

# **Einführung in die Programmierung**

**Ein Handbuch zum Unterricht**

**Steffen Wenck**



## Inhaltsverzeichnis

	Seite
<b>1 Einführung in die Programmierung .....</b>	<b>5</b>
1.1 Begriffe .....	5
1.2 Darstellung der Syntax .....	5
1.2.1 Backus Naur Form .....	5
1.2.1.1 Einfache Backus Naur Form .....	5
1.2.1.2 Erweiterte Backus Naur Form .....	5
1.2.1.3 Modifizierte Erweiterte Backus Naur Form .....	6
1.2.2 Syntaxdiagramm .....	7
<b>2 Einteilung der Programmiersprachen .....</b>	<b>13</b>
2.1 Programmierparadigma .....	13
2.1.1 Imperative Programmierung .....	13
2.1.1.1 Anfangs-Programmierung .....	13
2.1.1.2 Strukturierte Programmierung .....	13
2.1.1.3 Prozedurale Programmierung .....	13
2.1.1.4 Modulare Programmierung .....	14
2.1.1.5 Objektorientierten Programmierung .....	14
2.1.2 Deklarative Programmierung .....	15
2.1.2.1 Funktionale Programmierung .....	15
2.1.2.2 Logische Programmierung .....	15
2.1.2.3 Mengen-orientierte Programmierung .....	15
2.2 Einteilung nach Generationen .....	15
2.3 Einteilung zum Zeitpunkt der Maschinencode-Erstellung .....	17
<b>3 Programmspezifikation .....</b>	<b>21</b>
3.1 Klassische Modelle .....	21
3.1.1 Entscheidungstabelle .....	21
3.1.2 Pseudocode .....	23
3.1.3 Programmablaufplan .....	24
3.1.4 Struktogramm .....	25
3.2 Unified Modeling Language .....	27
3.2.1 Klassendiagramm .....	27
3.2.1.1 Diagrammelemente .....	28
3.2.1.2 Beispiel .....	29



# 1 Einführung in die Programmierung

## 1.1 Begriffe

Programmieren: Anweisungen erteilen

Programmiersprache: Sprache zum Abfassen von Anweisungen

Syntax: System von Regeln

Semantik: Bedeutung der Regeln

Beispiel:     a           +           b           \*           c

Syntax:     Bezeichner Operator Bezeichner Operator     Bezeichner

Semantik:   Bilde die Summe aus a und dem Produkt aus b und c.

[inf-schule | Einführung - Sprache als Zeichensystem » Syntax, Semantik, Pragmatik](#)

## 1.2 Darstellung der Syntax

Zur Darstellung der Syntax wird die Backus Naur Form oder das Syntaxdiagramm verwendet.

### 1.2.1 Backus Naur Form

John Backus und Peter Naur sind zwei Informatiker.

#### 1.2.1.1 Einfache Backus Naur Form

Hier werden nur Terminal- und Nicht-Terminal-Symbole verwendet.

Symbol	Verwendung	Bedeutung
	Terminalsymbol	Darstellbare Zeichen, die auf dem Bildschirm ausgegeben werden.
< Name >	Nicht Terminal Symbole	Dienen zur Begriffserklärung.
::=	Definition	Dient zur Definition von Nicht Terminal Symbolen.
	Alternative	Entweder ... oder ...

#### 1.2.1.2 Erweiterte Backus Naur Form

Symbol	Verwendung	Bedeutung
=	Definition	Dient zur Definition von Nichtterminalsymbolen
,	Aufzählung	Dient zur Darstellung von Sequenzen
;	Endezeichen	Ende der Begriffserklärung
	Alternative	Entweder ... oder ...
[ ]	Option	Der Inhalt der eckigen Klammer wird 0 oder 1-mal verwendet.
{ }	Optionale Wiederholung	Der Inhalt der geschweiften Klammer wird 0, 1 oder N-mal verwendet.

( )	Gruppierung	Angabe einer Gruppe von Begriffen
'...'	Terminalsymbole	Dient zur Darstellung von Terminalsymbolen
"..."	Terminalsymbole	Dient zur Darstellung von Terminalsymbolen
(* ... *)	Kommentar	
? ... ?	Spezielle Sequenzen	Spezielle Sequenz wird als abkürzende Schreibweise für viele Zeichen verwendet
-	Ausnahme	Das nachfolgende Zeichen darf nicht verwendet werden
—	Default-Wert	Standardwert

### 1.2.1.3 Modifizierte Erweiterte Backus Naur Form

Beispiel: Wie ist eine Zuweisung (basierend auf C++) definiert (hier nicht vollständig!)?

#### MEBNF

(\* Beispiel für eine Zuweisung \*)

Zuweisung ::= Bezeichner = Zahl | Bezeichner | Text ;  
 Bezeichner ::= Buchstabe { Buchstabe | Ziffer }  
 Buchstabe ::= Kleinbuchstabe | Großbuchstabe  
 Kleinbuchstabe ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q |  
                   r | s | t | u | v | w | x | y | z  
 Großbuchstabe ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |  
                   R | S | T | U | V | W | X | Y | Z  
  
 Zahl ::= [ + | - ] Zahl\_einstellig | Zahl\_mehrstellig  
 Zahl\_einstellig ::= Ziffer  
 Zahl\_mehrstellig ::= Ziffer\_ohne\_0 { Ziffer }  
 Ziffer\_ohne\_0 ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
 Ziffer ::= 0 | Ziffer\_ohne\_0  
  
 Text ::= " { AlleZeichen – } "  
 AlleZeichen ::= ? alle sichtbaren Zeichen ?

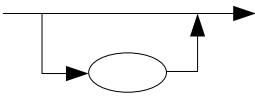
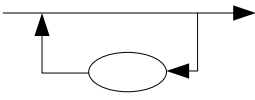
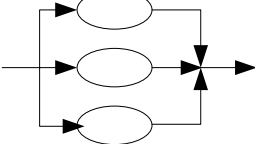
Syntaktisch korrekte Anweisung:

```
a = 5 ;
s = "Danke";
i = a;
e = - 514;
b1 = + 2 ;
```

