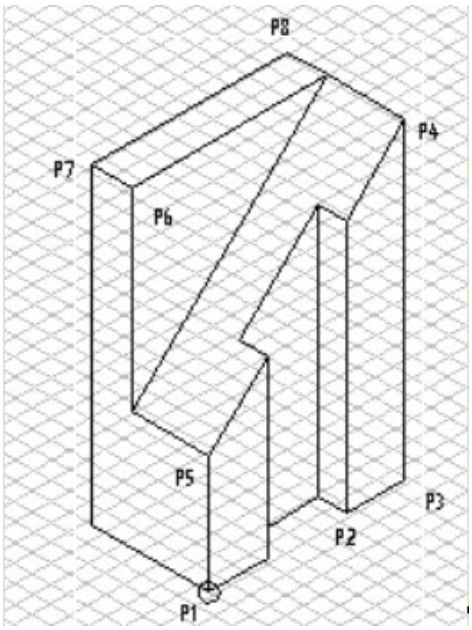
In die Rasterskizze werden - wie in der Abbildung dargestellt - markante Punkte eingetragen. Eine weitere Übung besteht nun darin, die Koordinaten dieser Punkte in eine Tabelle zu übertragen. Auch umgekehrt können Punkte aus vorgegebenen Koordinaten in die Zeichnung übertragen werden.

Auch hierzu die Vorgehensweise:

1. Iso-Rasterpapier verwenden (bzw. unterlegen)
2.
3. Anfangspunkt (Ursprung, hier P(0, 0, 0)) festlegen
   Grundkörper als Hüllform zeichnen
4. Achsen für Ellipsen anlegen
5. Markante Punkte setzen
6. Mittellinien, Hilfslinien, Kanten und Kurven zeichnen
7. Ergebnislinien nachziehen
8. Koordinatenwerte bestimmen bzw. eintragen



Ein weiteres Beispiele für eine isometrische Rasterskizze soll als Anregung dienen:



Wie in den Skizzen sichtbar sind die Grundkörper durch verschiedene Veränderungen bearbeitet worden. In diesem Zusammenhang sind die entsprechenden Fachbegriffe wichtig.


Schräge


Nut


Durchbruch


Ausklinkung

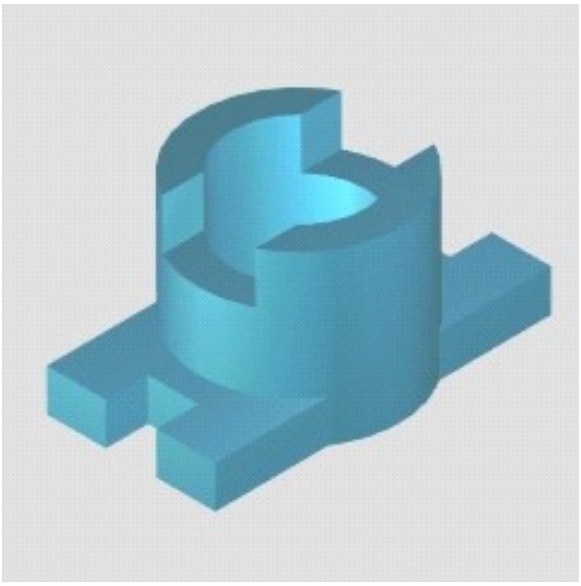

Bohrung


Abflachung

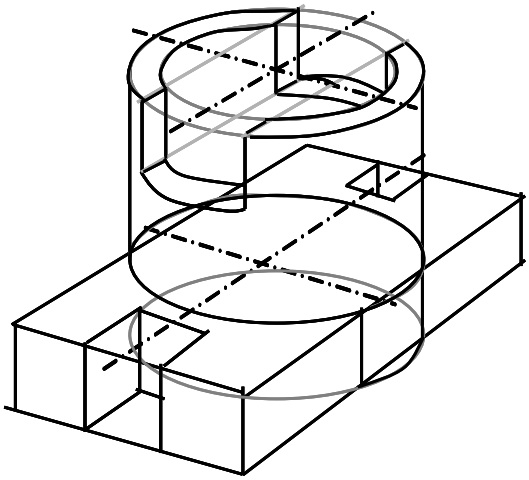

Freistich

Als Beispiel ist nachfolgend in Werkstück aufgeführt, dass als Freihandskizze und als isometrische Rasterskizze dargestellt wird.

