Abiturprüfung ab 2026	Berufliches Gymnasium (TG)
Formelsammlung	1.5.2 Informationstechnik

Formelsammlung

1.5.2 TG Informationstechnik

Formelsammlung Allgemein

Version: V 5.1

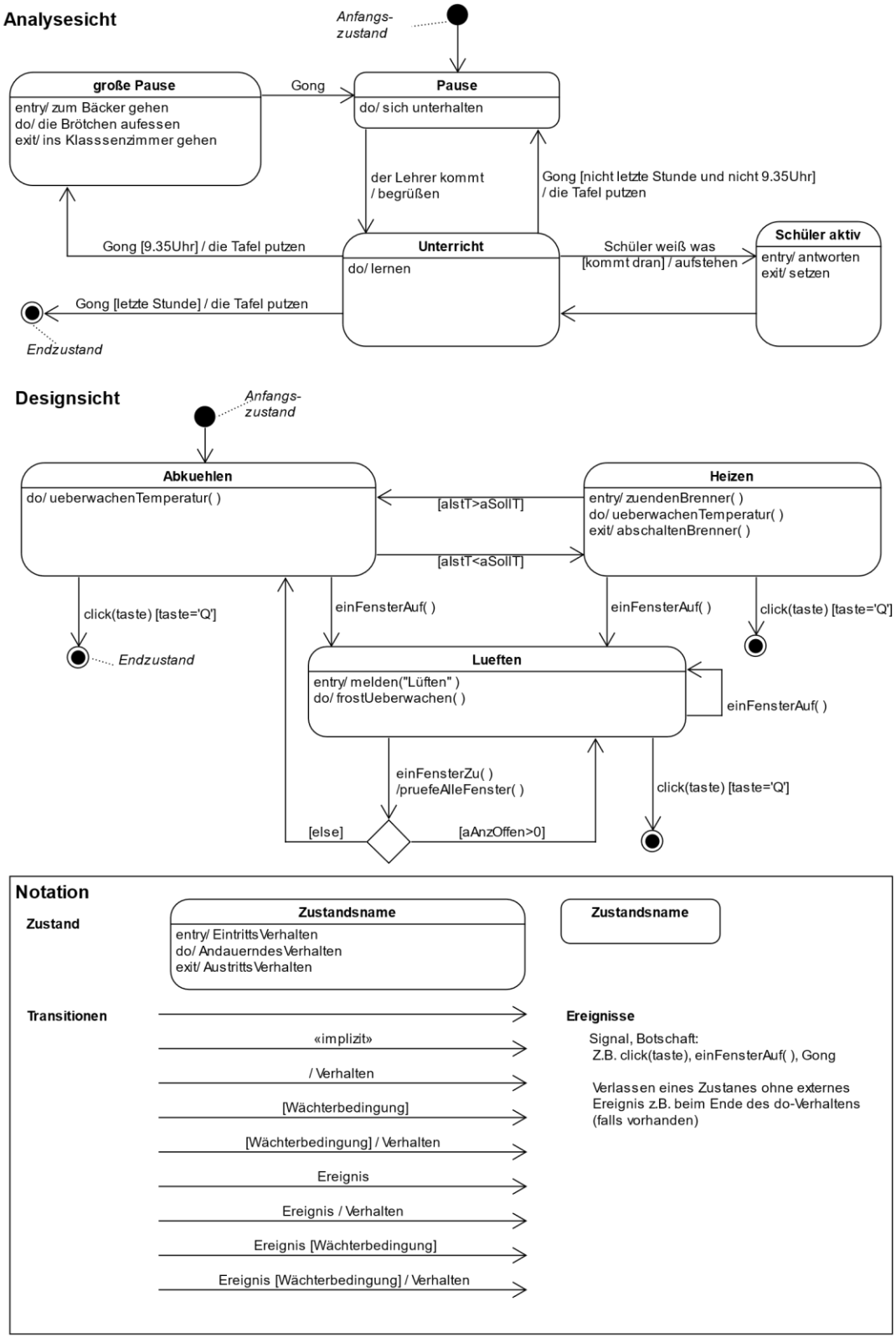
Gültig ab Abitur 2026

Abiturprüfung ab 2026	Berufliches Gymnasium (TG)
Formelsammlung	1.5.2 Informationstechnik

Inhaltsverzeichnis:

1 Beschreibung von Systemzuständen mit UML-Zustandsdiagrammen

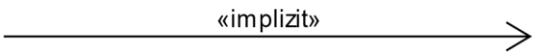
1.1 UML-Zustandsdiagramme (allgemein)



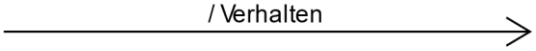
1.1.1 Varianten von Transitionen

Transitionen bezeichnen Zustandsübergänge und werden als Pfeil mit offener Spitze vom Ausgangszustand zum Zielzustand gezeichnet.

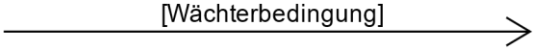
a) **Transition ohne Beschriftung:**
Sie bewirkt einen unmittelbaren Zustandswechsel, nachdem das `entry`-Verhalten beendet wurde.



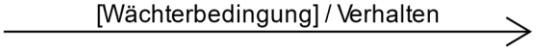
Alternative zu a)



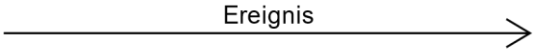
b) **Transition mit Verhalten:**
Verhalten wie bei a). Zusätzlich wird beim Zustandswechsel noch das Verhalten ausgeführt.



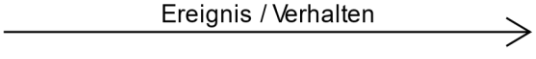
c) **Transition mit Wächterbedingung:**
Sie bewirkt einen Zustandswechsel, sobald die Wächterbedingung erfüllt ist.



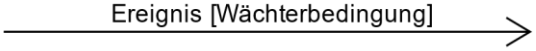
d) **Transition mit Wächterbedingung und Verhalten:**
Sie bewirkt einen Zustandswechsel, sobald die Wächterbedingung erfüllt ist. Beim Zustandswechsel wird das Verhalten ausgeführt.



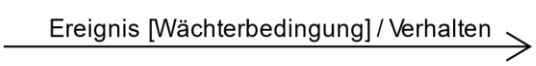
e) **Transition mit Ereignis:**
Sie bewirkt einen Zustandswechsel beim Eintreten des Ereignisses.



f) **Transition mit Ereignis und Verhalten:**
Verhalten wie bei d). Zusätzlich wird beim Zustandswechsel noch das Verhalten ausgeführt.



g) **Transition mit Ereignis und Wächterbedingung:**
Verhalten wie bei d), falls zusätzlich die Wächterbedingung erfüllt ist.



h) **Transition mit Ereignis, Wächterbedingung und Verhalten**
Verhalten wie bei f). Zusätzlich wird beim Zustandswechsel noch das Verhalten ausgeführt.

1.2 Allgemeines für UML-Zustandsdiagramme

- ✓ ISR klein halten => Zustandsänderung mit Variable, Funktionsaufrufe wenn möglich vermeiden
- ✓ Funktionen = Operationen, wenn möglich mit Kleinbuchstaben beginnen
- ✓ `init()` bzw. `setup()` sollte als Verhalten nach Start beschrieben werden und nicht unbedingt als Anfangszustand
- ✓ Keine Unterscheidung UML-Ereignis und HW-Ereignis => somit `isr_name()` als Ereignis möglich
- ✓ `#define` bzw. `enum`-Werte mit Großbuchstaben
- ✓ Guard = Wächter - didaktische Interpretation => HW-IR-Einheit als „Aufpasser“ der eine Flagge zeigt und somit die Bedingung erfüllt für ein HW-Ereignis

1.3 Ergänzungen für Mikrocontroller

Hinweis: Die folgenden Codebeispiele sind nicht verbindlich

1.3.1 Zustandsdefinition in C/CPP

Zuständen sollten aus Gründen der Übersichtlichkeit Namen gegeben werden. Dadurch wird der Zusammenhang von Zustandsdiagramm und Programm verdeutlicht.

Allgemein	Beispiel
`#define ZUSTANDSNAME Zustandsnummer`	`#define BLINKEN 1`
oder	
`enum zustandstyp {ZUSTANDSNAME=Zustandsnummer, … }`	`enum zustandstyp {BLINKEN =1, …};`

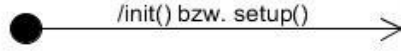
1.3.2 Zustandsvariable C/CPP

Ein Zustand kann durch eine Zustandsvariable gekennzeichnet werden:

Beispiele	Erklärung
`int zustand;`	Zustandsvariable vom Typ `int`
`zustandstyp zustand;`	Zustandsvariable als `enum` (siehe oben)

1.3.3 Der Start-Pseudozustand

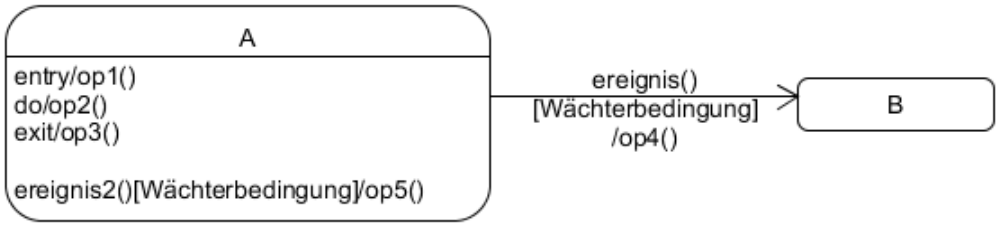
Die meisten Zustandsdiagramme beginnen mit einem Start-Pseudozustand:



Der ausgefüllte Kreis symbolisiert den Startpunkt des Zustandsdiagramms. Oft ist er mit dem Start des Mikrocontrollerprogramms gleich zu setzen. Danach kommt als *erstes Verhalten* der Initialisierungsaufruf.

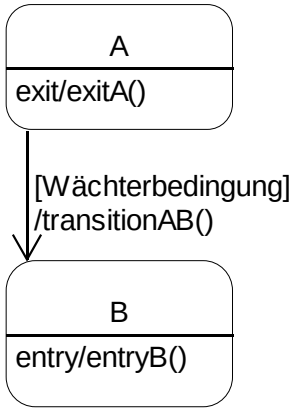
1.3.4 Verhalten

Verhalten sind Operationen oder Anweisungen, die an bestimmten Stellen des Zustandsdiagramms ausgeführt werden



Verhalten	Ausführung	Beispiel
`Entry`-Verhalten	bei Eintritt in einen Zustand	`op1()`
`Do`-Verhalten	andauernd, solange der Zustand anhält	`op2()`
`Exit`-Verhalten	bei Verlassen des Zustands	`op3()`
Verhalten an der Transition	beim Zustandswechsel	`op4()`
Verhalten am internen Ereignis	wenn das interne Ereignis eintritt und gegebenenfalls eine Wächterbedingung erfüllt ist	`op5()`

1.3.5 Zustandsübergang mit Wächterbedingung

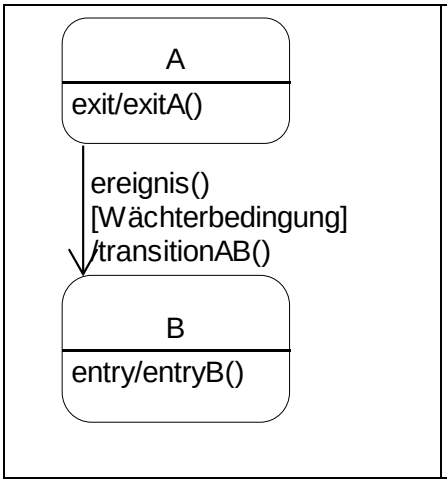



```
int main() {
   while(true) {
      switch (zustand) {
         case A:
            if (Wächterbedingung) {
               exitA();
               transitionAB();
               zustand=B;
               entryB();
            }
            break;
         case B:
            break;
      }
   }
}
```

In der Endlosschleife wird zuerst der Zustand geprüft. Falls sich der Mikrocontroller im Zustand A befindet, wird die Wächterbedingung an der Transition überprüft. Falls die Wächterbedingung erfüllt ist, erfolgt der Zustandswechsel. Es werden dann in folgender Reihenfolge die Verhalten ausgeführt:

1. `Exit`-Verhalten von Zustand A: `exitA()`
2. Verhalten an der Transition: `transitionAB()`
3. Zustandswechsel: `zustand=B`
4. `Entry`-Verhalten von Zustand B: `entryB()`

1.3.6 Zustandsübergang mit Ereignis und Wächterbedingung



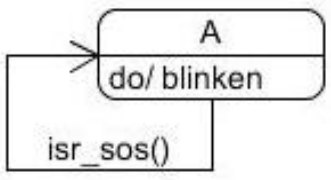

```
void ereignis() {
   switch (zustand) {
      case A:
         if (Wächterbedingung) {
            exitA();
            transitionAB();
            zustand=B;
            entryB();
         }
         break;
      case B:
         break;
   }
}
```

Es gibt **Aufruf-** und **Signal-Ereignisse**. Bei Signal-Ereignissen handelt es sich um Interrupts. Als Ereignisbezeichnung wird der Name der **Interrupt Service Routine** `isr_ereignis()` verwendet.

In der ISR wird zuerst der Zustand geprüft. Falls sich der Mikrocontroller im Zustand A befindet, wird die Wächterbedingung an der Transition überprüft. Falls die Wächterbedingung erfüllt ist, erfolgt der Zustandswechsel. Es wird dann in folgender Reihenfolge das Verhalten ausgeführt:

1. `Exit`-Verhalten von Zustand A: `exitA()`
2. Verhalten an der Transition: `transitionAB()`
3. Zustandswechsel: `zustand=B`
4. `Entry`-Verhalten von Zustand B: `entryB()`

1.3.7 Selbsttransition



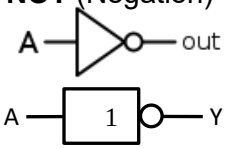

```
void isr_sos() {
   switch (zustand) {
      case A:
         do/blinken
         zustand = A;
         break;
   }
}
```

In der ISR `isr_sos()` wird zuerst der Zustand geprüft. Falls sich der Mikrocontroller im Zustand A befindet, wird die Wächterbedingung, falls vorhanden, an der Transition überprüft. Falls die Wächterbedingung erfüllt ist, erfolgt der Zustandswechsel wieder nach A.