Nicht syntaktisch korrekte Anweisung:

```
s = ; a
1n = 4;
s = "Dan"ke" ;
```

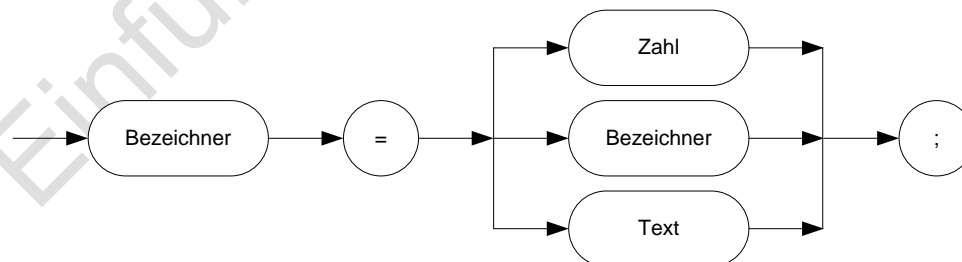
## 1.2.2 Syntaxdiagramm

Symbol	Verwendung	Bedeutung
Name ::=	Name	Jedes Syntaxdiagramm beginnt mit einem Namen.
○	Terminalsymbol	Angabe des sichtbaren Zeichens
⬭	Nichtterminalsymbol	Angabe des zu erklärenden Begriffes
□	Schlüsselwort	Schlüsselwort
→	Konkatenation	Übergang von einem zum nächsten Knoten
⬭	Knoten	Terminal- oder Nichtterminal-Symbol (Achtung: Ist kein Symbol des Syntaxdiagrammes!)
	Option	Ein Knoten kann (muss aber nicht durchlaufen werden).
	Optionale Wiederholung	Der Knoten wird 0, 1, 2 oder N-mal durchlaufen.
	Alternative	Genau ein Knoten wird durchlaufen.
...	Spezielle Sequenz	Abkürzende Schreibweise für viele Zeichen

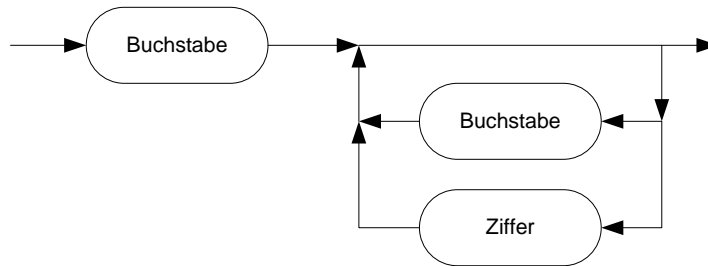
Beispiel: Wie ist eine Zuweisung (basierend auf C++) definiert (hier nicht vollständig)?

### Syntaxdiagramm

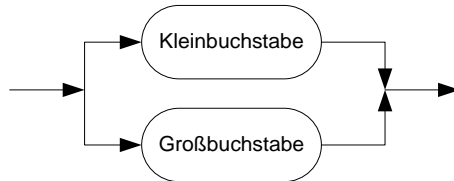
Zuweisung ::=



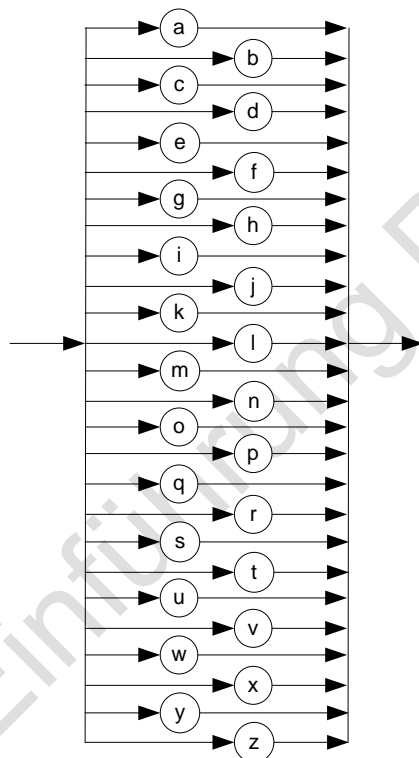
Bezeichner ::=



Buchstabe ::=

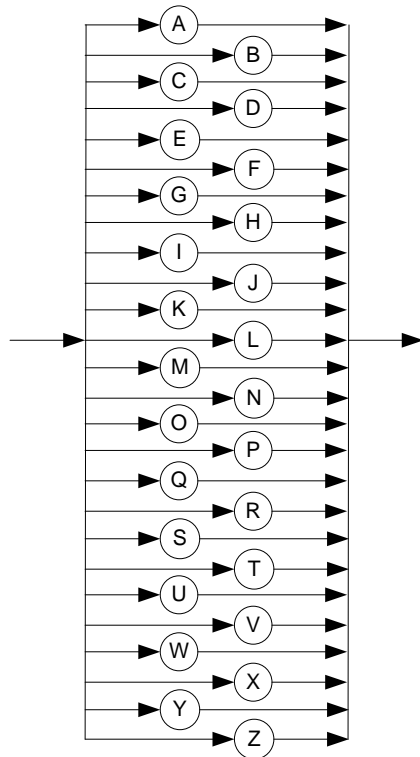


Kleinbuchstabe ::=

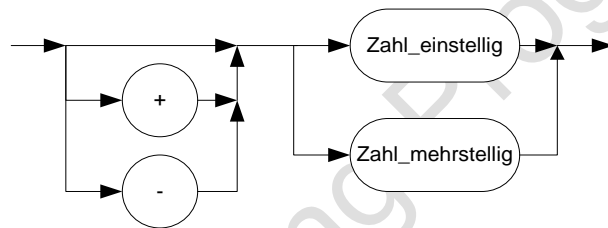




Großbuchstabe ::=



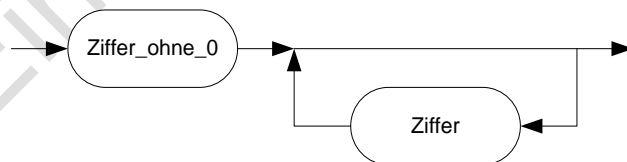
Zahl ::=



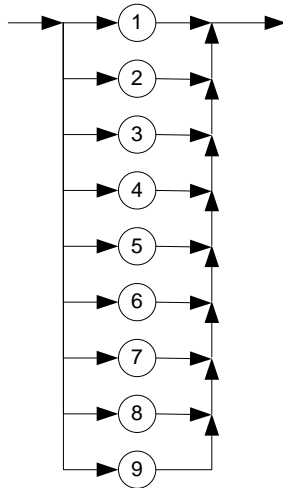
Zahl\_einstellig ::=



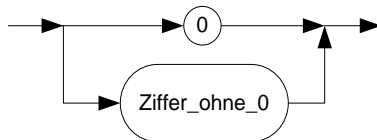
Zahl\_mehrstellig ::=



Ziffer\_ohne\_0 ::=



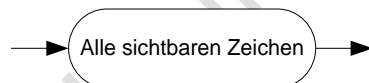
Ziffer ::=



Text ::=



AlleZeichen ::=



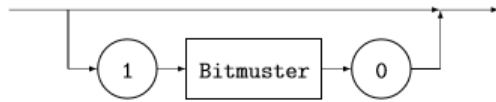
Programm:

Drawio: draw.io

EBNF-Visualizer: <http://dotnet.jku.at/applications/Visualizer/#Simple>

Beispiel:

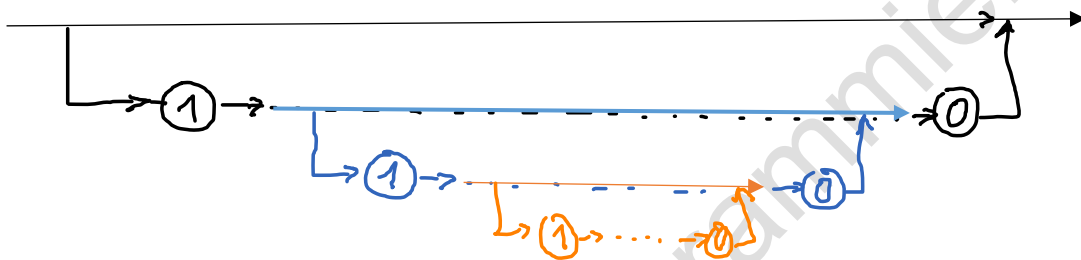
Bitmuster ::=



Welche der folgenden sechs gegebenen Bitmuster sind korrekt bzw. fehlerhaft?

