Info Abi

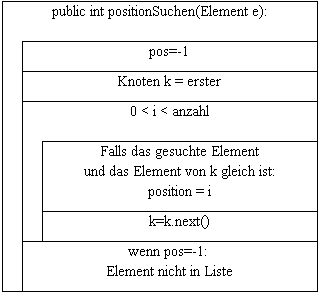
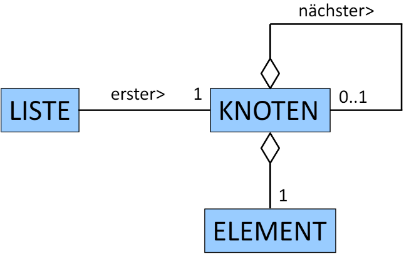
Fragen zum Stoff:

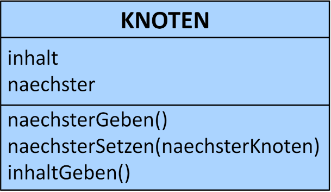
* Datenknoten entfernen in Binärbäumen nur als Pseudocode ungefähr?
* Hierarchische Bäume und B-Bäume relevant?
* Sortieralgorithmen relevant?

Vorallem bei HJ3 Formale Sprachen: Buch texte anschauen; Turing Machine?

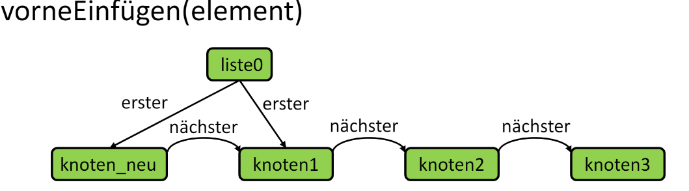
# HJ 1

## Basics

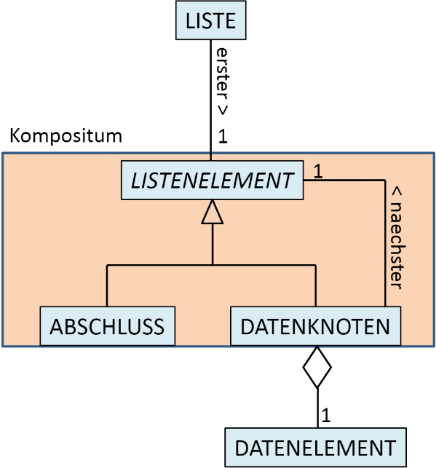
1. **Einfach verkettete Liste:** (Eine dynamische Datenstruktur -> Länge variabel)