2 Hardware - Digitaltechnik

2.1 Logikgatter

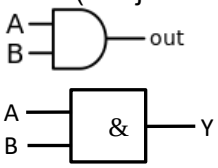
NOT (Negation)



A	Y
0	1
1	0

$Y = \overline{A}$
Y = !A
Y = ¬A

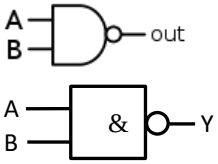
AND (Konjunktion)



B	A	Y
0	0	0
0	1	0
1	0	0
1	1	1

Y = A ∧ B
Y = A & B

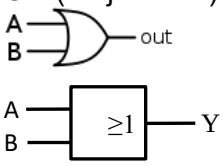
NAND



B	A	Y
0	0	1
0	1	1
1	0	1
1	1	0

$Y = \overline{A \wedge B}$
Y = !(A&B)

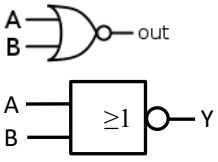
OR (Disjunktion)



B	A	Y
0	0	0
0	1	1
1	0	1
1	1	1

Y = A ∨ B
Y = A # B

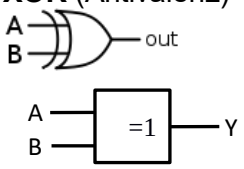
NOR



B	A	Y
0	0	1
0	1	0
1	0	0
1	1	0

$Y = \overline{A \vee B}$
Y = !(A#B)

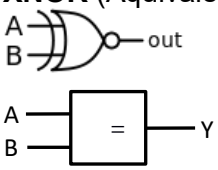
XOR (Antivalenz)



B	A	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = (\overline{A} \wedge B) \vee (A \wedge \overline{B})$
Y = (!A&B)#(A&!B)

XNOR (Äquivalenz)

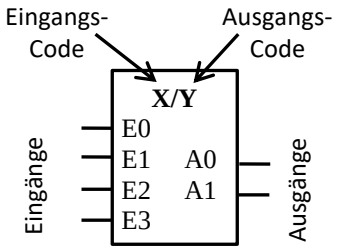


B	A	Y
0	0	1
0	1	0
1	0	0
1	1	1

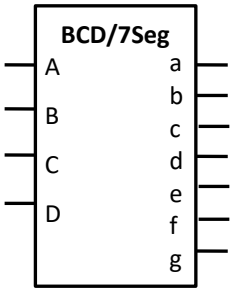
$Y = (\overline{A} \wedge \overline{B}) \vee (A \wedge B)$
Y = (!A&!B)#(A&B)

2.2 Schaltnetze

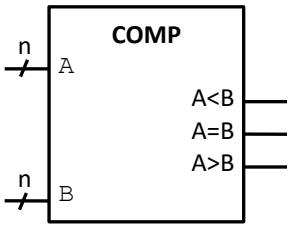
Codeumsetzer (Umcodierer)



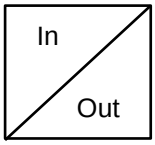
BCD zu 7 Seg



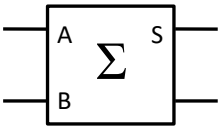
Vergleicher



BSB Codewandler

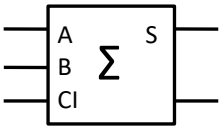


Halbaddierer



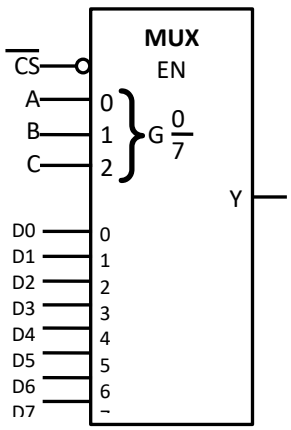
C_O = Carry OUT
S = Summe

Volladdierer



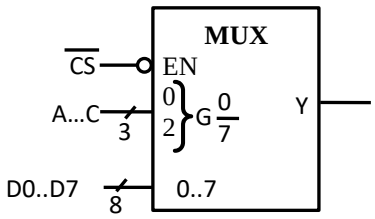
C_I = Carry IN
C_O = Carry OUT

MUX (8 zu 1)



C	B	A	$\overline{CS}$	Y
x	x	x	1	0
0	0	0	0	D0
0	0	1	0	D1
0	1	0	0	D2
0	1	1	0	D3
1	0	0	0	D4
1	0	1	0	D5
1	1	0	0	D6
1	1	1	0	D7

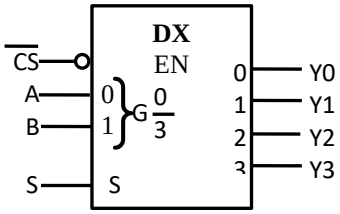
x = don't care



Adress- und Datenleitungen können auch zusammengefasst werden

CS = chip select (low active)

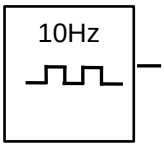
DEMUX (1 zu 4)
Decodierer



B	A	$\overline{CS}$	Y3	Y2	Y1	Y0
X	X	1	0	0	0	0
0	0	0	0	0	0	S
0	1	0	0	0	S	0
1	0	0	0	S	0	0
1	1	0	S	0	0	0

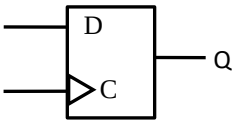
2.3 Schaltwerke

2.3.1 Taktgenerator



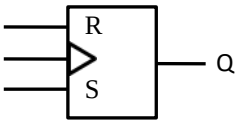
2.3.2 Flip-Flops

D-Flip-Flop



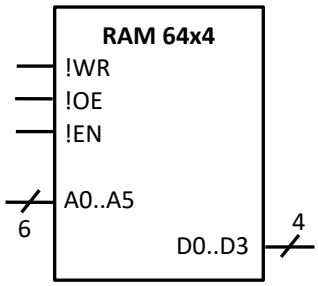
Takt	D	Q^{n+1}
↑	0	0
↑	1	1
sonst	X	Q^n

RS-Flip-Flop



Takt	R	S	Q^{n+1}
↑	0	0	Q^n
↑	1	0	0
↑	0	1	1
↑	1	1	Undefiniert
sonst	x	x	Q^n

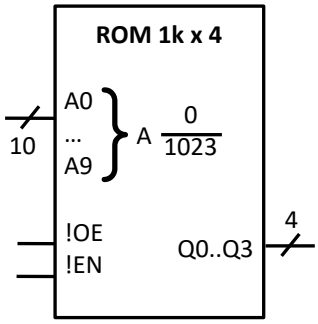
2.3.3 RAM



Schreib-Lese-Speicher mit 64 mal 4 Bit

- 4-Bit Registerbreite
- 64 Register gesamt
- **A0-A5**: Adresseingänge
- **D0-D3**: Ein-/Ausgabe des Speicherinhalts
- **WR=0:** lesen (von **D0-D3** in den Speicher)
  WR=1: schreiben (vom Speicher an **D0-D3**)
- **OE=1:** Hochohmig
- **OE=0:** Speicherinhalt lesen
- **EN=0**: aktiviert den Baustein

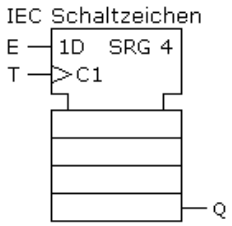
2.3.4 ROM

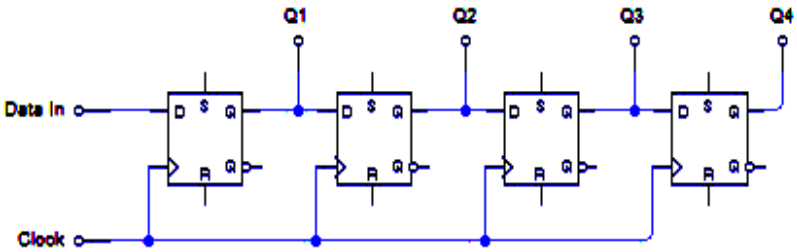


Festwertspeicher mit 1024 (1KiBi) mal 4 Bit

- **A0-A9**: Adresseingänge
- **OE=1:** Hochohmig
- **OE=0:** Speicherinhalt lesen
- **EN=0**: aktiviert den Baustein
- **Q0-Q3**: Wert der Speicherzelle an Adresse A

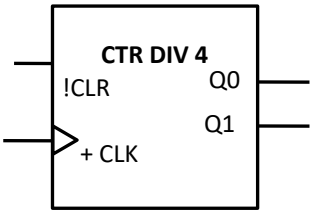
2.3.5 Schieberegister





Beispiel: Seriell In => Parallel Out

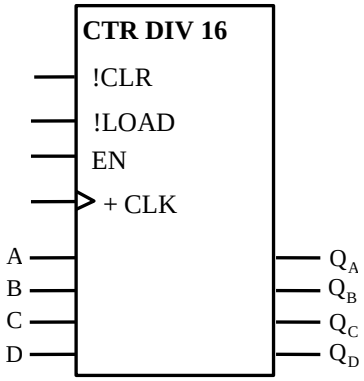
2.3.6 Zähler (Blockschaltbild)



Mit jeder steigenden Flanke an **CLK** wird der Zählerwert um 1 erhöht. Nach dem maximalen Wert wird der Zählwert wieder auf 0 gesetzt.

- **CTR:** Zähler (counter)
- **DIV 4:** 4 verschiedene binäre Zustände
- **CLR = 0** setzt den Counter auf den Wert **0** zurück
- **Q_n** gibt den Zählerzustand aus

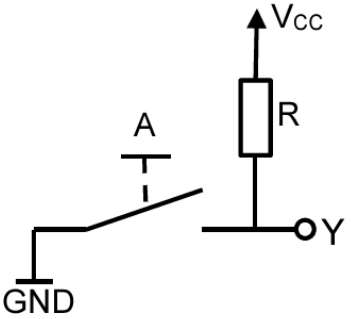
2.3.7 Zähler (4-Bit)



- **CTR:** Zähler (counter)
- **DIV 16:** 16 verschiedene binäre Zustände
- Vorwärtszähler (+)
- **EN = 1** und die positive Taktflanke führen zum nächsten Zählzustand
- Mit **LOAD = 0** kann ein Anfangszustand geladen werden
- **CLR = 0** setzt den Counter auf den Wert 0 zurück

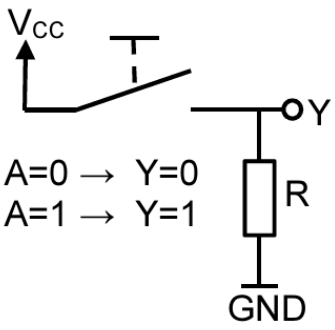
2.4 Sensoren

Taster mit Pull-Up-Widerstand



A=0 → Y=1
A=1 → Y=0

Taster mit Pull-Down Widerstand



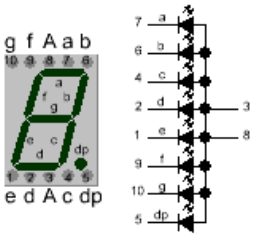
A=0 → Y=0
A=1 → Y=1

2.5 Aktoren

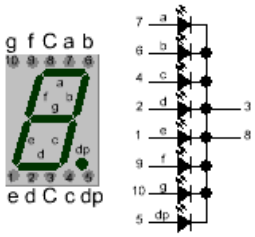
7-Segmentanzeige

Common Anode ⇔ Common Cathode

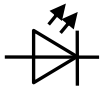
SA52



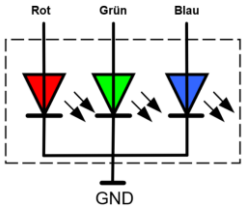
SC52

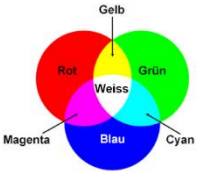


LED

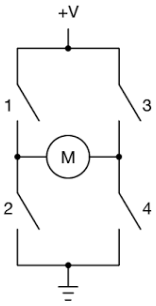


RGB LED

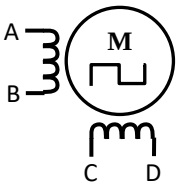




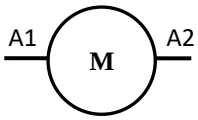
H-Brücke



Schrittmotor



DC-Motor



BLDC-Motor

3 Programmentwicklung und Objektorientierter Entwurf

3.1 Vergleichsoperatoren für Bedingungen (Pseudocode)

<, <=, >, >=, == oder =, ≠ oder !=

Anmerkung: Die Operatoren für Vergleiche und Wertzuweisungen müssen unterschieden werden können.

3.2 Kontrollstrukturen (Pseudocode)

Zuweisung

```
dieVariable ← derAusdruck
dieVariable := derAusdruck
dieVariable = derAusdruck
```

Sequenz

```
anweisung1
anweisung2
anweisung3
```

Auswahl

Einseitige Auswahl

```
WENN bedingung
  anweisung1
  …
ENDE WENN
```

Zweiseitige Auswahl

```
WENN bedingung
  anweisungA1
  …
SONST
  anweisungB1
  …
ENDE WENN
```

Mehrfachauswahl

```
FALLS variable GLEICH
  bedingung1: anweisungA1
              …
  bedingung2: anweisungB1
              …
  bedingung3: anweisungC1
              …
  SONST: anweisungD1
         …
ENDE FALLS
```

Schleife (Iteration)

Schleife mit Eintrittsbedingung

```
SOLANGE bedingung
  anweisung1
  …
ENDE SOLANGE
```

Schleife mit Austrittsbedingung

```
WIEDERHOLE
  anweisung1
  …
SOLANGE bedingung
```

Zählschleife

```
FÜR i←0 BIS n SCHRITT s
  anweisung1
  …
ENDE FÜR
```

Schleife über Kollektion

```
FÜR element IN kollektion
  anweisung1
  …
ENDE FÜR
```

Schleife mit Abbruchbedingung

```
FÜR element IN kollektion
  anweisungA1
  …
  WENN bedingung
    ABBRUCH
  ENDE WENN
  anweisungB1
  …
ENDE FÜR
```

3.3 Datentypen

3.3.1 Elementare Datentypen

Datentyp	Abkürzungen	Werte
Boolscher Datentyp	Boolean, boolean, bool, …	wahr, falsch, true, false
Ganzzahliger Datentyp	GZ, Integer, int, …	-24, 0, 123, …
Fließkomma-Datentyp	FKZ, Real, double, …	-3.567, 0.0, 3.141, …
Zeichen-Datentyp	Zeichen, char, …	'Z', 'a', '&', …
Text-Datentyp	Text, String, string, …	"Hello world!!!", …

Für den Datentyp `Text` ist als Vergleichsoperator nur == bzw. = definiert. Außerdem kann der Operator + für die Verbindung von zwei Texten verwendet werden. Auch bei Texten muss der Vergleich und die Zuweisung eindeutig unterschieden werden können (vgl. 4.1).