Beispiel	korrekt	fehlerhaft
10	<input checked="" type="checkbox"/>	<input type="checkbox"/>
01	<input type="checkbox"/>	<input checked="" type="checkbox"/>
100	<input type="checkbox"/>	<input checked="" type="checkbox"/>
1100	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<leeres Bitmuster>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
111100000	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Bitmuster ::=





## 2 Einteilung der Programmiersprachen

Programmiersprachen können unterteilt werden basierend auf dem Programmierparadigma, Einteilung nach den Generationen oder nach dem Zeitpunkt der Maschinencode-Erstellung.

### 2.1 Programmierparadigma

Programmierparadigma: ist ein fundamentaler Programmierstil, der zum Erstellen von Quellcode nach bestimmten Regeln dient.

#### 2.1.1 Imperative Programmierung

Unter imperativer Programmierung wird die lineare Abfolge von Anweisungen verstanden.

Im Vordergrund steht das WIE der Abarbeitung.

WIE komme ich zum Ziel?

##### 2.1.1.1 Anfangs-Programmierung

Programmierung mit Variablen und dem Befehl GOTO.

Variablen: dient zur Datenhaltung

GOTO: Gehe zu einer Adresse (Beispiel: zum Programmieren von Schleifen, Verzweigung)

##### 2.1.1.2 Strukturierte Programmierung

Weiterentwicklung der Anfangs-Programmierung.

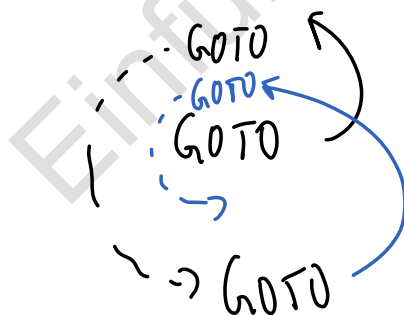
Ziel: Die Sprunganweisung GOTO nicht zu verwenden.

GOTO ist eine Sprunganweisung. Damit wurde früher Schleifen programmiert.

GOTO wurde durch Schleifen- und Verzweigungs-Konstrukte ersetzt. Diese Konstrukte sind nicht so fehleranfällig wie GOTO.

Programmiersprache: PASCAL

Spagetti-Code:

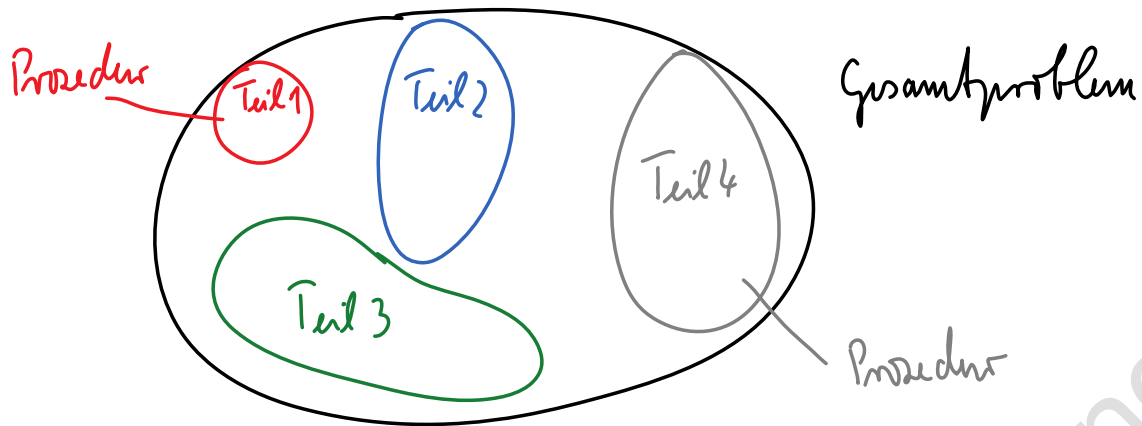


##### 2.1.1.3 Prozedurale Programmierung

Weiterentwicklung der strukturierten Programmierung.

Programm in einzelne Prozeduren unterteilen, d.h. ein großes Problem in Teilprobleme zu unterteilen.

Programmsprachen: PASCAL und C



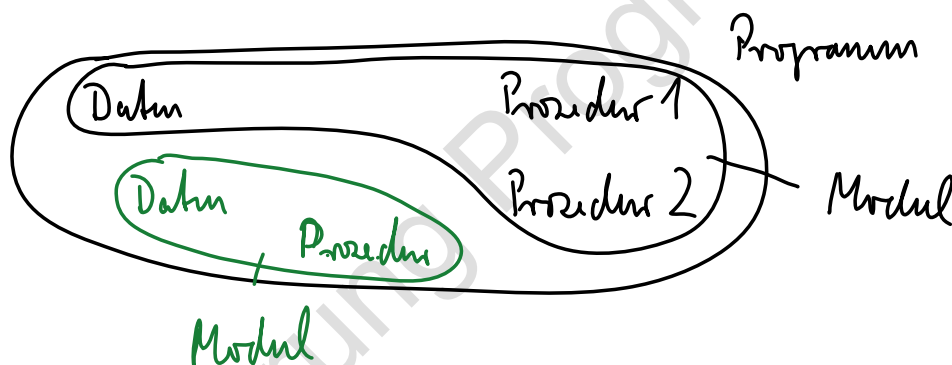
Die Lösung von jedem Teilproblem führt zur Lösung des Gesamtproblems.

#### 2.1.1.4 Modulare Programmierung

Weiterentwicklung der prozeduralen Programmierung.

Die Daten (Datenhaltung) werden zusammen mit den Prozeduren (Datenverarbeitung) gemeinsam in einem Modul gespeichert.

Programmiersprachen: Modula-2, PL/SQL



#### 2.1.1.5 Objektorientierten Programmierung

Weiterentwicklung der modularen Programmierung.