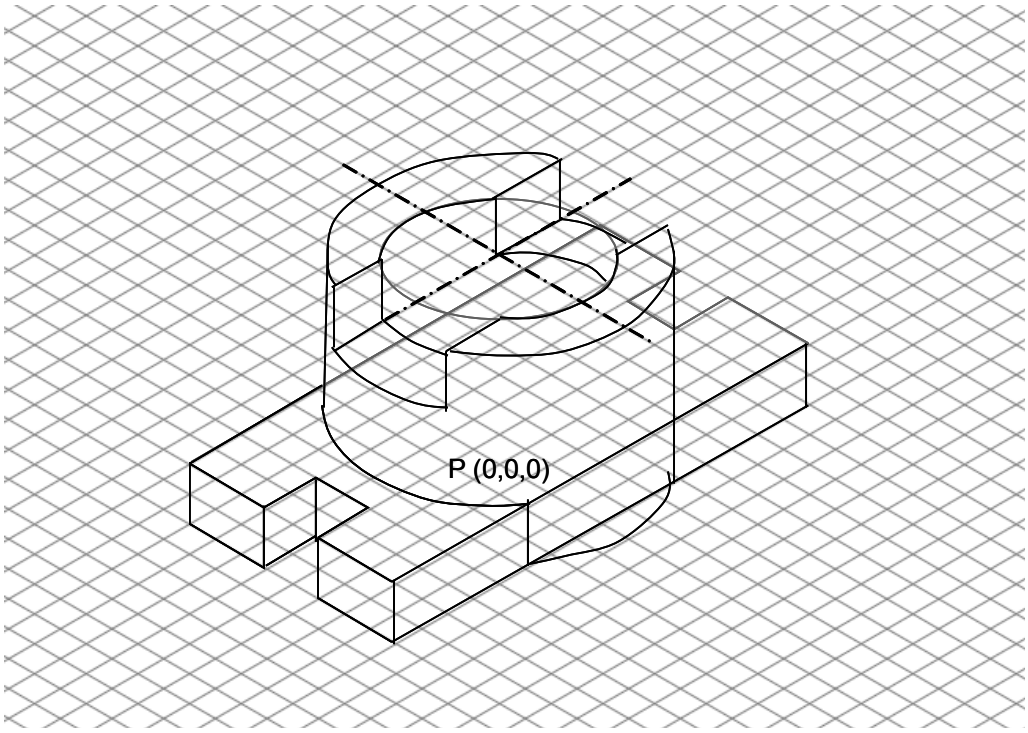
Werkstück



Freihandskizze



Isometrische Rasterskizze

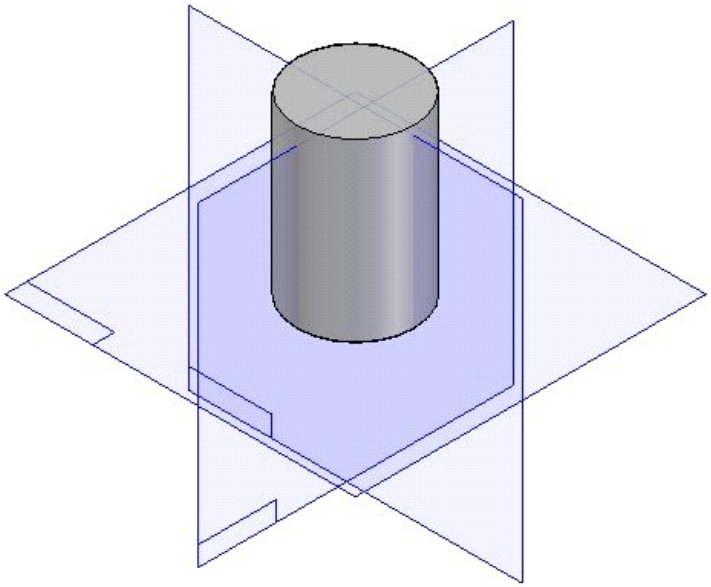


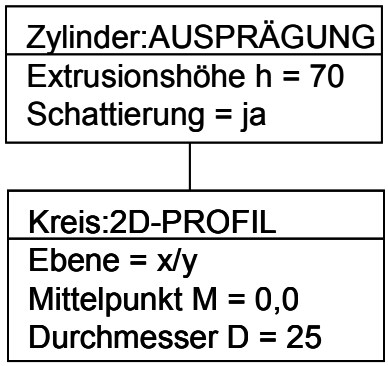
Grundlagen des Computer Aided Design – CAD