3.3.2 Komplexe Datentypen

Zeit
...
+Zeit( ) +Zeit(pStunde:GZ,pMinute:GZ,pSekunde:GZ) +gibStunde( ):GZ +gibMinute( ):GZ +gibSekunde( ):GZ +istVor(pZeit:Zeit):Boolean +istNach(pZeit:Zeit):Boolean +zeitMinusSekunden(pSekunden:GZ):Zeit +zeitPlusSekunden(pSekunden:GZ):Zeit +gibText( ):Text

Datum
...
+Datum( ) +Datum(pTag:GZ,pMonat:GZ,pJahr:GZ) +gibTag( ):GZ +gibMonat( ):GZ +gibJahr( ):GZ +istVor(pDatum:Datum):Boolean +istNach(pDatum:Datum):Boolean +anzahlTageBis(pDatum:Datum):GZ +anzahlTageSeit(pDatum:Datum):GZ +gibText( ):Text

Liste<Typ>
...
+Liste<Typ>( ) +istLeer( ):Boolean +anzahlElemente( ):GZ +gibInhalt(pIndex:GZ):Typ +ersetzen(pIndex:GZ,pInhalt:Typ) +einfuegen(pIndex:GZ,pInhalt:Typ) +einfuegenVorne(pInhalt:Typ) +anhaengen(pInhalt:Typ) +verketten(pListe:Liste<Typ>) +entfernen(pIndex:GZ):Typ +entfernenInhalt(pInhalt:Typ) +enthaelt(pInhalt:Typ):Boolean +kopieren( ):Liste<Typ> ...

Listen beinhalten Daten vom gleichen Typ. Dabei kann es sich um elementare oder komplexe Datentypen (Klassen) handeln, z.B. `Liste<GZ>` oder `Liste<Person>`.

Die Operationen `ersetzen` und `einfuegen` unterscheiden sich dadurch, dass beim Ersetzen der Inhalt am Index `pIndex` ersetzt wird und die Liste somit ihre Länge behält, während beim Einfügen die Liste verlängert wird, da der Inhalt `pInhalt` die nachfolgenden Inhalte um eine Position nach hinten verschiebt.

Die Operation `entfernen` gibt das gelöschte Objekt vom Datentyp `Typ` zurück. Die Operation `entfernenInhalt` wird mit einem Argument vom Datentyp `Typ` aufgerufen. Sie sucht den Inhalt in der Liste von vorne und löscht den ersten gefundenen Inhalt, falls vorhanden.

Alternative Notationen für Listen

Liste `highscore` vom Datentyp `Liste<GZ>`

Standardnotation	Alternative Notation	Bedeutung
`highscore ← NEU Liste<GZ>()`	`highscore ← []`	Leere Liste anlegen.

`highscore ← NEU Liste<GZ>()`
`FÜR i←0 BIS 2 SCHRITT 1`
`  highscore.anhaengen(0)`
	`highscore ← [0, 0, 0]`	Liste mit drei Elementen anlegen.

`h ← highscore.gibInhalt(0)`	`h ← highscore[0]`	Element einer Liste lesen.
`highscore.ersetzen(3,5)`	`highscore[3] ← 5`	Element einer Liste schreiben.

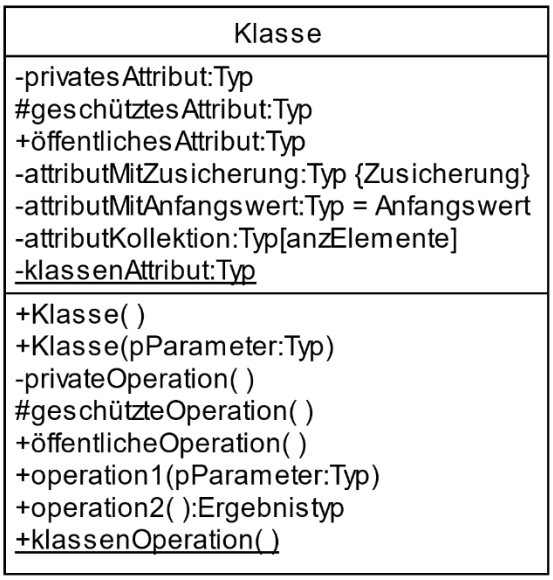
Notationen für Felder

Standardnotation	Bedeutung
`highscore ← NEU GZ[10]`	Feld für 10 Highscores anlegen.
`highscore[0] ← 15`	Ersten Highscore auf 15 setzen.

Notationen für fehlende Referenz

Standardnotation	Alternative Notation	Bedeutung
`gewinner ← NICHTS`	`gewinner ← NULL` `gewinner ← NONE`	Objektreferenz, die auf kein Objekt verweist.

3.4 Klassen



3.4.1 Attribute

Die Bezeichner von Attributen beginnen mit einem Kleinbuchstaben (vgl. UML-Standard). Attribute haben im Klassendiagramm folgenden Aufbau:

```
Sichtbarkeit bezeichner: Typ<[Multiplizität]><=Anfangswert><{Zusicherung}>
```

Die in spitzen Klammern notierten Inhalte, z.B. `<[Multiplizität]>`, sind optionale Bestandteile der Attribute.

Sichtbarkeit	Zeichen
privat	-
geschützt	#
öffentlich	+

Typ
Elementarer Datentyp
Komplexer Datentyp (Klasse)

Anfangswert
Wert, den das Attribut bei der Erzeugung des Objekts annimmt.

Zusicherung
Vorschriften für Attribute `{wert>0}`, `{read only}`.

3.4.2 Operationen

Prozeduren bzw. Funktionen von Programmiersprachen nennt man im Kontext der Objektorientierung Operationen. Ihre Bezeichner starten, wenn möglich, mit einem Verb. Wie bei Attributen ist der erste Buchstabe ein Kleinbuchstabe. Operationen haben im Klassendiagramm folgenden Aufbau:

```
Sichtbarkeit operationsbezeichner(<Parameterliste>)<: Rückgabetyp>
```

Eine Parameterliste kann leer sein oder einen oder mehrere Parameter enthalten. Die Parameter werden nach folgendem Schema definiert:

```
pName: Typ, …
```

Die in spitzen Klammern notierten Inhalte, z.B. `<Parameterliste>`, sind optionale Bestandteile der Operationsdeklaration.

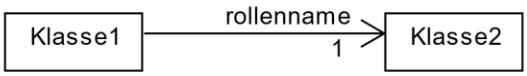
Beispiel einer Operation mit einer Kollektion in Pseudocode

```
OPERATION anlegenPerson(pName: Text, personen: Liste<Person>): Boolean

    Lokale Variablen: gefunden: Boolean, neuePerson: Person, person: Person

    gefunden ← falsch
    FÜR person IN personen
      WENN person.gibName() = pName
        gefunden ← wahr
        ABBRUCH
      ENDE WENN
    ENDE FÜR
    WENN gefunden = falsch
      neuePerson ← NEU Person(pName)
      personen.anhaengen(neuePerson)
    ENDE WENN
    RÜCKGABE gefunden
```

3.4.3 Assoziationen, Rollennamen und Multiplizitäten

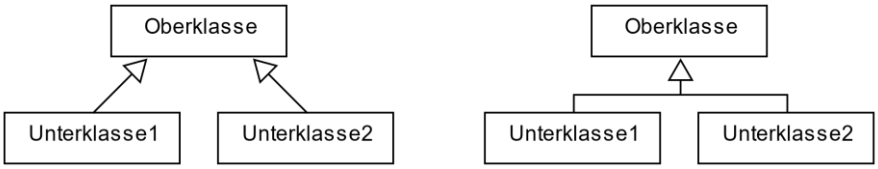


Gerichtete Assoziation

Bidirektionale Assoziation

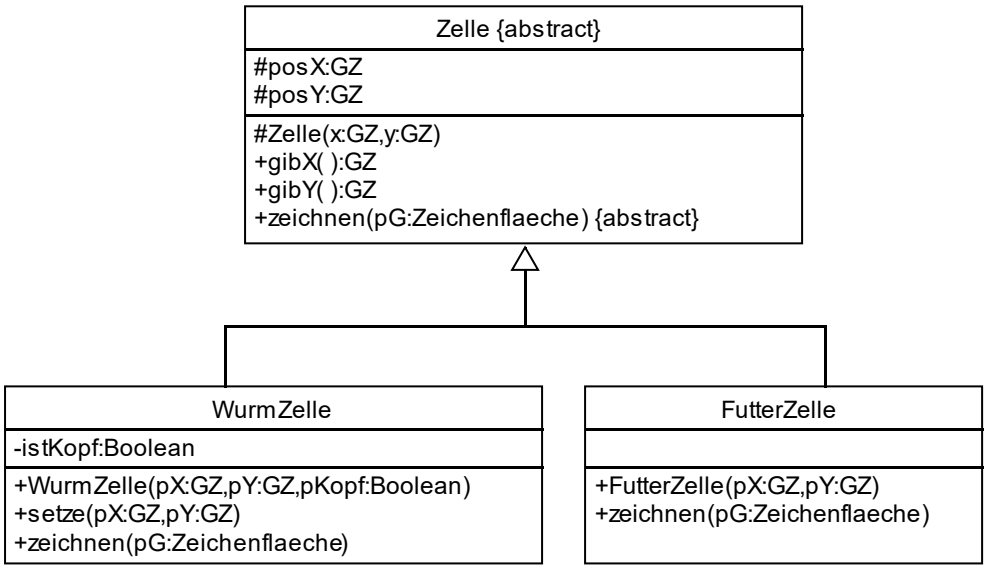
Multiplizität	Bedeutung
1	genau 1
0..1	0 oder 1
3..6	3, 4, 5 oder 6
*	0 bis viele
2..*	2 bis viele

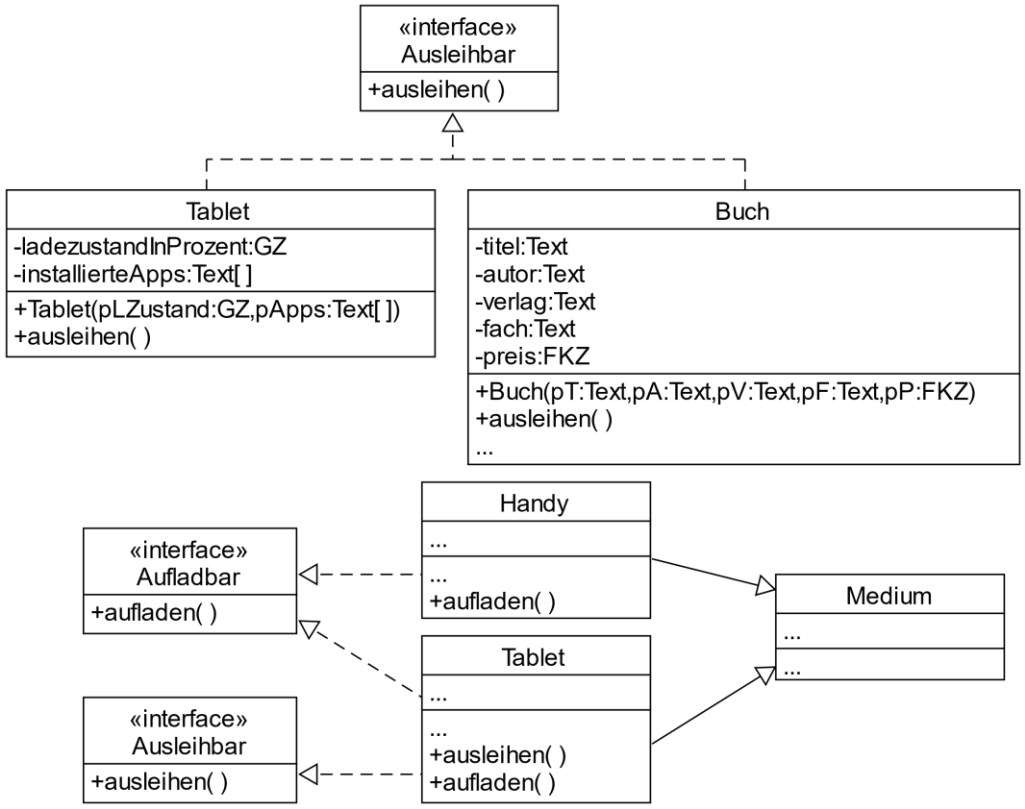
3.5 Vererbung



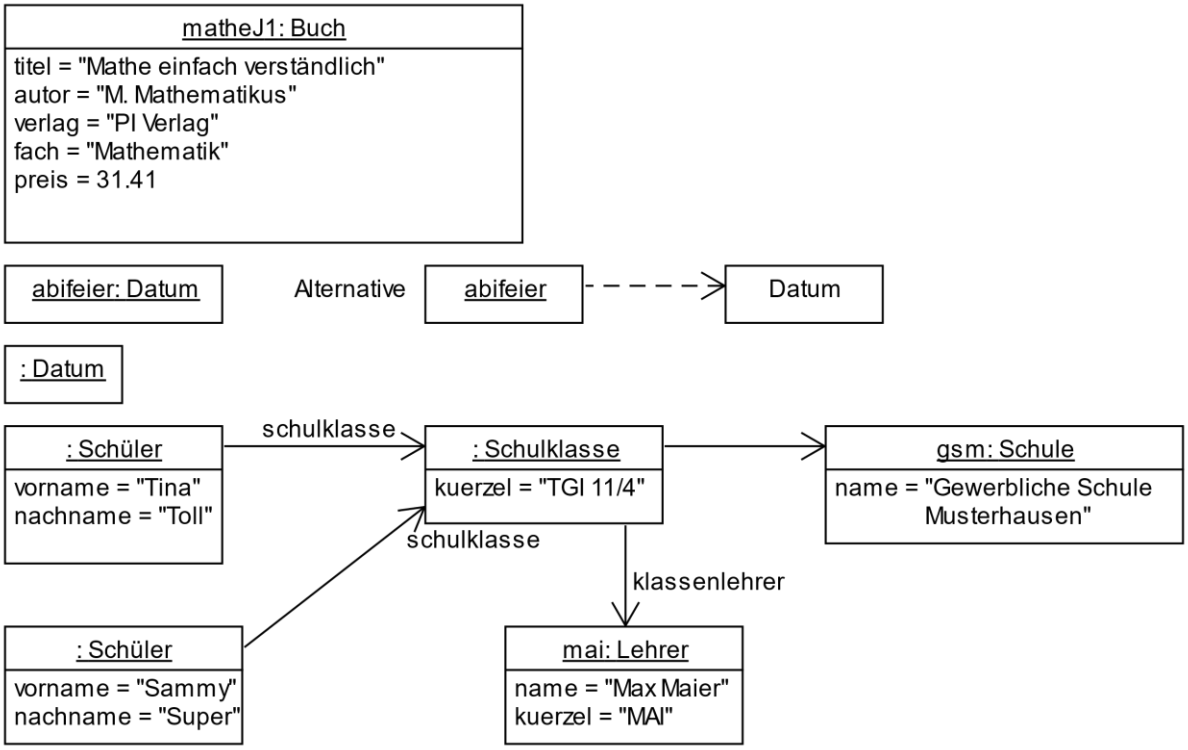
Oberklassen sind Generalisierungen und Unterklassen Spezialisierungen.