Programmiersprachen: Java, C++, Delphi

**Klasse:** ist selbstdefinierter Datentyp, der die Datenhaltung (Klassenvariable) und/oder die Datenverarbeitung (Methode) umfasst

**Objekt:** ist eine Variable (Instanz) vom Datentyp der Klasse

**Klassenvariable:** ist eine Variable innerhalb der Klasse

**Methode:** ist ein Unterprogramm innerhalb der Klasse

**Member:** ist eine Klassenvariable oder Methode

Umgangssprachlich: Klasse ist ein Bauplan für ein Objekt.

Eckpfeiler der objektorientierten Programmierung:

- Vererbung: Die Kindklasse erbt die Eigenschaften der Elternklasse (soweit erlaubt.)
- Polymorphie: Eine Methode ist polymorph, wenn sie in verschiedenen Klassen (Vererbung) die gleiche Signatur besitzt, jedoch erneut implementiert wird.
- Kapselung: ist das Verbergen von Mitgliedern vor dem Zugriff von außen  
Syntax: public | protected | private

## 2.1.2 Deklarative Programmierung

Im Vordergrund steht nicht das WIE der Abarbeitung, sondern WAS ist das Ziel.

## 2.1.2.1 Funktionale Programmierung

Das Programm ist eine Funktion, die sich typischer Weise auf einfachen Funktionen stützt. Eine Funktion kann weitere Funktionen aufrufen.

Programmiersprache: LISP

## 2.1.2.2 Logische Programmierung

Die logische Programmierung basiert auf der mathematischen Logik.

Nutzer geben eine Menge von Fakten, Regeln und Abfragen ein und erhalten als Ergebnis: Ja oder Nein.

Programmiersprache: PROLOG

## 2.1.2.3 Mengen-orientierte Programmierung

Dient zur Verarbeitung sehr großer Datenmengen.

Programmiersprache: SQL

## 2.2 Einteilung nach Generationen

1. Generation (Maschinensprache)	
Beschreibung	<p>Der Computer kann nur Binär- oder Hexadezimalzahlen verarbeiten.</p> <p>Maschinenbefehl 010 bedeutet Addition</p> <p>Maschinencode ist sehr schnell.</p> <p>Maschinencode ist sehr schwer zu lesen und zu verstehen.</p> <p>Maschinencode ist maschinenabhängig.</p>
Programmiersprache	
Beispiel:	<p><u>Addition von 3 + 4:</u></p> <p>010 00000011 00000100</p> <p>010            - Addition</p> <p>00000011    - 3 als Binärzahl</p> <p>00000100    - 4 als Binärzahl</p>
2. Generation (Assemblerprogrammierung)	

Beschreibung	Worte sind besser als Zahlen zu merken. ADD - Addition Der Assemblercode muss in Maschinencode überführt werden.
Programmiersprache	
Beispiel:	<u>Addition von 3 + 4:</u> PUSH rbp MOV rbp, rsp MOV DWORD ptr[rbp-4], 3 MOV DWORD ptr[rbp-8], 4 MOV eax, DWORD ptr[rbp-4] MOV edx, DWORD ptr[rbp-8] ADD eax edx
3. Generation (Problemorientierte Sprachen)	
Beschreibung	Kommen der menschlichen Sprache immer näher Leicht verständlich Unterstützen Algorithmen Werden für bestimmte Aufgabengebiete erzeugt
Programmiersprache	Naturwissenschaftliche Probleme: FORTRAN, C Kaufmännische Probleme: COBOL Sprache zum Erlernen einer Programmiersprache: BASIC
Beispiel:	<u>Addition von 3 + 4</u> 3 + 4
4. Generation (Datenorientierte Sprachen)	
Beschreibung:	Verarbeitung von Mengen an Daten
Programmiersprache	SQL
Beispiel:	<u>Lesen aller Kunden</u> SELECT * FROM Kunden;
5. Generation (Programmiersprachen der Künstlichen Intelligenz)	
Beschreibung	Beschreibung von Sachverhalten
Programmiersprache	PROLOG
Beispiel:	<u>Prolog-Textdatei:</u> Mann(Uwe). Mann(Gerd). Frau(Ina). Frau(Else).  <u>Eingabeaufforderung:</u> ?- Mann(Uwe). Yes. ?- Frau(Uwe). No.



## 2.3 Einteilung zum Zeitpunkt der Maschinencode-Erstellung

Was benötigt der Programmierer unbedingt zum Erstellen eines Programmes?

1. Editor: Schreiben des Quellcodes (Speicherung in einer Datei)
2. Compiler/Interpreter: Syntaxprüfung und Maschinencodeerstellung

Integrierte Entwicklungsumgebung (IDE):

- **Debugger:** erlaubt das schrittweise Abarbeiten eines Programmes
- Bibliotheken
- Syntax-Highlighting
- Autovervollständigung
- Formatierungen: Einrückungen
- Klassendiagramm
- ...

Compiler: Übersetzung des Quellcodes in Maschinencode erfolgt **vor dem Start** des Programmes

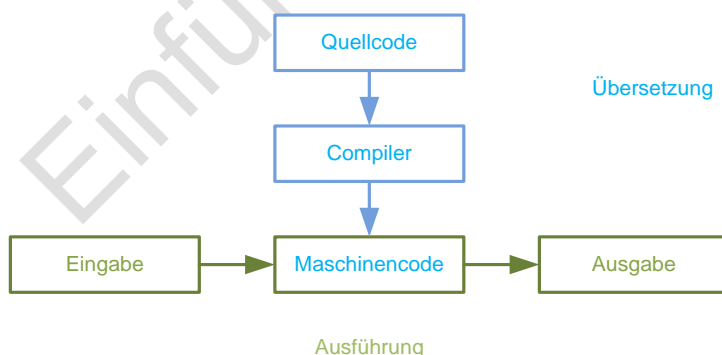
Vorteile: Überprüfung der Syntax, d.h. es können nur "fehlerfreie" Programme gestartet werden  
Schnellere Programmabarbeitung als bei Interpreter

Nachteile: plattformabhängig

Programmiersprachen: C, C++, Delphi, Rust

Vor dem Starten des Programmes werden alle Programmierbefehle in Maschinencode übersetzt, d.h. jede Anweisung wird gelesen und analysiert (Syntaxprüfung). Es kann eine Optimierung des Quellcodes vorgenommen werden. Die Übersetzung wird dann in eine ausführbare Datei gespeichert. Bei der Ausführung des Programms werden die Anweisungen "direkt" vom Prozessor verarbeitet.

Kompilierte Programme sind effizient und arbeiten sehr schnell, was sich gerade bei lang laufenden Programmen lohnt.