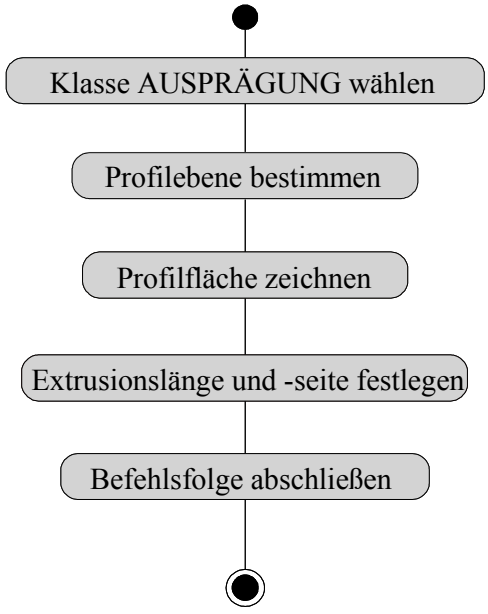
Ausgehend von Freihandskizzen setzen die Schüler ein 3D-CAD-System ein und erkennen dabei Klassen, Attribute und Methoden. Sie nehmen einfache Veränderungen an 3D-Modellen vor. Sie leiten 2D-Ansichten zur Wiedergabe technischer Informationen ab.

Die Grundkonstruktionen sind im folgenden nach aufsteigender Schwierigkeit und Komplexität unter Verwendung des Programms 'Solid Edge' dargestellt. Dabei ist es immer wichtig, dass auch eine textuelle Beschreibung des Werkstücks vorangeht. Das Erstellen von Klassendiagrammen ist insbesondere bei komplexeren Werkstücke eine sinnvolle Vorarbeit. Das Aktivitätsdiagramm macht den Arbeitsablauf deutlich.