3.6 Abstrakte Klassen und Schnittstellen





3.7 Objektdiagramme



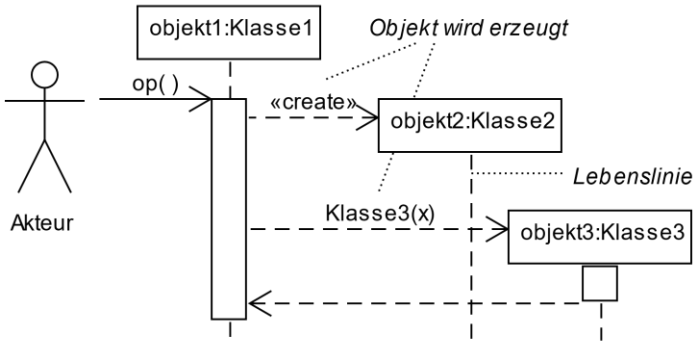
3.8 Sequenzdiagramme

Allgemeines:

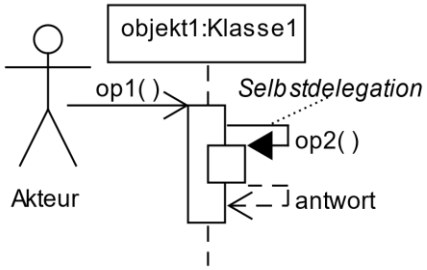
Es wird nicht zwischen unterstrichenen und nicht-unterstrichenen Objekten im Sequenzdiagramm unterschieden.

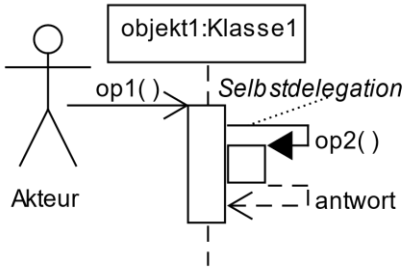
Erzeugung von Objekten

Ein Objekt kann im Sequenzdiagramm immer mit einem spezifischen Konstruktor erzeugt werden. Ist die Auswahl des Konstruktors nicht bedeutsam, so kann die Objekterzeugung durch `<<create>>` dargestellt werden.

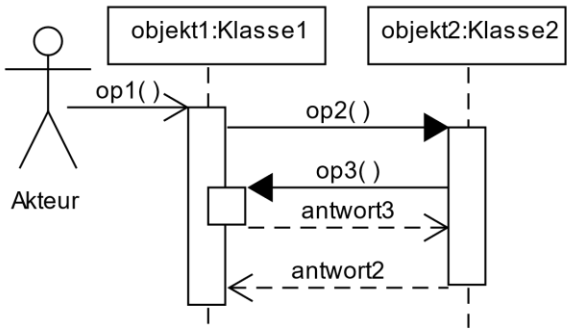


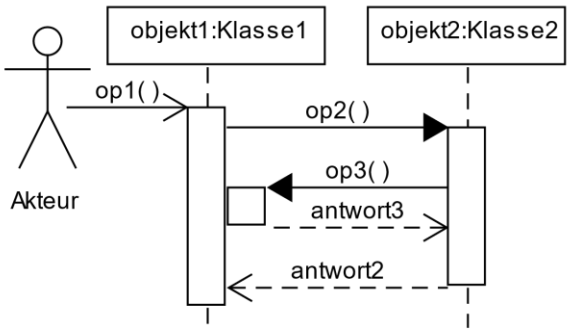
Selbstdelegation (alternative Darstellungen)



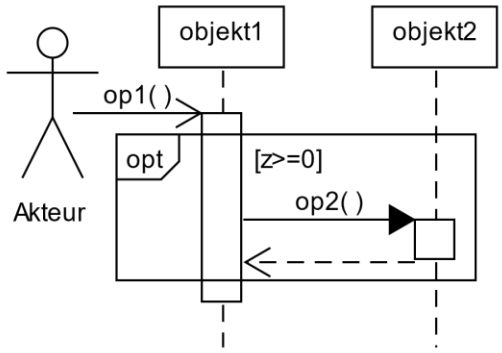


Wechselseitige Botschaften (alternative Darstellungen)

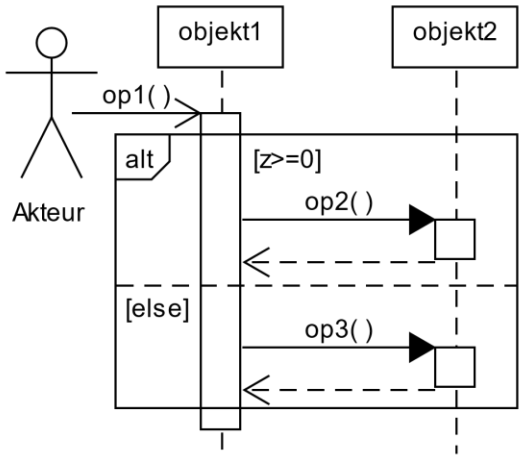




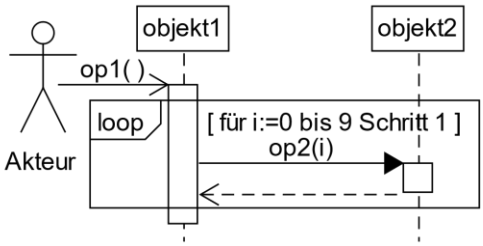
Option – einseitige Verzweigung



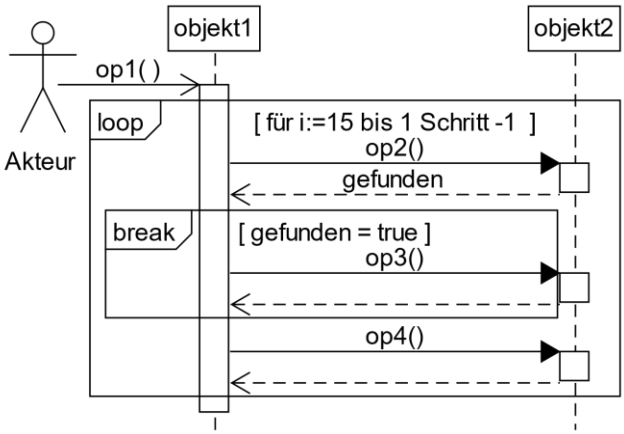
Alternative – mehrseitige Verzweigung



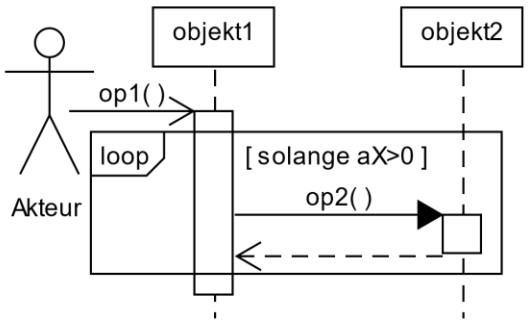
Zählschleife



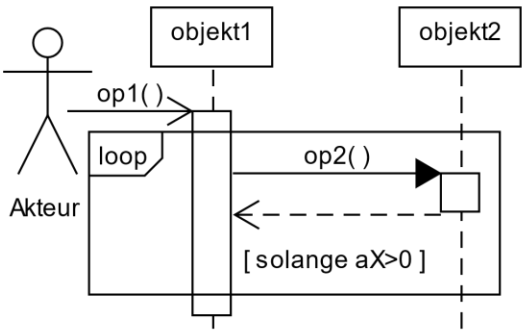
Schleife mit Abbruch



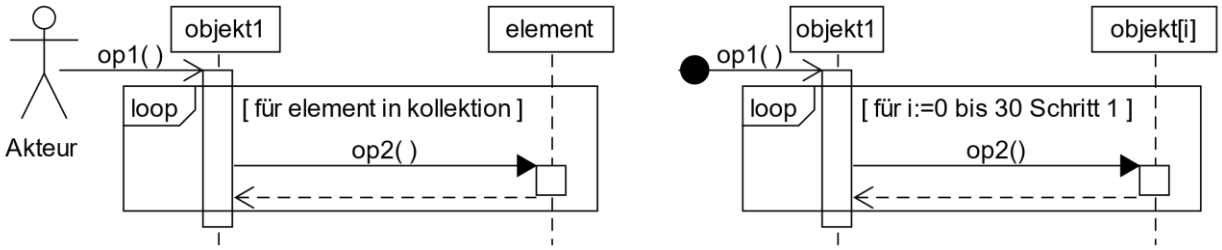
Kopfgesteuerte Schleife

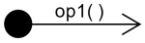


Fußgesteuerte Schleife



Schleife über Kollektion





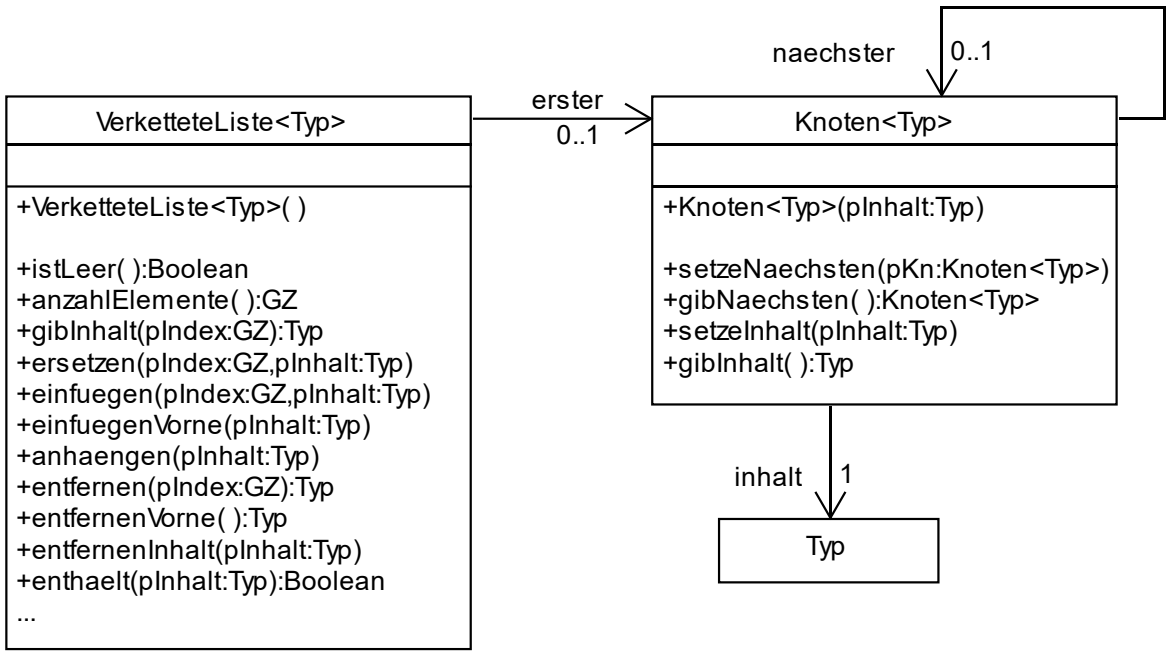
Nachricht, bei welcher der Sender nicht spezifiziert ist.