Arbeitsweise:

1. **Übersetzen** des Quellcodes in Maschinencode durch den Compiler (ausführbares Programm) und
2. **Ausführen** des Programms

Interpreter: Übersetzung des Quellcodes in Maschinencode erfolgt **während** der Programmabarbeitung

Vorteile: plattformunabhängig

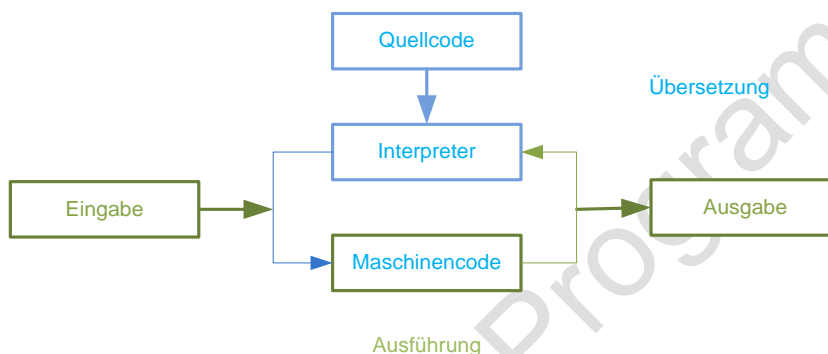
Nachteile: Langsamere Programmabarbeitung als bei Compiler

Programmiersprachen: Python, BASIC, Java Script, PHP, Java (alt)

Der Interpreter übersetzt den Quellcode in Maschinencode, aber erst zur Laufzeit des Programmes. Jede Anweisung wird einzeln gelesen, analysiert (Syntaxprüfung) und ausgeführt.

Wenn ein Fehler auftritt, dann wird eine Fehlermeldung ausgegeben und das Programm wird angehalten. Der Programmierer kann den Fehler beheben und das Programm an dieser Stelle fortsetzen.

Interpretersprachen sind ineffizient und langsam. So werden dieselben Programmteile wie bei Schleifen und Funktionen neu übersetzt. Dafür eignen sich solche Programm für plattformunabhängige Anwendungen.



### JIT-Compiler (Just in time)

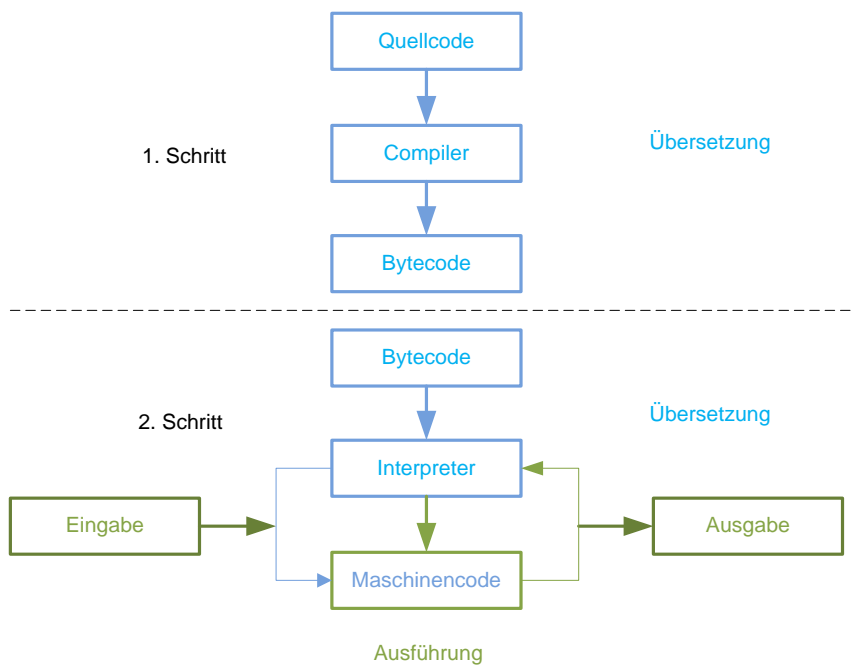
Verbindet die Vorzüge von Compiler mit Interpreter.

Programmiersprachen: Java (neu), C#, Perl

Um die Geschwindigkeit von Interpreter-Sprachen zu erhöhen, wurden Bytecode-Interpreter entwickelt.

Der Compiler übersetzt den Quellcode in Bytecode (Hardware unabhängige Anweisungen). Der Interpreter übersetzt den Bytecode zur Laufzeit in Maschinencode. Sie berücksichtigen die verschiedenen Spezifikationen zwischen einzelnen Prozessoren.

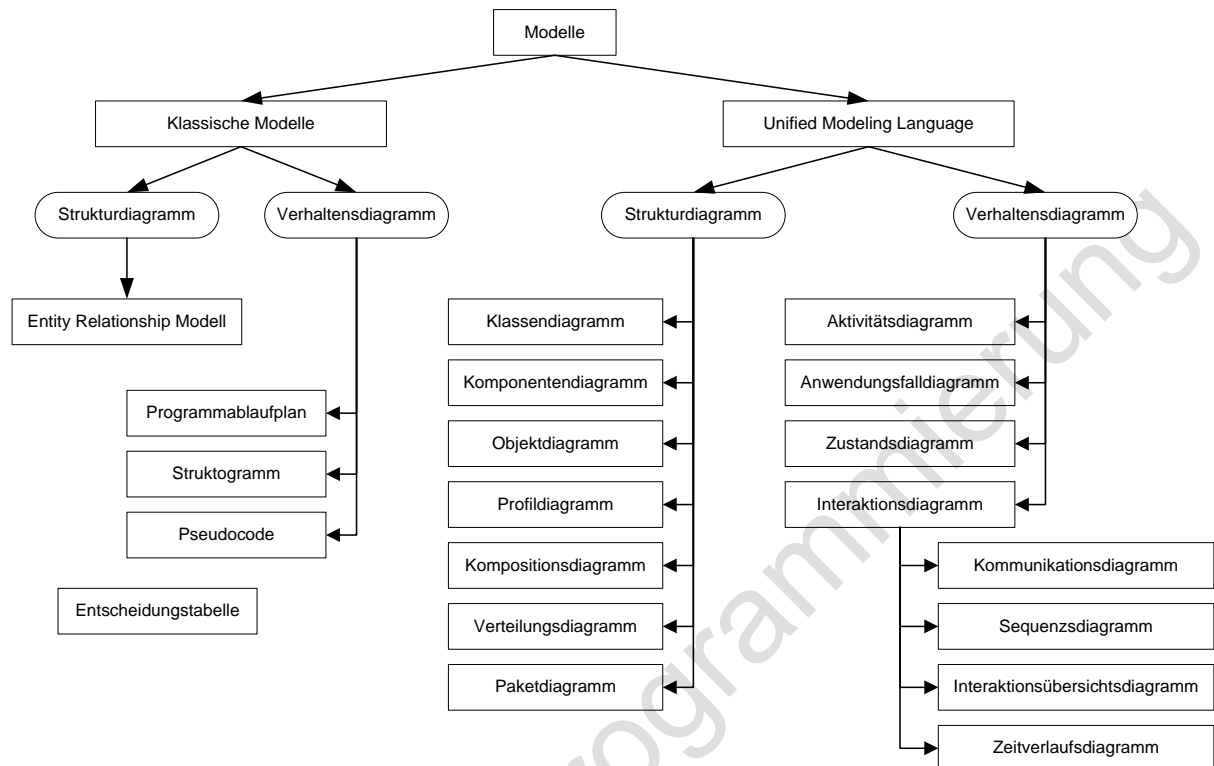
Die JIT-Compiler können den Bytecode schneller übersetzen als der Interpreter.