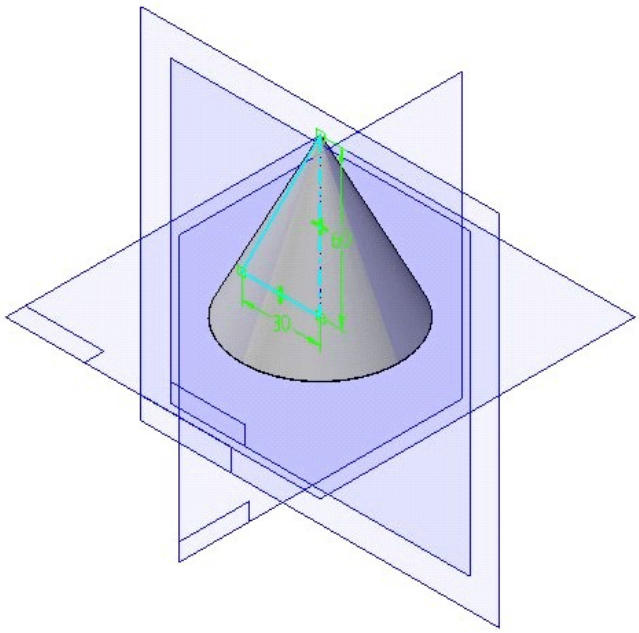
Gerade Extrusion

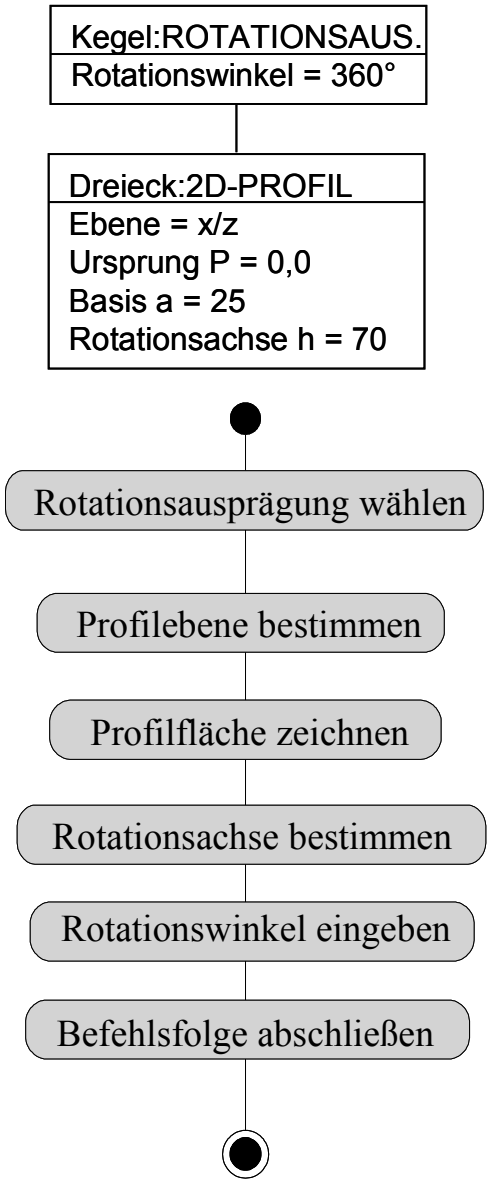




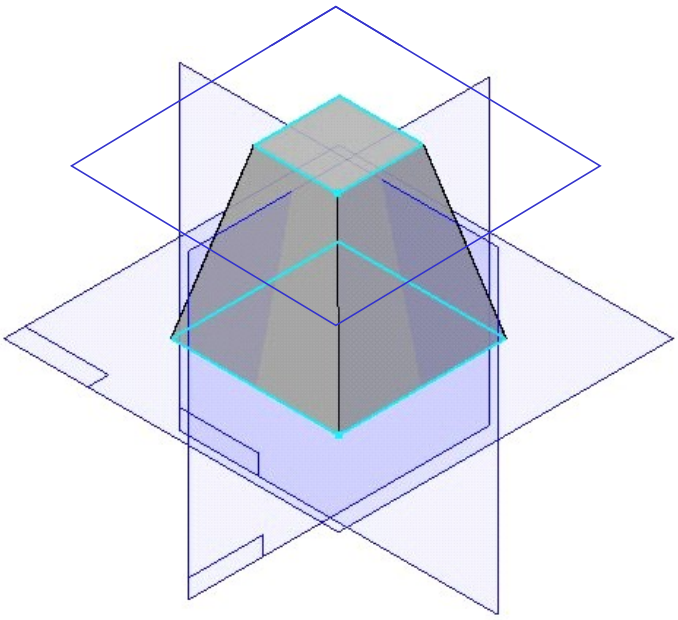


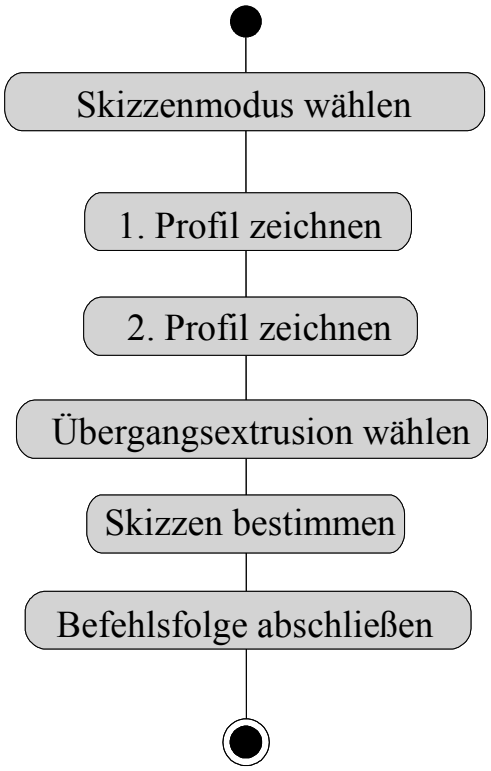
Rotationsausprägung

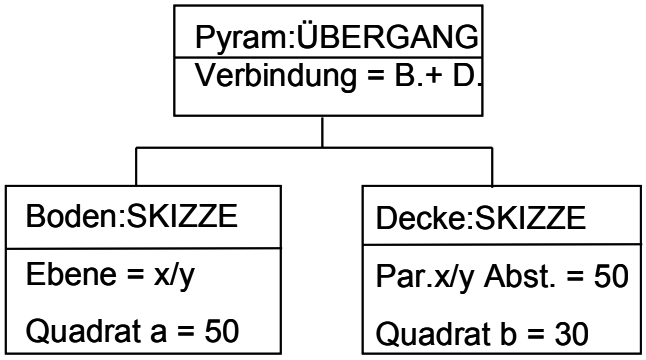




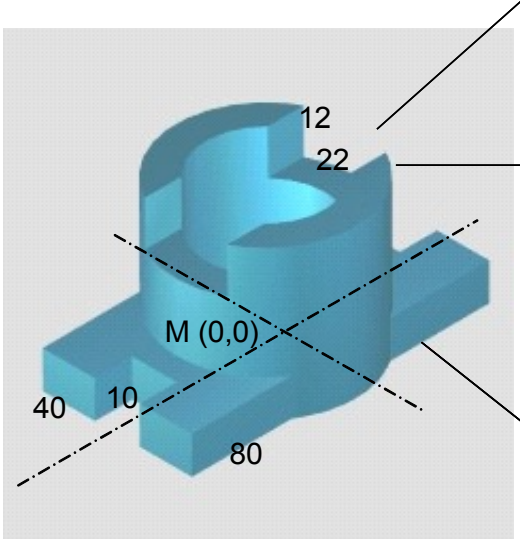
Übergangsausprägung







Komplexes Werkstück
(Objekte, Attribute, Attributwerte)



Nut:AUSSCHNITT
Extrusion = über alles

Nut:2DPROFIL
Ebene = x/z-Ebene