3.9 Zustandsdiagramme

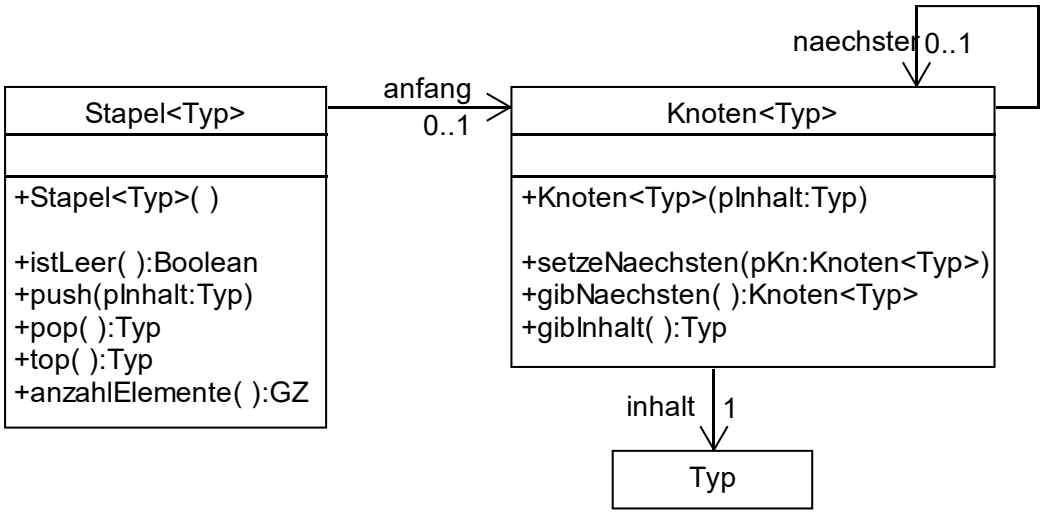
Zustandsdiagramme siehe Kapitel 1

4 Datenstrukturen

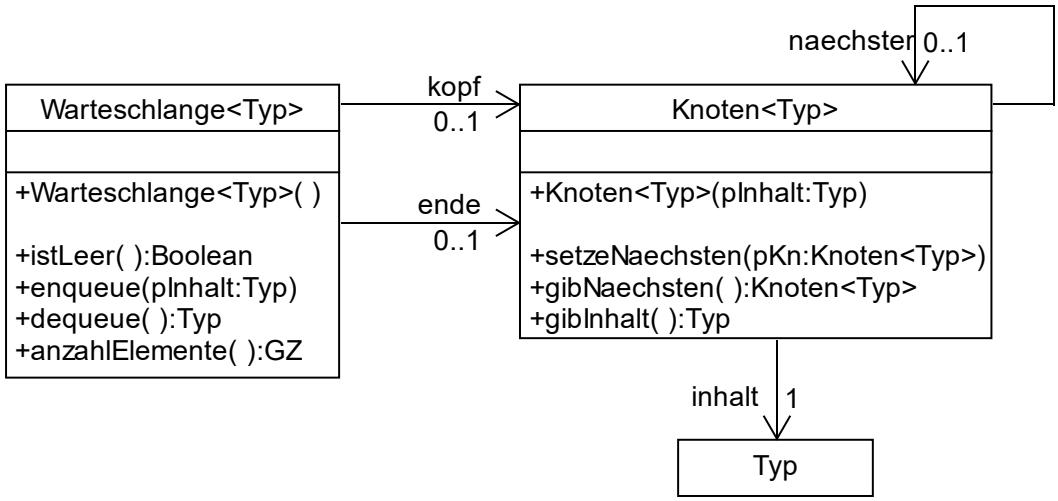
4.1 Verkettete Liste



4.2 Stapel

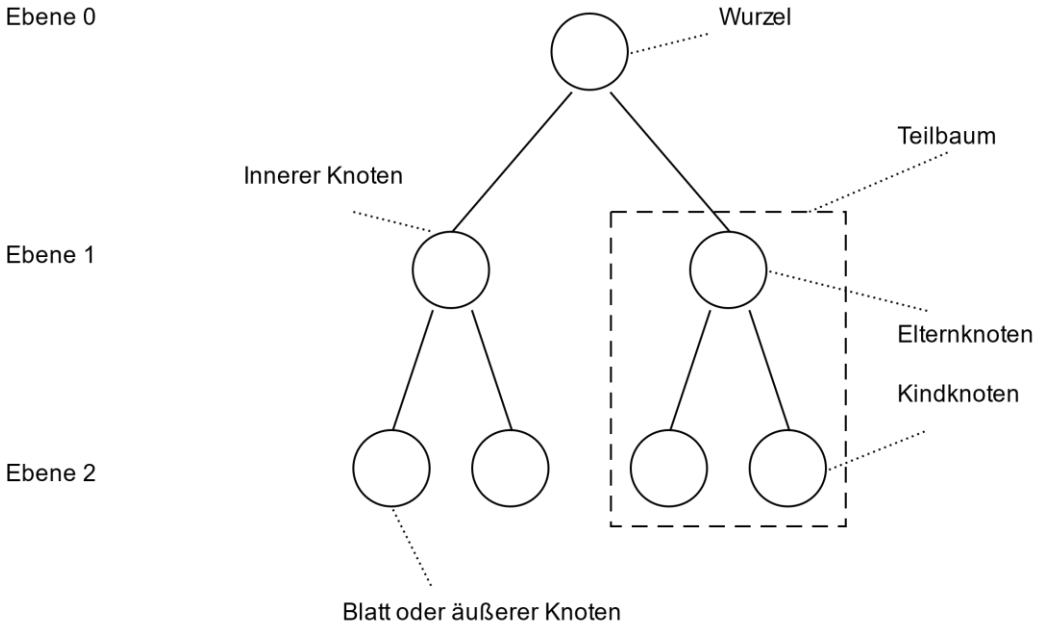


4.3 Warteschlange

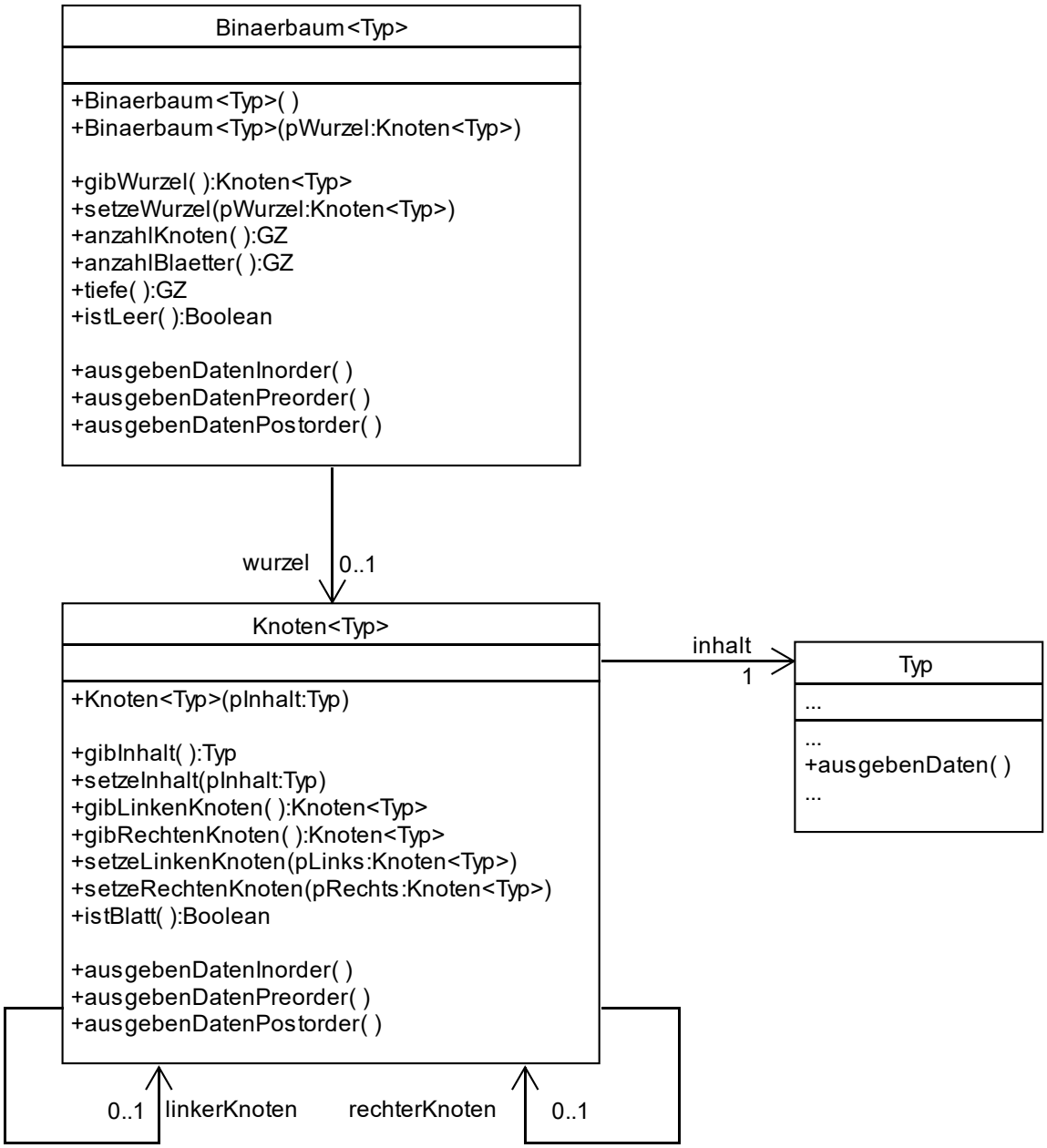


4.4 Binärbaum

4.4.1 Beispiel für einen Binärbaum der Tiefe 2



4.4.2 Datenstruktur



4.4.3 Operation `ausgebenDatenInorder()` der Klasse `Knoten` in Pseudocode

`OPERATION ausgebenDatenInorder() der Klasse Knoten`

```
WENN linkerKnoten != NICHTS
   linkerKnoten.ausgebenDatenInorder()
ENDE WENN
inhalt.ausgebenDaten()
WENN rechterKnoten != NICHTS
   rechterKnoten.ausgebenDatenInorder()
ENDE WENN
```

5 Künstliche Intelligenz

5.1 Klassifikation

Distanzfunktionen für $P(p_1|\ldots|p_n)$ und $Q=(q_1|\ldots|q_n)$

- Euklidische Distanz $d(P,Q)=\sqrt{\sum_{i=1}^{n}(p_i\text{-}q_i)^2}$
- Manhattan-Distanz $d(P,Q)=\sum_{i=1}^{n}|p_i\text{-}q_i|$
- Maximum-Distanz $d(P,Q)=max(|p_i\text{-}q_i|)$

5.2 Gini-Unreinheit

Für eine (ausgewählte) Menge von Datensätzen D und einem Ziel-Feature mit k möglichen Ausprägungen ist die **Gini-Unreinheit** (auch: Gini-Koeffizient, Gini-Index, Gini Impurity) wie folgt definiert:

$$Gini(D)=1\text{-}\sum_{i=1}^{k}(p_i)^2$$

wobei p_i die relative Häufigkeit der i-ten Ausprägung des Ziel-Merkmals ist.

Mit $Gini(F=v)$ bezeichnen wir die Gini-Unreinheit der Auswahl von Datensätzen, bei denen das Merkmal/Feature F den Wert v hat.

Ein Feature F kann verschiedene Werte $v \in V_f$ annehmen. Tritt ein bestimmter Wert v mit der relativen Häufigkeit p_v auf, dann berechnet sich die **gewichtete Gini-Unreinheit** für das Feature F folgendermaßen:

$$Gini(F)=\sum_{v\in V_F} p_v \cdot Gini(F=v)$$

5.3 Normalisierung von Daten

Normalisierung eines Werts x, d.h. Abbildung in den Wertebereich $[0;1]$:

$$x_{norm}=\frac{x\text{-}x_{min}}{x_{max}\text{-}x_{min}}$$

wobei x_{min} der kleinste und x_{max} der größte Wert des entsprechenden Merkmals ist.

Bsp. Werte: $7; 1; \text{-}3; 12; 0; 4$ $\rightarrow x_{max}=12;\ x_{min}=\text{-}3$
Normalisierte Werte: $\frac{2}{3}; \frac{4}{15}; 0; 1; \frac{1}{5}; \frac{7}{15}$

5.4 Neuronale Netze

5.4.1 Das (einlagige) Perzeptron

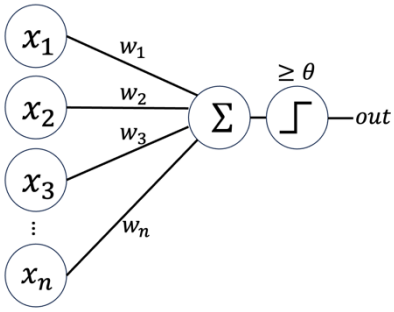
Klassische Darstellung

Einlagiges Perzeptron mit Inputs $x_1, x_2, \ldots, x_n$ und dazugehörigen Gewichten $w_1, w_2, \ldots, w_n$.
Wenn die gewichtete Summe

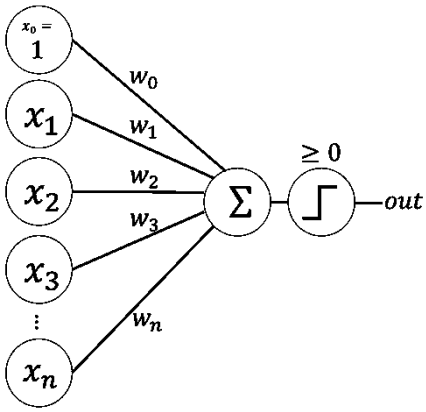
$$z = w_1 \cdot x_1 + \ldots + w_n \cdot x_n = \sum_{i=1}^{n} w_i \cdot x_i$$

den Schwellwert θ erreicht, wird das Neuron aktiv („feuert“).
Als Formel:

$$out = \begin{cases} 1, & wenn\ z \geq \theta \\ 0, & sonst \end{cases}$$



Alternative Darstellung mit Bias-Neuron



Fester Schwellwert 0, aber zusätzliches Pseudo-Inputneuron x_0, das immer den Wert 1 hat. Das Gewicht w_0 nennt man *Bias* (Variablenname deshalb oft auch b). Es gilt

$$w_0 = b = -\theta$$

d.h. das Gewicht am Bias-Neuron entspricht dem bisherigen Schwellwert (nur negiert, da jetzt auf der anderen Seite der Ungleichung):

$$out = \begin{cases} 1, & wenn\ z \geq 0 \\ 0, & sonst \end{cases}$$

$z = \sum_{i=0}^{n} w_i \cdot x_i$ ist wieder die gewichtete Summe der Inputs, jetzt aber inkl. Bias-Neuron $x_0 = 1$ und Bias-Gewicht w_0.

5.4.2 Lernregel für das Perzeptron

Für konkrete Inputwerte $x_1, x_2, \ldots, x_n$ wird (laut den Trainingsdaten) der Zielwert (engl. target) t erwartet. Der wirkliche Output des Perzeptrons für diese Inputwerte ist out (s. voriger Abschnitt). Der *Fehler* beträgt dann

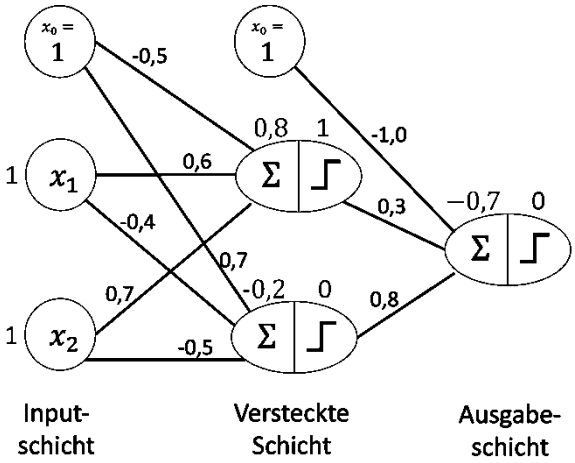
$$error = t - out$$

Beim Lernvorgang wird das zum *k*-ten Inputwert x_k gehörende Gewicht w_k wie folgt angepasst:

$$w_k \leftarrow w_k + \alpha \cdot error \cdot x_k$$

Dabei ist α die *Lernrate* ($\alpha > 0$).

5.4.3 Mehrlagiges Perzeptron



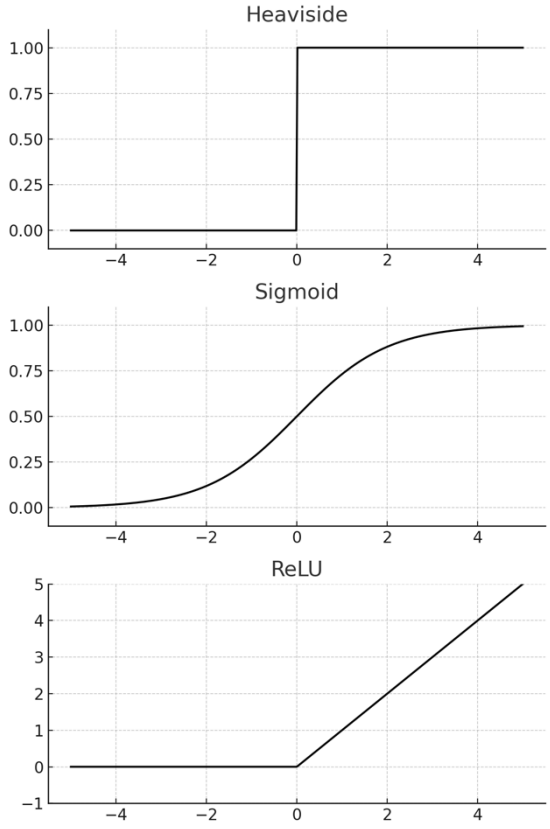
Das Outputneuron eines Perzeptrons kann als Inputneuron für ein anderes Perzeptron dienen. So entstehen *mehrlagige* Perzeptrons.

Die Abbildung zeigt ein *2-lagiges/2-schichtiges* Perzeptron (Input-Schicht wird nicht mitgezählt).