### 3 Programmspezifikation

Zweck: eine gemeinsame Modellierungssprache, um Programmieralgorithmen zu erklären



#### 3.1 Klassische Modelle

Aufgabe: Ermittle die GröÙte von drei unterschiedlichen Ganzen Zahlen: A, B und C.

Der Benutzer gibt zur Laufzeit des Programmes den Wert für A, B und C ein.

Grundidee: Vergleichen mit dem Operator > (alternativ <).

Beispiel:

Benutzer:	A = 4	A = 12	A = ?
	B = 6	B = -4	B = ?
	C = 3	C = 45	C = ?

Ausgabe: B = 6 ist der größte Wert. C = 45 ist der größte Wert.

##### 3.1.1 Entscheidungstabelle

Entscheidungstabellen gibt es seit 1957.

1979: DIN 66421

Entscheidungstabelle:

	Name der Entscheidungstabelle	Regelnummer
<b>WENN</b>	Bedingung	Bedingungsanzeiger
<b>DANN</b>	Aktionen	Aktionsanzeiger

Arbeitsschritte:

1. Bedingung festlegen
2. Aktionen festlegen
3. Bedingungsanzeiger setzen
4. Aktionsanzeiger setzen
5. Entscheidungstabelle konsolidieren

Entscheidungstabelle, um die größte ganze Zahl zu ermitteln.

1. Bedingungen festlegen

ET1: Größte Zahl ermitteln	
B1	$A > B$
B2	$B > C$
B3	$C > A$

Die Anzahl der Bedingungen ergibt sich aus der Aufgabenstellung.

2. Aktionen festlegen

ET1: Größte Zahl ermitteln	
B1	$A > B$
B2	$B > C$
B3	$C > A$
A1	A ist die größte Zahl.
A2	B ist die größte Zahl.
A3	C ist die größte Zahl.

Die Anzahl der Aktionen ergibt sich aus der Aufgabenstellung. Die Anzahl der Aktionen kann von der Anzahl der Bedingung unterschiedlich sein.

3. Bedingungsanzeiger setzen

ET1: Größte Zahl ermitteln		R1	R2	R3	R4	R5	R6	R7	R8
B1	$A > B$	J	J	J	J	N	N	N	N
B2	$B > C$	J	J	N	N	J	J	N	N
B3	$C > A$	J	N	J	N	J	N	J	N
A1	A ist größte Zahl								
A2	B ist größte Zahl								
A3	C ist größte Zahl								

Anzahl der Regeln: 2 <sup>Bedingungsanzahl</sup>

Angabe jeder möglichen Kombination aus Ja und Nein.

4. Aktionsanzeiger setzen

ET1: Größte Zahl ermitteln		R1	R2	R3	R4	R5	R6	R7	R8
B1	A > B	J	J	J	J	N	N	N	N
B2	B > C	J	J	N	N	J	J	N	N
B3	C > A	J	N	J	N	J	N	J	N
A1	A ist größte Zahl	-	X		X				-
A2	B ist größte Zahl	-				X	X		-
A3	C ist größte Zahl	-		X				X	-

3 Möglichkeiten zum Ausfüllen: X - Die Aktion ist zutreffend.

' ' - Die Aktion ist nichtzutreffend.

- - Die Aktion ist nicht möglich.

Es können durchaus mehrere Aktionszeiger gleichzeitig pro Regel gelten.

#### 5. Entscheidungstabelle konsolidieren

ET1: Größte Zahl ermitteln		R1	R2 (R24)	R3 (R37)	R4 (R56)	R5 (R8)
B1	A > B	J	J	-	N	N
B2	B > C	J	-	N	J	N
B3	C > A	J	N	J	-	N
A1	A ist größte Zahl	-	X			-
A2	B ist größte Zahl	-			X	-
A3	C ist größte Zahl	-		X		-

Das Konsolidieren ist nur möglich für gleiche Aktionsanzeiger und sie dürfen sich in den Wahrheitswerten der Bedingungen nur an einer Stelle unterscheiden.

#### 3.1.2 Pseudocode

Pseudocode ist eine textuelle, semiformale Darstellungsform für problemorientierte Programmiersprachen.

Es gibt keine Normierung!

Aufgabe:

Ermittle die Größte von 3 unterschiedlichen Ganzen Zahlen: A, B und C.

Die Eingabe der Werte erfolgt interaktiv.

Die Ausgabe der größten Zahl erfolgt auf dem Monitor.

Die Ermittlung der größten Zahl kann für andere Zahlen beliebig oft wiederholt werden.





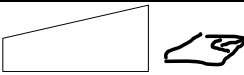
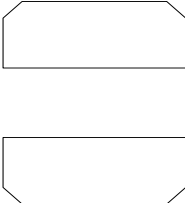

<p>Ganze Zahlen: A, B und C</p> <p>Ganze Zahl: GrößteZahl</p> <p>Solange</p> <p>Eingabe von A, B und C</p> <p>GrößteZahl = Ermitteln (A, B, C)</p> <p>Ausgabe auf Monitor: GrößteZahl</p> <p>Nochmal (J/N) ?</p> <p>Bis Nochmal &lt;&gt; N</p>	<p>Ermitteln (A, B, C)</p> <pre> graph TD     A["A &gt; B"] -- J --&gt; B1["B &gt; C"]     A -- N --&gt; B2["B &gt; C"]     B1 -- J --&gt; C1["C &gt; A"]     B1 -- N --&gt; C2["C &gt; A"]     B2 -- J --&gt; B["B"]     B2 -- N --&gt; C3["C &gt; A"]     C1 -- J --&gt; A1["A"]     C1 -- N --&gt; A1     C2 -- J --&gt; C["C"]     C2 -- N --&gt; A1     C3 -- J --&gt; C["C"]     C3 -- N --&gt; A1     </pre>
--	---

### 3.1.3 Programmablaufplan





Programmablaufpläne (PAP) werden auch als Flussdiagramm oder Blockdiagramm bezeichnet.

DIN 66001 bzw. ISO 5807

PAP's gibt es seit .

Symbol	Grafische Darstellung	Bedeutung
Ablauflinie		Linie mit Pfeilspitze verdeutlicht die weitere Abarbeitungsreihenfolge: - Von oben nach unten bzw. - Von links nach rechts
Anzeige		Symbol für die Ausgabe: hier Monitor
Bemerkung		Die Klammer dient dazu, Erläuterungen einzugeben.
Grenzstelle		Rechteck mit abgerundeten Ecken. Kennzeichnet den Beginn und das Ende des Programmes.
Manuelle Eingabe von Daten		Das Symbol zeigt von der Seite betrachtete Tastatur.
Schleifenbegrenzung		Das obere Symbol stellt den Beginn der Schleife und das untere Symbol das Ende der Schleife dar. Im oberen Symbol wird der Name der Schleife eingetragen und um unteren Symbol die Ende-Bedingung.
Unterprogrammaufruf		Aufruf des angegebenen Unterprogramms.