P_1 = -11, 12
Linienzug = P_1 … P_4

3. Objekt

Hohlzylinder:AUSPRÄG.
Extrusionshöhe = 40

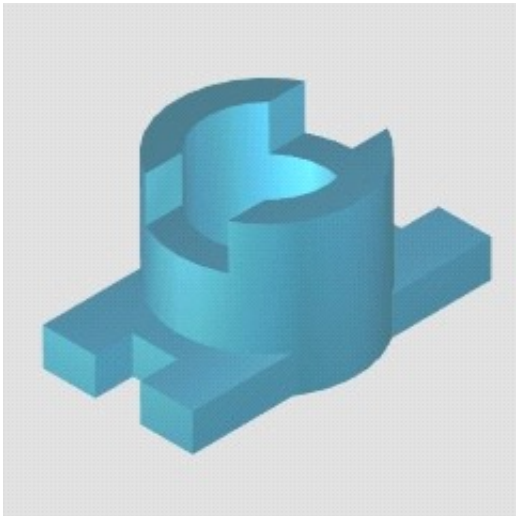
K1+K2:2DPROFIL
Ebene = x/y-Ebene
Profil = K1 d 48; K2 d 32
Lage = Mittelpunkt 0,0

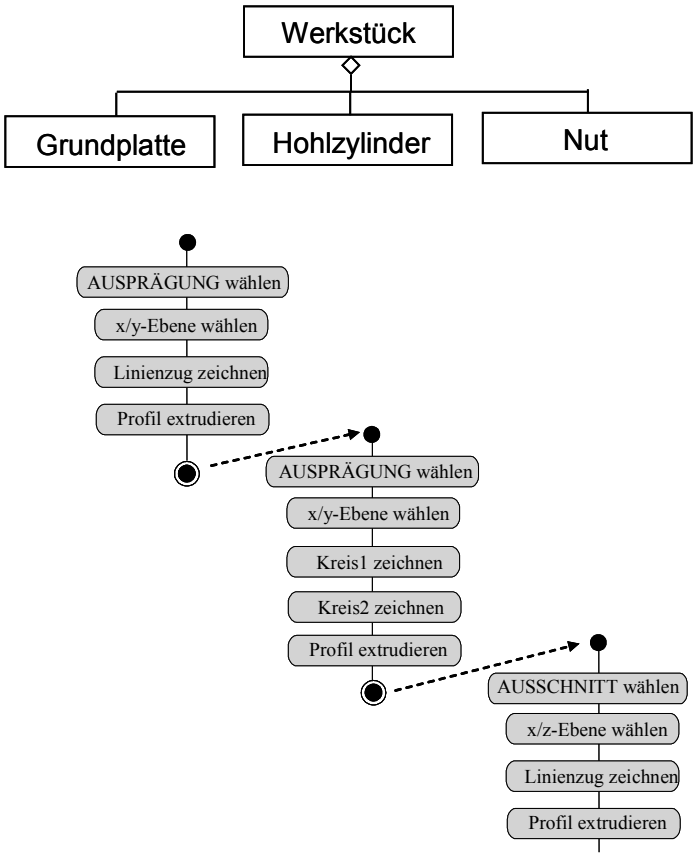
2. Objekt

Grundplatte:AUSPRÄG.
Extrusionshöhe = 10

Grundfläche:2DPROFIL
Ebene = x/y-Ebene
Profil = Linienzug P_1…P_n
Lage P_1 = -40, -20

1. Objekt





Ableitung einer 2D-Zeichnung