5.4.4 Aktivierungsfunktionen

Alle Arten künstlicher Neuronen haben gemeinsam, dass aus den Inputs x_i und den dazugehörigen Gewichten w_i eine gewichtete Summe $z = \sum_{i=0}^{n} w_i \cdot x_i$ berechnet wird (s. Abschnitt 5.4.1). Sie unterscheiden sich aber darin, wie aus dem Wert z danach der Output out des Neurons bestimmt wird. Dabei kommen unterschiedliche sogenannte *Aktivierungsfunktionen* zum Einsatz. Die in 6.4.1 verwendete Aktivierungsfunktion nennt man auch *Heaviside*- oder Treppenfunktion:

$$H(z) = \begin{cases} 1, & wenn\ z \geq 0 \\ 0, & sonst \end{cases}$$



Alternativen sind die *Sigmoid*-Funktion:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

oder die *ReLU*-Funktion:

$$ReLU(z) = \begin{cases} z, & wenn\ z \geq 0 \\ 0, & sonst \end{cases}$$

alternativ: $ReLU(z) = max(0, z)$

Die *Softmax*-Funktion ist eine Besonderheit unter den Aktivierungsfunktionen, die oft in der letzten Schicht eines neuronalen Netzes verwendet wird, um die Ausgaben in Wahrscheinlichkeiten umzuwandeln. Für das k-te der n Neuronen in dieser Schicht wird die gewichtete Summe z_k mit der folgenden Formel mit den anderen Neuronen in dieser Schicht in Beziehung gesetzt:

$$Softmax(z_k) = \frac{e^{z_k}}{\sum_{i=1}^{n} e^{z_i}}$$

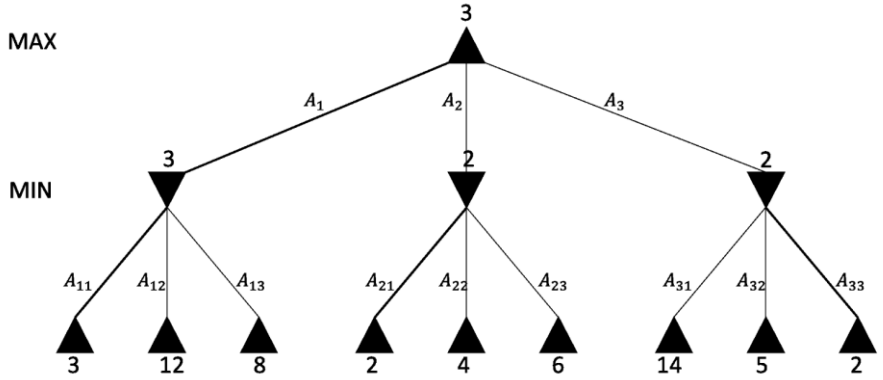
Durch diese Normierung ergeben sich für alle z_k Ausgabewerte, die zwischen 0 und 1 liegen und zusammen 1 ergeben, was eine Interpretation als Wahrscheinlichkeit ermöglicht.

Beispiel: In der letzten Schicht eines neuronalen Netzes befinden sich 5 Neuronen. Die gewichteten Summen der Inputs ergeben z.B. die Werte z_1=-2, z_2=1,3, z_3=0, z_4=-1, z_5=0,4. Für die Softmax-Funktion ergibt sich in diesem Fall die folgende Wertetabelle:

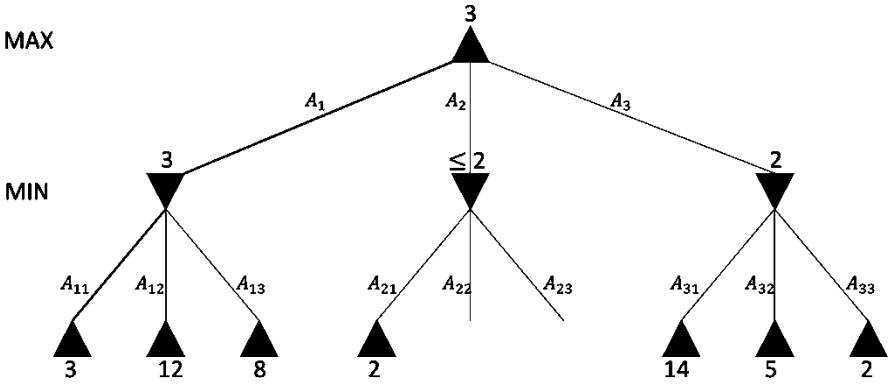
k	1	2	3	4	5
z_k	-2	1,3	0	-1	0,4
$Softmax(z_k)$	0,020	0,551	0,150	0,055	0,224

5.5 Minimax und Alpha-Beta-Pruning

Minimax-Algorithmus: Beispiel für die Zugfindung des MAX-Spielers:

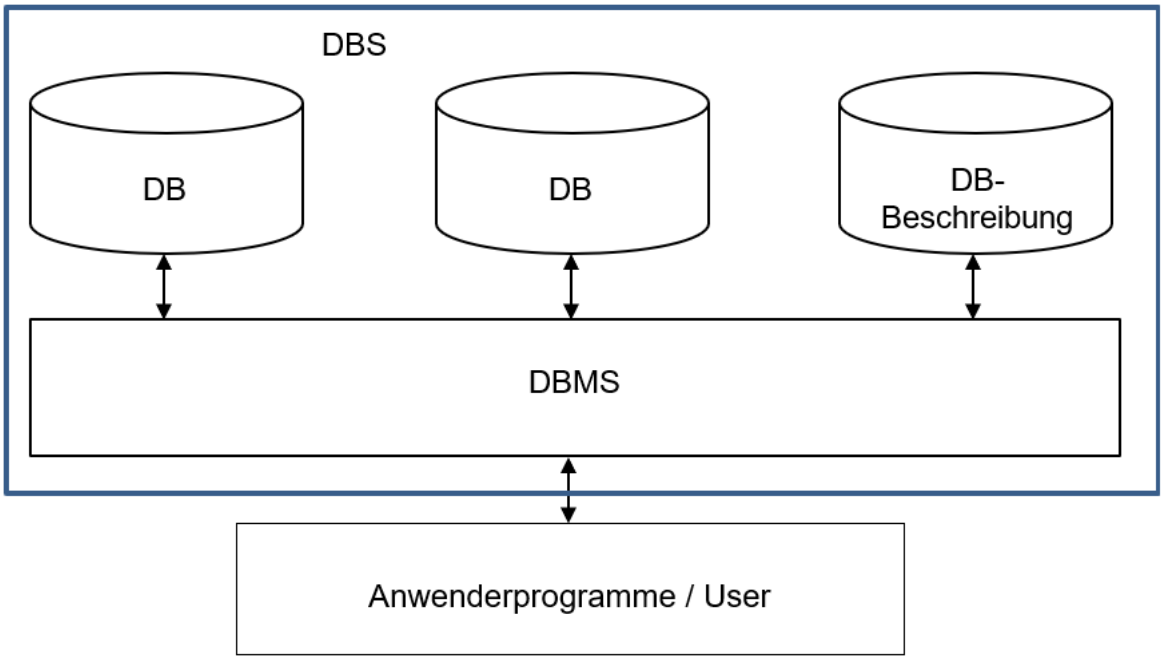


Dasselbe Spiel mit Alpha-Beta-Pruning:



6 Datenbanken

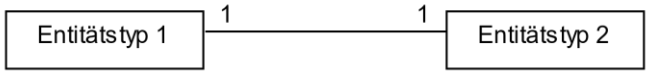
6.1 Datenbankmanagementsystem



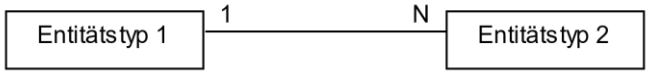
DBS = Datenbanksystem DBMS = Datenbankmanagementsystem DB = Datenbank

6.2 Entity-Relationship-Diagramm (ER-Diagramm)

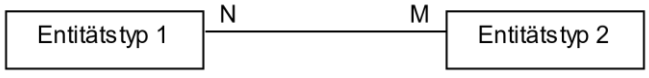
1:1 Beziehung



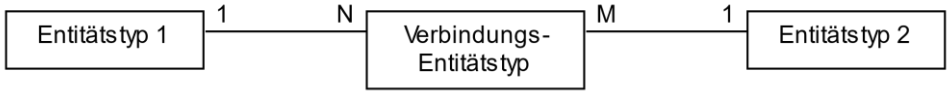
1:N Beziehung



N:M Beziehung



N:M Beziehung aufgelöst



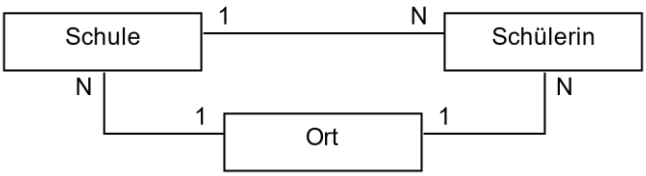
6.3 Relationenmodell

Alle Entitätstypen des Entity-Relationship-Diagramms mit Primär- und Fremdschlüsseln und allen Attributen der Entitätstypen in folgender Form:

`Entitätstyp(Primärschlüssel, Attribut1, Attribut2, …, Fremdschlüssel1, …)`

Beispiel: Schülerinnen, die ein Mädchengymnasium besuchen.

Entity-Relationship-Diagramm



Relationenmodell

```
Ort(OrtsNr, PLZ, Name)
Schule(SchulNr, Schulname, Straße, OrtsNr)
Schülerin(SchuelerinNr, Vorname, Name, Straße, OrtsNr, SchulNr)
```

6.4 Abfrageformulierung mit SQL

6.4.1 Projektion und Formatierung

Auswahl aller Spalten einer Tabelle

Syntax:
```
SELECT    *
FROM      <Tabelle>;
```

Auswahl mehrerer Spalten einer Tabelle

Syntax:
```
SELECT    <Spalte1>,<Spalte2>,<Spalte3>
FROM      <Tabelle>;
```

Auswahl ohne mehrfaches Auftreten derselben Zeile

Syntax:
```
SELECT DISTINCT    <Spalte>
FROM               <Tabelle>;
```

Umbenennen von Spalten bei der Ausgabe

Syntax:
```
SELECT    <Spalte> AS <neuer Spaltenname>
FROM      <Tabelle>;
```

Sortierung aufsteigend (`ASC` (optional)) oder absteigend (`DESC`)

Syntax:
```
SELECT        <Spalte>
FROM          <Tabelle>
ORDER BY      <Spalte> [ASC];

SELECT        <Spalte>
FROM          <Tabelle>
ORDER BY      <Spalte> DESC;
```

Beispiel Relationenmodell `Schüler (SID, Vorname, Name, Klasse)`

```
SELECT   *
FROM     Schüler
ORDER BY Name, Vorname;

SELECT   Vorname, Name
FROM     Schüler
ORDER BY Name ASC;

SELECT DISTINCT Klasse
FROM     Schüler
ORDER BY Klasse DESC;

SELECT Name AS "Nachname", Vorname
FROM   Schüler;
```

6.4.2 Selektion

Auswahl von Zeilen

Syntax:
```
SELECT    <Spalte>
FROM      <Tabelle>
WHERE     <Bedingung>;
```

Vergleichsoperatoren
```
=, <>, >, <, >=, <=          ( <> ungleich)
BETWEEN wert1 AND wert2
LIKE '_…%'  oder "_…%"       ( _ ein Zeichen
                               % beliebig viele Zeichen)
IN ('Wert1','Wert2') oder    IN ("Wert1","Wert2")
NOT IN ('Wert1','Wert2','Wert3')
IS NULL
IS NOT NULL
```

Logische Operatoren `AND, OR, NOT`

Beispiel Relationenmodell `Schüler (SID, Vorname, Name, Klasse)`

```
SELECT *
FROM Schüler

      Alle Schüler der TGI-J2
      WHERE Klasse = "TGI-J2";

      Alle Schüler der TG-Klassen
      WHERE Klasse LIKE 'TG%';

      Alle Schüler der TGI-Klassen
      WHERE Klasse IN ('TGI-E','TGI-J1','TGI-J2');

      Alle Schüler, die noch keiner Klasse zugeordnet sind
      WHERE Klasse IS NULL;
```

Beispiel Relationenmodell `Laborübung(LID, Thema, Dauer)`

```
SELECT *
FROM Laborübung
```

Alle Laborübungen die mindestens 60 und höchstens 90 Minuten ($60 \leq Dauer \leq 90$) gedauert haben
```
WHERE Dauer BETWEEN 60 AND 90;
```

Alle Laborübungen, deren Themen nichts mit Radioaktivität oder Atmosphärenchemie zu tun haben
```
WHERE Thema NOT IN ("Radioaktivität","Atmosphärenchemie");
```

Alle Laborübungen zur Organik, die kürzer als 60 Minuten waren
```
WHERE Thema = "Organik"
  AND Dauer < 60;
```

6.4.3 Verbund von Tabellen

Inner Join

Syntax:
```
SELECT   A.<Spalte1>,B.<Spalte2>
FROM     <Tabelle1> A INNER JOIN <Tabelle2> B
         ON A.<Spalte1> = B.<Spalte2>
```

Beispiel Relationenmodell
```
Schüler (SID, Vorname, Name, Klasse)
Teilnahme(TID, SID, Datum, Punkte)
Laborübung(LID, Thema, Dauer)
```

```
SELECT Vorname, Name, Datum, Punkte
FROM   Schüler INNER JOIN Teilnahme ON Schüler.SID = Teilnahme.SID;
```

Anmerkung: Tabellennamen können in FROM durch Aliase abgekürzt werden.

```
SELECT Vorname, Name, Datum, Punkte
FROM   Schüler S INNER JOIN Teilnahme T ON S.SID = T.SID;
```

```
SELECT Vorname, Name, Datum, Thema, Dauer
FROM   Schüler S
       INNER JOIN Teilnahme T ON S.SID = T.SID
       INNER JOIN Laborübung L ON L.LID = T.LID;
```

Equi-Join

Syntax:
```
SELECT   <Spalte1>,<Spalte2>
FROM     <Tabelle1>,<Tabelle2>
WHERE    <Join-Bedingung>;
```

In der Join-Bedingung wird festgelegt, dass der Inhalt bestimmter Spalten identisch sein muss.

Beispiel Relationenmodell
```
Schüler (SID, Vorname, Name, Klasse)
Teilnahme(TID, SID, LID, Datum, Punkte)
Laborübung(LID, Thema, Dauer)
```

```
SELECT Vorname, Name, Datum, Punkte
FROM   Schüler, Teilnahme
WHERE  Schüler.SID = Teilnahme.SID;
```

```
SELECT Vorname, Name, Datum, Thema, Dauer
FROM   Schüler S, Teilnahme T, Laborübung L
WHERE  S.SID = T.SID
  AND  L.LID = T.LID;
```

6.4.4 Aggregatfunktion

Aggregatfunktionen können auf einer ganzen Tabelle bzw. Zwischentabelle ausgeführt werden. Ihre Ergebnistabelle besteht dann aus einer Zelle.

Syntax:
```
SELECT    Aggregatfunktion(<Spalte>)
FROM      <Tabelle>;
```

SUM	Summierung der numerischen Werte in der Spalte
MIN	Minimum der Spalte
MAX	Maximum der Spalte
AVG	Durchschnitt der numerischen Werte in der Spalte
COUNT	Anzahl der Zeilen des Zwischenergebnisses

Hinweis: NULL-Werte werden vor der Auswertung einer Aggregatfunktion eliminiert.