Übergangsstelle		Kreis symbolisiert eine Übergangsstelle. Damit wird ausgedrückt, dass an anderer Stelle (Name) die weitere Bearbeitung erfolgt.
Verarbeitung		Anweisung ausführen
Verzweigung		Raute (Rhombus) kennzeichnet die Stelle im Programm, wo eine alternative Abarbeitung erfolgt.
Zusammenführung		Hier treffen sich mehrere Abarbeitungsrichtungen.

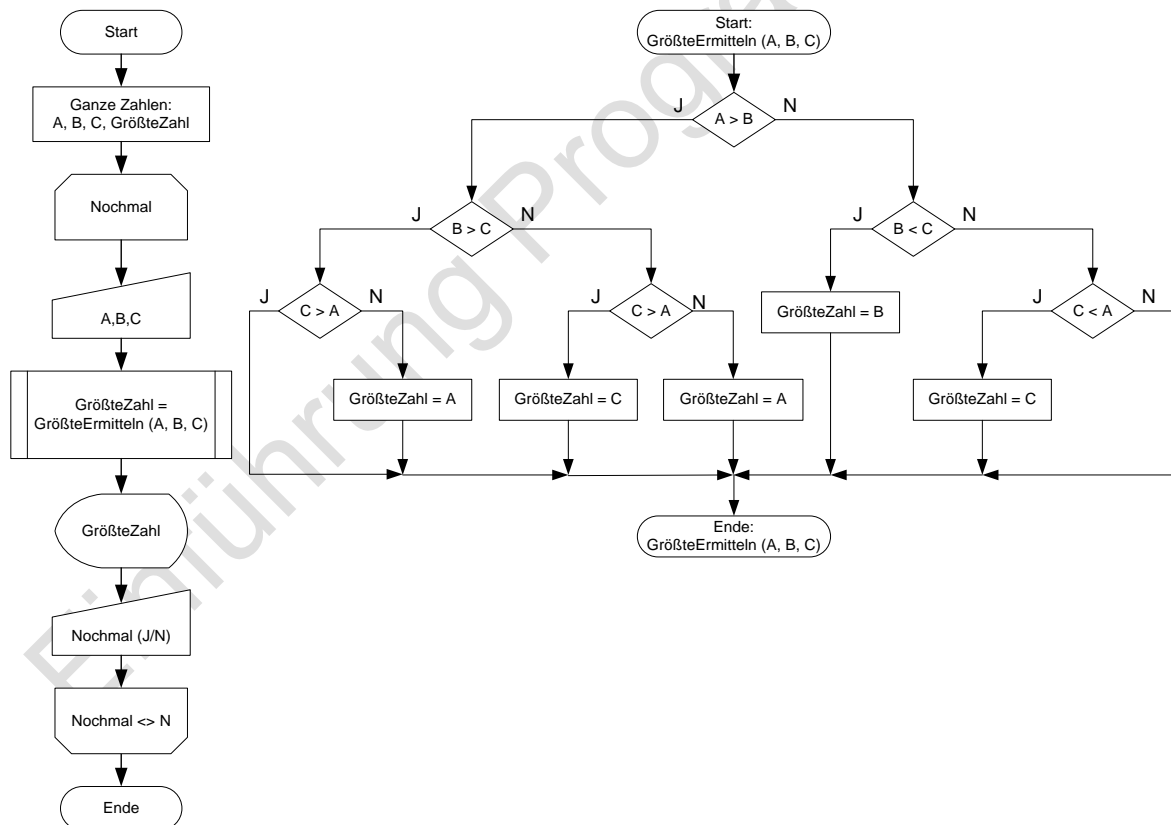
Aufgabe:

Ermittle die Größe von 3 unterschiedlichen Ganzen Zahlen: A, B und C.

Die Eingabe der Werte erfolgt interaktiv.

Die Ausgabe der größten Zahl erfolgt auf dem Monitor.

Die Ermittlung der größten Zahl kann für andere Zahlen beliebig oft wiederholt werden.



### 3.1.4 Struktogramm


Struktogramm wird auch als Nassi-Shneiderman-Diagramm bezeichnet.

1973: I. Nassi und B. Shneiderman

DIN 66261

Ein Struktogramm wird in Form eines geschlossenen Blockes dargestellt.

Symbol	Grafische Darstellung	Bedeutung
Folge (Sequenz)		Eine Folge stellt mehrere Arbeitsschritte dar, die von oben nach unten bearbeitet werden.
Verarbeitung		Ausführung einer Anweisung, beispielsweise einer Berechnung
Verzweigung-einfach		Es wird nur ein möglicher Fall betrachtet.
Verzweigung - zweifach		Je nachdem ob die Bedingung erfüllt ist, erfolgt eine alternative Abarbeitung.
Verzweigung mehrfach		In Abhängigkeit vom Wert, der in der Variablen hinterlegt ist, wird ein Anweisungsblock abgearbeitet. Nur wenn keine Gleichheit vorliegt, wird der Fall sonst bearbeitet.
Wiederholung Zählschleife		Bei einer Zählschleife ist die Anzahl der Iterationen bekannt. 1. Die Zählvariable wird mit den angegebenen Startwert initialisiert. 2. Ist der Wert der Zählvariablen kleiner als der ebenfalls angegebene Endwert, so wird der Schleifenkörper abgearbeitet und anschließend der Wert der Zählvariablen um die festgelegte Schrittweite geändert. Anderenfalls wird die Zählschleife beendet. Gehe zu 2.
Wiederholung kopfgesteuerte (anfangsgeprüfte) Schleife		Bei einer kopfgesteuerten Schleife ist die Anzahl der Iterationen nicht bekannt. Nur wenn die Bedingung erfüllt ist, wird der Schleifenkörper abgearbeitet. Der Schleifenkörper wird 0, 1, 2 oder N-mal durchlaufen.
Wiederholung fußgesteuerte (endgeprüfte) Schleife		Bei einer fußgesteuerten Schleife ist die Anzahl der Iterationen nicht bekannt. Es wird zunächst der Schleifenkörper

Schleife		bearbeitet. Erst danach wird die Bedingung überprüft. Nur wenn die Bedingung erfüllt ist, wird der Schleifenkörper erneut abgearbeitet. Der Schleifenkörper wird 1, 2 oder N-mal durchlaufen.
Aufruf		Aufruf eines Unterprogramms Dieses Symbol ist nicht genormt.