Beispiel Relationenmodell
```
Schüler (SID, Vorname, Name, Klasse)
Teilnahme(TID, SID, Datum, Punkte)
```

Summe der von den Schülern der Klasse TGI-E am 24.07.2021 erreichten Punkte
```
SELECT SUM(Punkte) AS "Gesamtpunktzahl der Klasse TGI-E am 24.07.21"
FROM   Schüler S INNER JOIN Teilnahme T ON S.SID = T.SID
WHERE  Klasse = "TGI-E"
  AND  Datum = #24/07/2021#;
```

Maximal erreichte Punktezahl
```
SELECT MAX(Punkte) AS "Max. Punkte"
FROM   Teilnahme;
```

Datum der ersten Teilnahme, d.h. des ersten Termins der Veranstaltung
```
SELECT MIN(Datum) AS "Startdatum"
FROM   Teilnahme;
```

Punktedurchschnitt der Klasse TGI-E
```
SELECT AVG(Punkte) AS "Klassendurchschnitt TGI-E"
FROM   Schüler S INNER JOIN Teilnahme T ON S.SID = T.SID
WHERE  Klasse = "TGI-E";
```

Anzahl der Schüler in der Klasse TGI-E
```
SELECT COUNT(*) AS "Anzahl Schüler TGI-E"
FROM   Schüler
WHERE  Klasse = "TGI-E";
```

Spezialfall: COUNT(DISTINCT …)

Beispiel Relationenmodell
```
Schüler (SID, Vorname, Name, Klasse)
Teilnahme(TID, SID, Datum, Punkte)
```

Anzahl Klassen
```
SELECT COUNT(DISTINCT Klasse) AS "Anzahl Klassen"
FROM   Schüler S INNER JOIN Teilnahme T ON S.SID = T.SID;
```

6.4.5 Aggregatfunktion mit Gruppierung

Mit `GROUP BY` werden Abfrageergebnisse nach bestimmten Kriterien in Gruppen zusammengefasst. Auf jeder Gruppe wird einzeln die Aggregatfunktion ausgewertet und ein eigener Wert berechnet. Somit besteht die Ergebnistabelle aus den Aggregatwerten der einzelnen Gruppen.

Syntax:
```
SELECT     <Spalte1>, Aggregatfunktion(<Spalte2>) AS <Name>
FROM       <Tabelle>
GROUP BY   <Spalte1>;
```

Beispiel Relationenmodell
```
Schüler (SID, Vorname, Name, Klasse)
Teilnahme(TID, SID, Datum, Punkte)
```

Punktedurchschnitte pro Klasse
```
SELECT   Klasse, AVG(Punkte) AS "Gesamtpunktzahl pro Klasse"
FROM   Schüler S INNER JOIN Teilnahme T ON S.SID = T.SID
GROUP BY Klasse;
```

Beste Leistung pro Tag
```
SELECT   Datum, MAX(Punkte) AS "Bestes Tagesergebnis"
FROM     Teilnahme T
GROUP BY Datum;
```

6.4.6 Selektion von Gruppen

Im Unterschied zur einfachen Selektion mit `SELECT` können mit `HAVING` Abfrageergebnisse von Aggregatfunktionen auf Gruppen selektiert werden.

Syntax:
```
SELECT     <Spalte1>, Aggregatfunktion(<Spalte2>) AS <Name>
FROM       <Tabelle>
WHERE      <Bedingung>
GROUP BY   <Spalte1>
HAVING     <Bedingung für Aggregatfunktion>;
```

Beispiel Relationenmodell
```
Schüler (SID, Vorname, Name, Klasse)
Teilnahme(TID, SID, Datum, Punkte)
```

Punktedurchschnitte pro Klasse, aber nur wenn der Durchschnitt größer als 20 Punkte ist.
```
SELECT   Klasse, AVG(Punkte) AS "Gesamtpunktzahl pro Klasse"
FROM   Schüler S INNER JOIN Teilnahme T ON S.SID = T.SID
GROUP BY Klasse
HAVING   AVG(Punkte)>20;
```

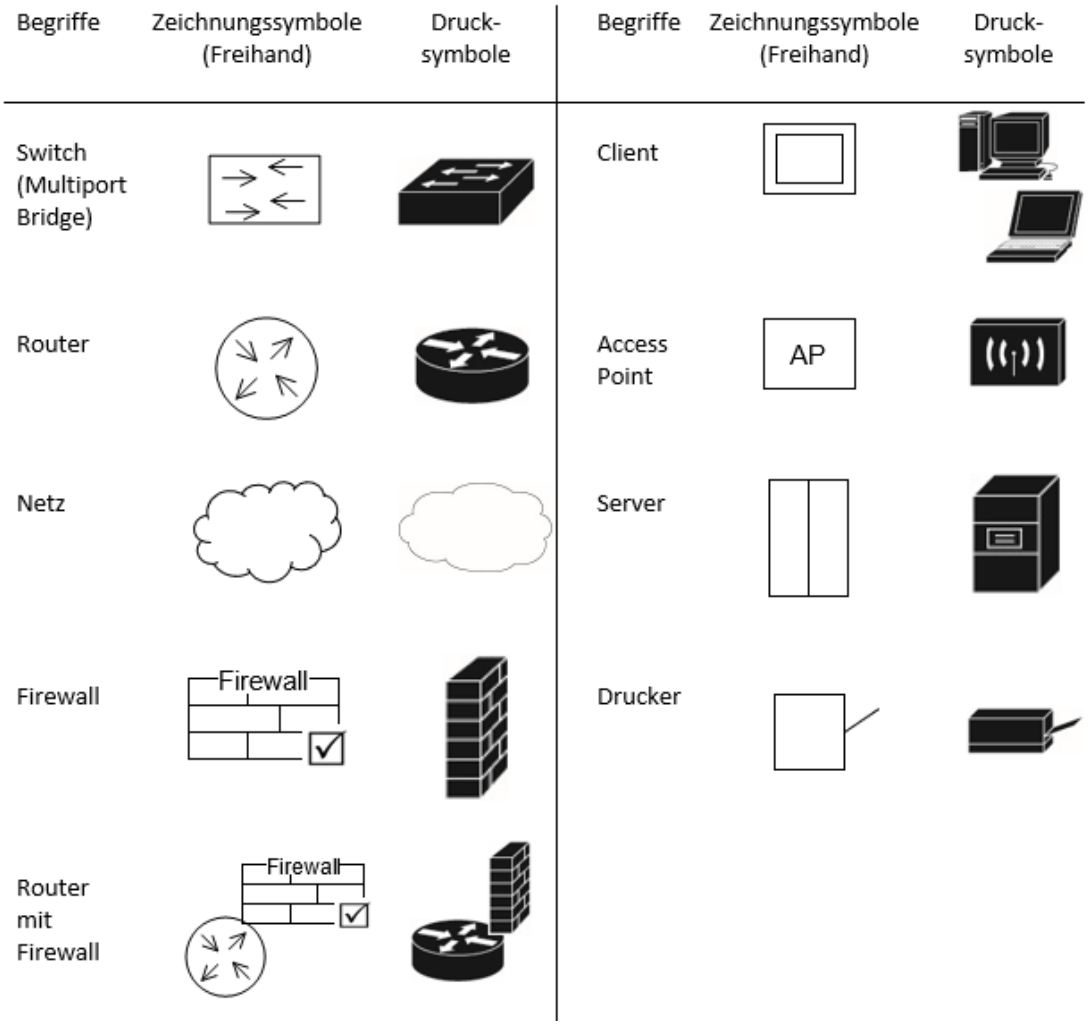
6.4.7 Komplette SQL-Anweisung

Syntax:
```
SELECT      …
FROM        …
WHERE       …
GROUP BY    …
HAVING      …
ORDER BY    … ;
```

7 Vernetzte Systeme

7.1 Netzwerktechnik

7.1.1 Netzwerksymbole



7.1.2 Routing-Tabelle (IPv4)

Die Routingtabelle des Router R_Extern (vgl. Abbildung Netzwerkplan) sieht folgendermaßen aus:

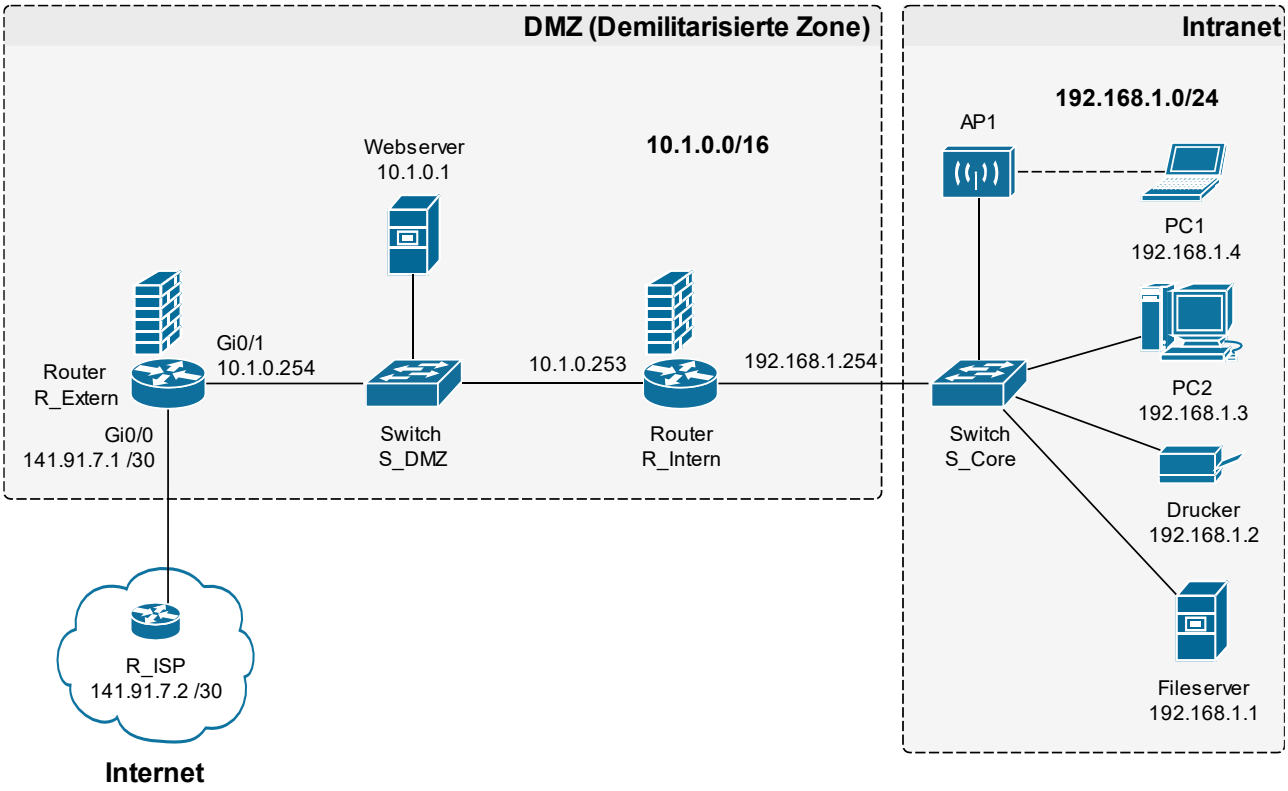
Netzadresse	Subnetzmaske	nächster Router	Ausgangs-Schnittstelle
141.91.7.0	/30	*	Gi0/0
10.1.0.0	/16	*	Gi0/1
192.168.1.0	/24	10.1.0.253	Gi0/1 *optional
0.0.0.0	0.0.0.0	141.91.7.2	Gi0/0 *optional

Hinweis:

Bezeichnung für die Ausgangs-Schnittstelle (Exit-Interface) ist plattformabhängig,

gängige Bezeichnungen sind: Gi0/0, Fa0/0, Eth0, …

Abbildung: Netzwerkplan



7.2 IP-Adressen

7.2.1 Aufbau IPv4-Adresse

IP-Adresse (dotted-decimal-format): z.B. 177 . 17 . 223 . 1
IP-Adresse (binär): 10110001.00010001.11011111.00000001

8 Bit = 1 Oktett

32 Bit = 4 Bytes

IP-Adresse z.B.	192.168. 1 . 1	➔ 11000000.10101000.00000001.00000001
Netzmaske z.B. /24 =	255.255.255. 0	➔ 11111111.11111111.11111111.00000000
Netz-ID	**192.168. 1** . 0	← **11000000.10101000.00000001**.00000000
Host-ID	0 . 0 . 0 . **1**	← 00000000.00000000.00000000.**00000001**

Alle Host-ID-Bits = 0: Netz-Adresse, hier 192.168.1.0
Alle Host-ID-Bits = 1: Broadcast-Adresse, hier 192.168.1.255

7.2.2 Aufbau IPv6-Adresse

IP-Adresse (hexadezimal): z.B. 2001:07c0:8280:0253:0000:0000:0000:0020

16 Bit

8 Blöcke (16 Bit) = 128 Bit

Weitere IPv6-Schreibweise:

Führende Nullen können ausgelassen werden	2001:7c0:8280:253:0:0:0:20
Aufeinanderfolgende Null-Blöcke können durch zwei Doppelpunkte einmal ersetzt werden	2001:7c0:8280:253::20

Adressformat:

64 Bits	64 Bits
Netzwerk Präfix	Interface Identifier (IID)

48 Bits	16 Bits
Global Routing Präfix	Subnetz ID

Netzwerk-Präfix:

2001:07c0:8280:0253 → Global Routing Präfix

2001:07c0:8280**:0253** → Subnetz Identifier

Interface Identifier:

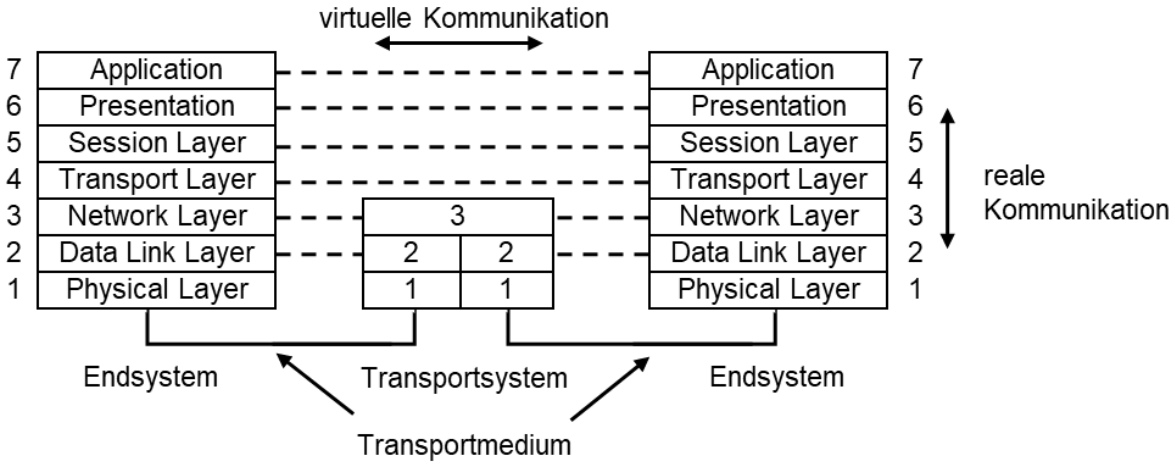
xxxx:xxxx:xxxx:xxxx:**0000:0000:0000:0020**