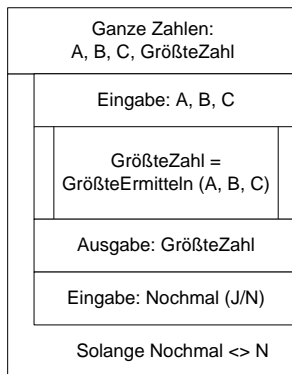
#### Aufgabe:

Ermittle die GröÙte von 3 unterschiedlichen Ganzen Zahlen: A, B und C.

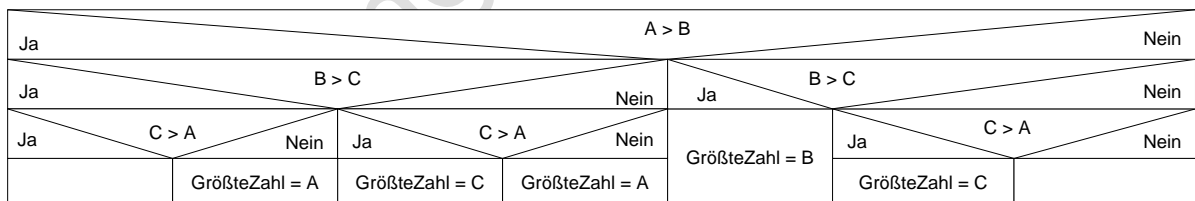
Die Eingabe der Werte erfolgt interaktiv.

Die Ausgabe der größten Zahl erfolgt auf dem Monitor.

Die Ermittlung der größten Zahl kann für andere Zahlen beliebig oft wiederholt werden.



GröÙteErmitteln (A, B, C)



## 3.2 Unified Modeling Language

UML ist eine Sammlung von Notationen zur Formulierung von Algorithmen/Abläufen bei der Software-Entwicklung.

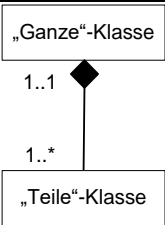
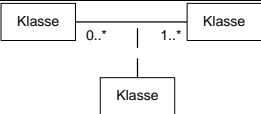
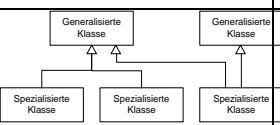
Ein UML-Diagramm kann durchaus mehrere Diagrammartentypen enthalten.

### 3.2.1 Klassendiagramm

Das Klassendiagramm verdeutlicht die Struktur einer Klasse, die Datenhaltung (Klassenvariablen) und/oder die Datenverarbeitung (Methoden).

### 3.2.1.1 Diagrammelemente

Diagramm- element	Symbol	Bedeutung
Klasse		<p>Eine Klasse kann aus drei Bestandteilen bestehen, die jeweils durch horizontale Linie voneinander getrennt sind.</p> <p>Obere Bereich: enthält den Klassennamen</p> <p>Mittlere Bereich: Angabe der Klassenvariablen</p> <p>Untere Bereich: Angabe der Methoden</p>
Klassen- variable	[ Sichtbarkeit ] [ / ] Klassenvariablenname [ : Datentyp ] [ = Initialwert ]	
Methode	[Sichtbarkeit] [/] Methodenname [( {Parameterliste} )] [: Rückgabedatentyp]	
Sichtbarkeit	+	public: Jeder hat Zugriff die Komponente der Klasse.
	#	protected: Der Zugriff auf die Komponenten ist auf das Objekt der Klasse selbst und auf die Objekte der Klassen, deren Klassen in der Vererbungslinie angegeben sind, beschränkt.
	-	private: Der Zugriff auf die Komponenten ist auf das Objekt der Klasse selbst beschränkt.
	~	package: Klassen im gleichen Paket
Assoziation		<p>Eine Assoziation beschreibt eine Beziehung zwischen Klassen. Der Grad der Beziehung erklärt, wie viele Klassen an der Beziehung gleichzeitig teilnehmen:</p> <p>Grad 1: Unäre Beziehungen</p> <p>Grad 2: Binäre Beziehungen</p> <p>Grad 3: Ternäre Beziehungen</p> <p>...</p> <p>Am Ende wird die Multiplizität angegeben.</p>
Multiplizität	<p>Die Multiplizität drückt aus, wie viele Objekte der einen Klasse Objekte der anderen Klasse kennen.</p> <p>Unter Multiplizität wird ein Intervall der Form: untereGrenze .. obereGrenze verstanden, wobei die untere und die obere Grenze jeweils durch eine natürliche Zahl festgelegt werden.</p>	
Aggregation		<p>Aggregation ist eine Beziehung zwischen einer "Ganze"-Klasse und seinen "Teil"-Klassen. Die "Ganze"-Klasse setzt sich aus den "Teile"-Klassen zusammen. Fällt die "Ganze"-Klasse aus, so können die "Teile"-Klassen einer anderen "Ganze"-Klasse zugeordnet werden.</p>

Komposition		<p>Die Komposition ist eine strenge Form der Aggregation. Im Gegensatz zur Aggregation können die "Teile"-Klassen nicht ohne die "Ganze"-Klasse existieren. Wird beispielsweise die "Ganze"-Klasse kopiert, so werden die "Teile"-Klassen ebenfalls mit kopiert.</p>
Assoziations- klasse		<p>Eine Assoziationsklasse ist eine Klasse, die von dem Vorhandensein einer Assoziation zwischen zwei Klassen abhängt. Sie enthält Komponenten, die keiner der beiden Klassen sinnvoll zugeordnet werden können.</p> <p>Wenn die Assoziation einen Namen besitzt, so muss die Assoziationsklasse denselben Namen besitzen.</p>
Vererbung		<p>Vererbung ist eine spezielle Beziehung zwischen einer Oberklasse und einer oder mehrere Unterklassen (einfache Vererbung). Die Unterklasse kann aber auch von mehreren Oberklassen erben (mehrfache Vererbung).</p> <p>Die Unterklasse kann implizit auf die vererbten Komponenten der Oberklasse zugreifen, ohne sie in der Unterklasse definieren zu müssen.</p>

### 3.2.1.2 Beispiel

Jedes Gebäude besitzt einen oder mehrere Räume und jeder Raum gehört zu einem Gebäude. Wenn das Gebäude nicht mehr existiert, so werden auch die Räume gelöscht.

Jeder PC steht in einem Raum, wobei im Raum aber mehrere PC's stehen können. Der PC kann durchaus in einem anderen Raum stehen, wenn beispielsweise der Raum nicht mehr existiert.

Jeder PC kann einem Angestellten zugeordnet sein, aber jeder Angestellte kann an mehreren PC's arbeiten.

Angestellte und Freiberufler sind spezialisierte Klassen der generalisierten Klasse Mitarbeiter.

Zu jedem Mitarbeiter kann es einen Ehepartner geben, aber jeder Ehepartner ist mit einem Mitarbeiter verheiratet. Zu dieser Assoziation wird das Hochzeitsdatum angegeben.