7.2.3 IP-Adressarten

IPv6	IPv4
Öffentliche Adressen (-bereiche)	
Global Unicast Address (GUA) 2000:... bis 3fff:... (von der IANA derzeit vergeben) z.B. 2003:c4:4731:f000:245b:3f0b:5bfb:c31e	öffentlicher IPv4-Adressbereich 0.0.0.0 – 223.255.255.255 (ohne die unten aufgeführten Adressen)
Link Local – Adressen (-bereiche)	
fe80::/64 z.B. fe80::ca0e:14ff:fe79:7e10	169.254.0.0/16
Loopback – Adressen (-bereiche)	
::1/128	127.0.0.0/8 z.B. 127.0.0.1/8 (localhost)
Private Adressen (-bereiche)	
Unique Local Address (ULA) fc00::/7 bis fdff::/7, z.B. fd00::1/64	• 10.0.0.0/8 • 172.16.0.0/12 • 192.168.0.0/16
Standard-Route (default route)	
::/0	0.0.0.0/0

7.3 Schichtenmodelle

7.3.1 ISO-OSI-7-Schichtenmodell



7.3.2 TCP-IP-Schichtenmodell

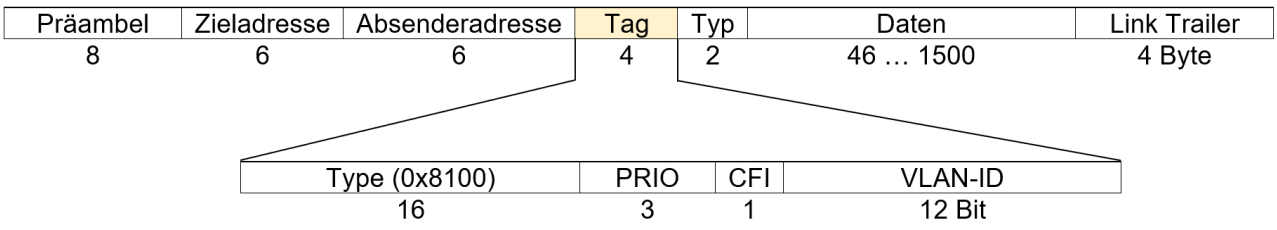
OSI-Schicht		TCP/IP-Schicht	PDU-Begriffe	Protokoll-Beispiele
7	Application Layer	Application	Data	HTTP, FTP, SMTP, Telnet, DHCP, MQTT, … TLS
6	Presentation Layer			
5	Session Layer			
4	Transport Layer	Transport	Segment, Datagram	TCP, UDP
3	Network Layer	Internet	Packet	IP (IPv4, IPv6), ICMP
2	Data Link Layer	Network Access	Frame	Ethernet
1	Physical Layer		Bits, Bitstream	

7.4 Header

7.4.1 Ethernet II

Präambel	Zieladresse	Absenderadresse	Typ	Daten	Link Trailer
8	6	6	2	46 … 1500	4 Byte

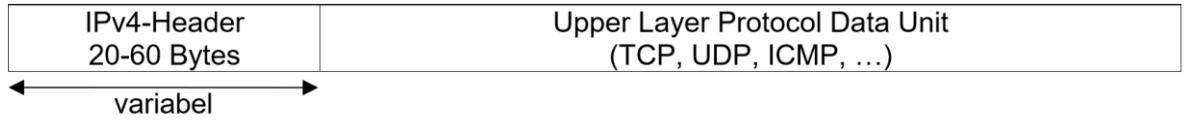
7.4.2 IEEE 802.1Q



7.4.3 IPv4-Header

Byte	Inhalt							
0	Version				IHL			
1	TOS							
2-3	Paketlänge							
4-5	Identifikation							
6	Flags			Fragmentabstand				
7	Fragmentabstand							
8	Time To Live (TTL)							
9	Protokoll							
10-11	Kopf-Prüfsumme							
12-15	IP-Sendeadresse							
16-19	IP-Empfängeradresse							
20 ...	Optionen (mit evtl. Füllzeichen)							

IPv4-Paketstruktur:



7.4.4 IPv6-Header

Byte	Inhalt					
0-3	Version	Traffic Class		Flow Label		
4-7	Payload Length		Next Header		Hop Limit	
8-23	Source Address					
24-39	Destination Address					

IPv6-Paketstruktur:

7.4.5 TCP-Header

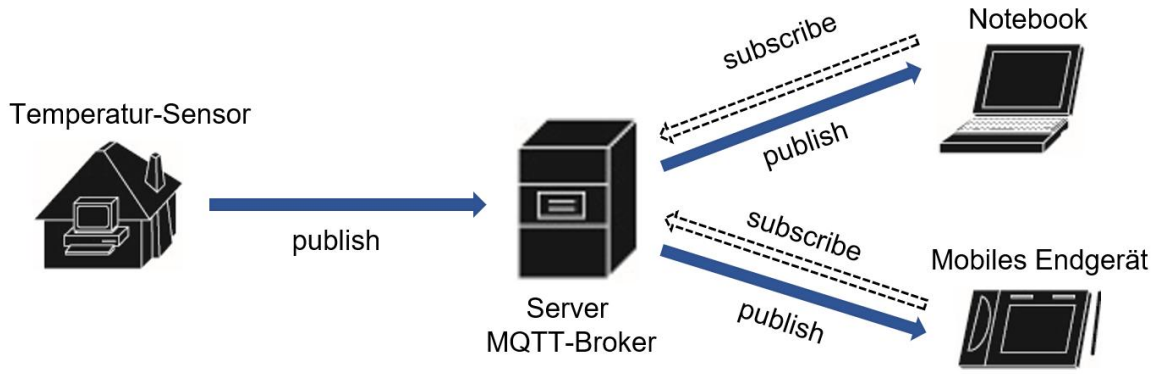
Byte	Inhalt							
0-1	Source Port							
2-3	Destination Port							
4-7	Sequenznummer							
8-11	Quittungsfeld (Piggyback, Acknowledgement Number)							
12	Header-Länge				reserviert			
13	reserviert	URG	ACK	PSH	RST	SYN	FIN	
14-15	Fenstergröße							
16-17	Prüfsumme							
18-19	Urgent Zeiger							
20 ...	Optionen (evtl. mit Füllzeichen)							

7.4.6 UDP-Header

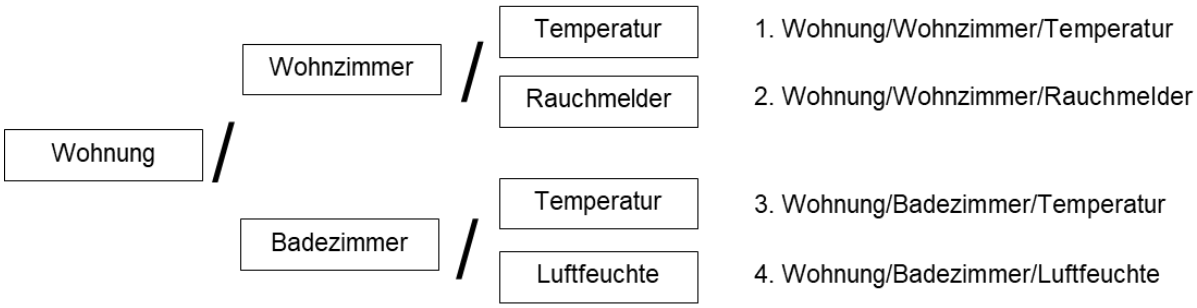
Byte	Inhalt
0-1	Source Port
2-3	Destination Port
4-5	Länge des Datagramms
6-7	Check-Summe

7.5 Internet der Dinge (IoT)

7.5.1 MQTT-Protokoll (Message Queuing Telemetry Transport)

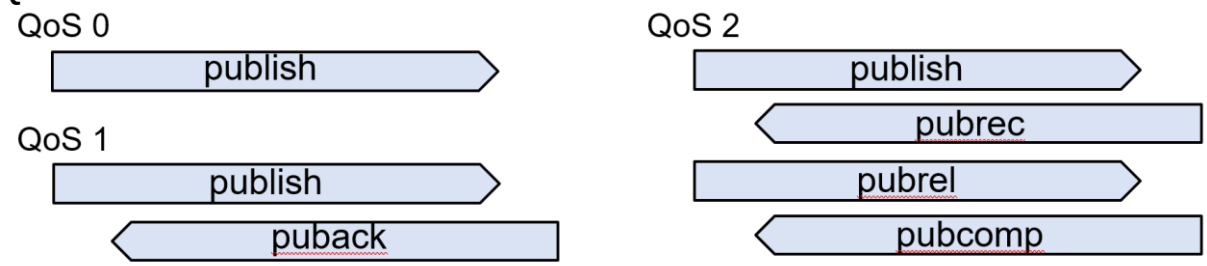


Topicstruktur:



Multi-Level-Wildcard: **#** z.B. Wohnung/Wohnzimmer/#
Single-Level-Wildcard: **+** z.B. Wohnung/+/Temperatur

Qualitätsstandards:



Ports:

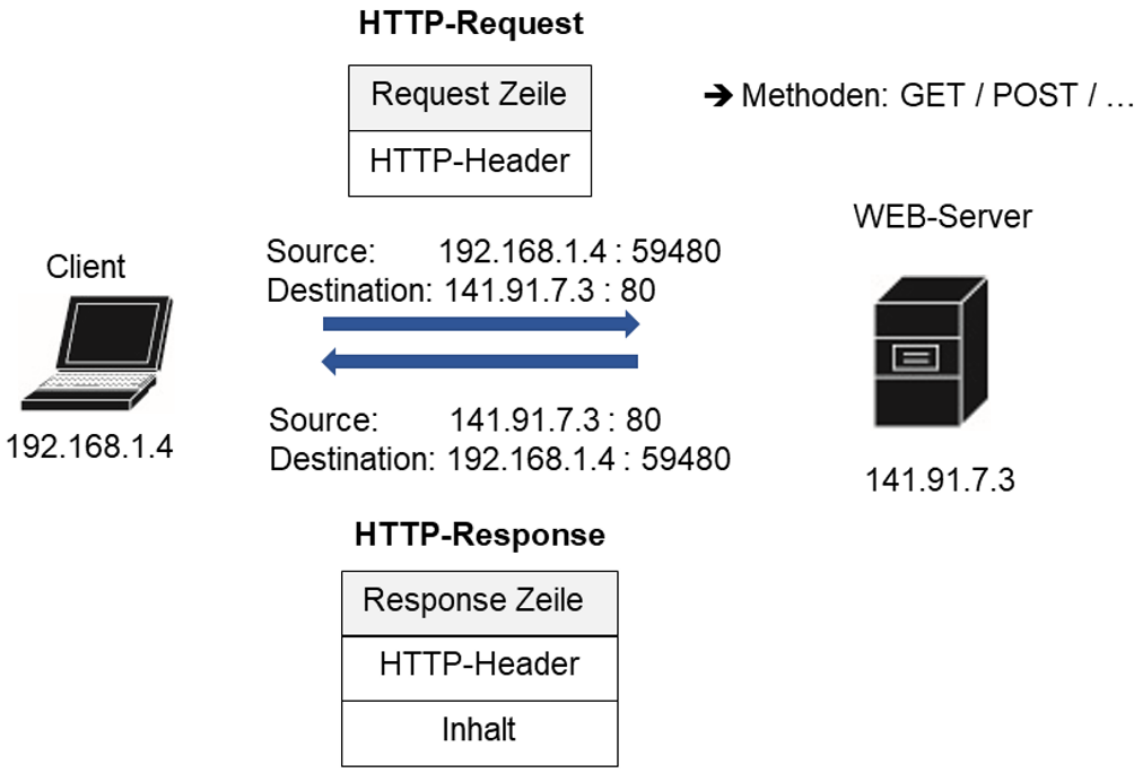
1883 : MQTT, unverschlüsselt
8883 : MQTT, verschlüsselt
8884 : MQTT, verschlüsselt, Client Zertifikat notwendig
8080 : MQTT über WebSockets, unverschlüsselt

MQTT –Header: (Beispiel - Publish Message)

Byte	Inhalt			
0	Nachrichtentyp (4 Bit)	Dup-Flag (1 Bit)	Quality of Service (2 Bit)	Retain-Flag (1 Bit)
1	Länge des restlichen MQTT-Pakets			
…	MQTT-Topic → Topic-Länge / Topic / Payload			

7.5.2 HTTP-Protokoll (Hypertext Transfer Protocol)

Kommunikationsprinzip:



URL (Uniform Resource Locator):

Protokoll	Domain	Pfad
https://	gsoe.de	/bildungsangebote/technisches-gymnasium/

Ports:

80 : HTTP, unverschlüsselt
443 : HTTPS, verschlüsselt

Request HTTP 1/1

Methode	Pfad	Protokoll
GET	/wp/content/uploads/2020/11/pixels-fauxels.jpg	HTTP/1.1\r\n

HTTP-Header - Name: Wert (Beispiele)	
Host:	Domain-Name des Servers
User-Agent:	User-Agent des Clients
Accept:	Welche Inhaltstypen der Client verarbeiten kann
z.B.	• Accept-Charset: → Welche Zeichensätze der Client anzeigen kann. • Accept-Encoding: → Welche komprimierten Formate der Client unterstützt. • Accept-Language: → Gewünschte Sprachversion
Date:	Datum und Zeit des Requests
Connection:	Bevorzugte Art der Verbindung
Referrer:	URL der Ressource, von der aus verlinkt wurde.
Content-Length:	Länge des Request-Bodys
Content-Type:	MIME-Typ des Bodys (bei POST- und PUT-Requests)

Response HTTP 1/1

Protokoll	Status-Code
HTTP/1.1	200 OK\r\n

HTTP-Header - Name: Wert (Beispiele)	
Date:	Zeitpunkt der Response
Server:	Kennung des Servers
Accept-Ranges:	Welche Einheiten der Server akzeptiert
Allow:	Erlaubte Request-Typen (Methoden)
Connection:	Bevorzugte Art der Verbindung

Status-Codes	(Beispiele)
100 … 199:	Information
200 … 299:	Client-Anfrage erfolgreich **z.B. 200 – OK**
300 … 399:	Client-Anfrage umgeleitet **z.B. 301 – Moved Permanently** **302 – Moved Temporarily**
400 … 499:	Fehlen des Dokuments **z.B. 403 – Forbidden** **404 – Not Found**
500 … 599:	Serverfehler