* Iterative Lösung

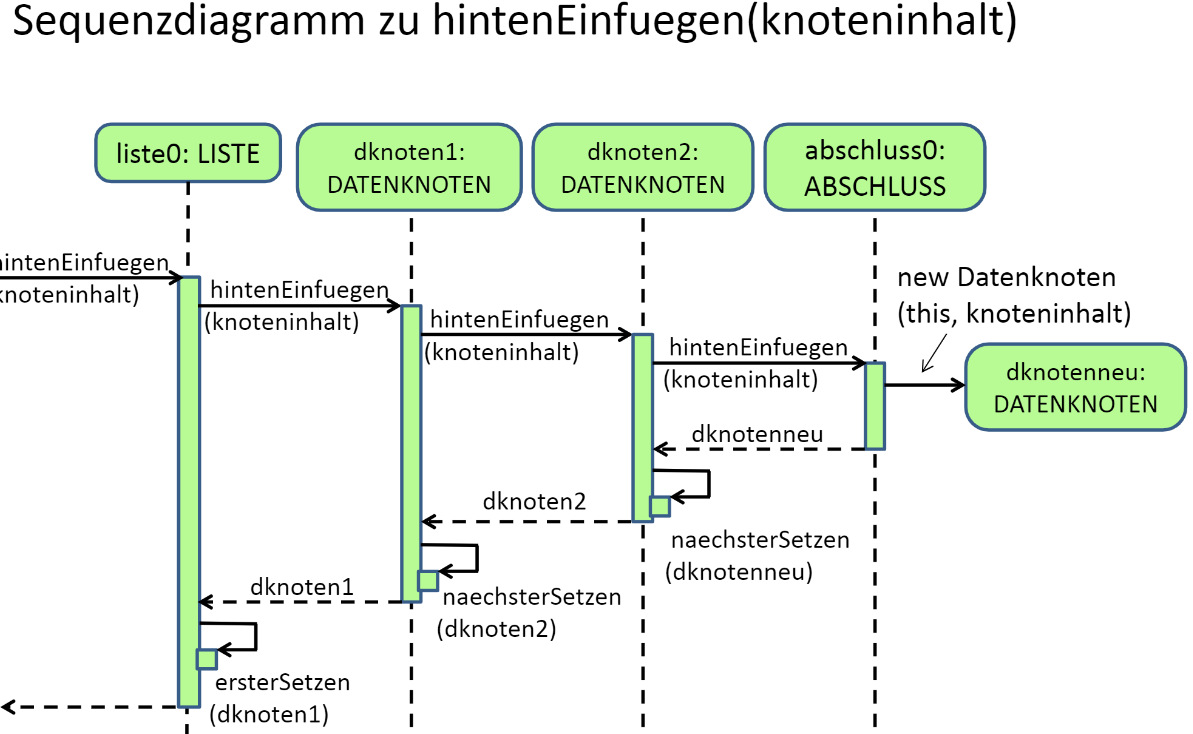


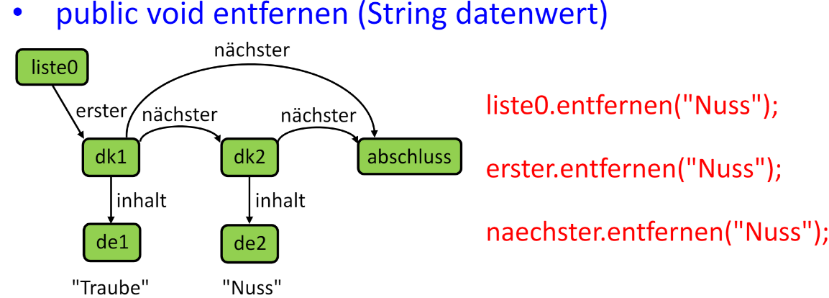
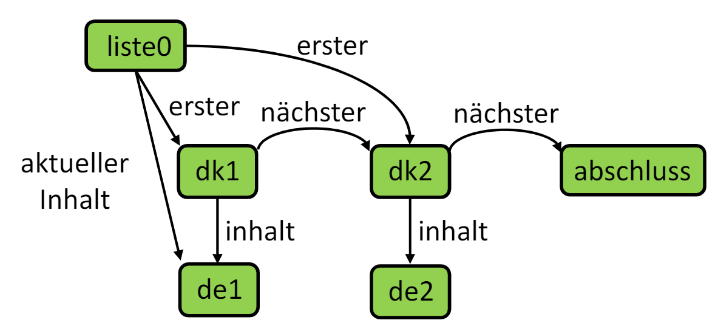
1. **Rekursive Liste:**

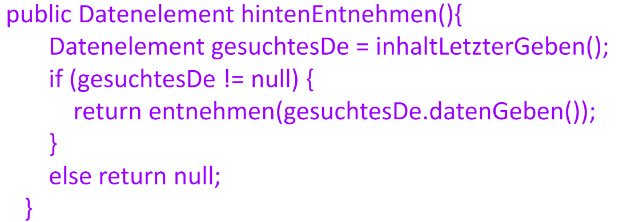
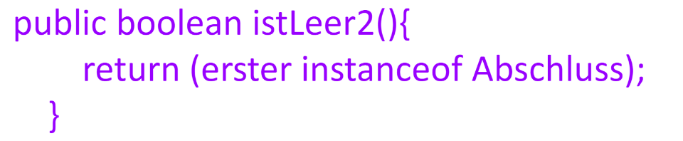


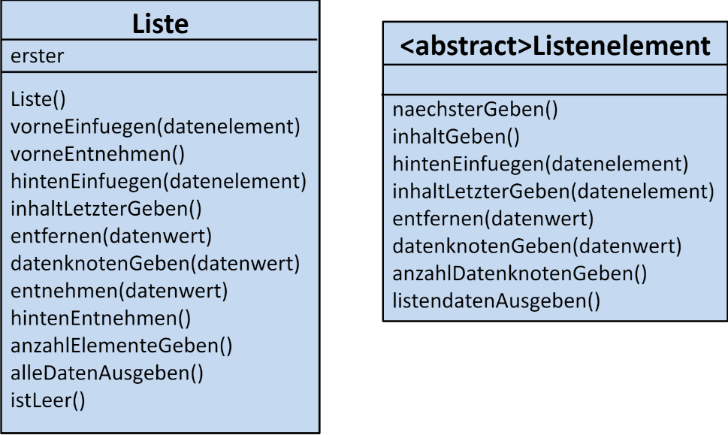
Alle Knoten bis auf den letzten verhalten sich gleich

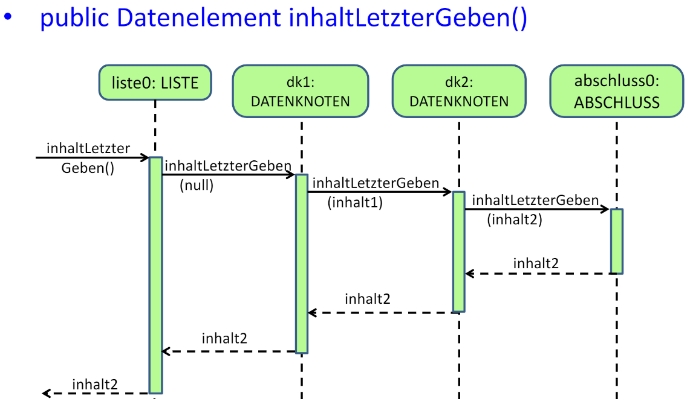
* Knoten reichen Inhalte meist nur an den nächsten weiter
* Arbeit macht oft der Abschluss



public Datenelement vorneEntnehmen()



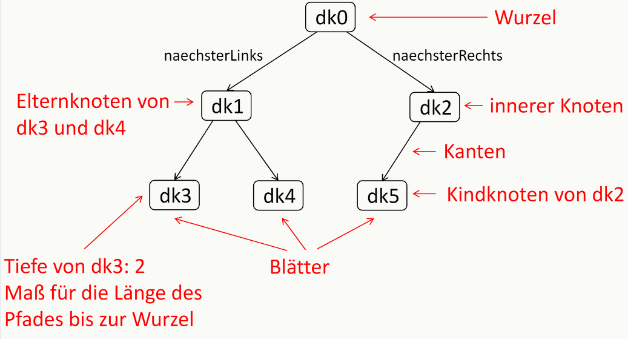




1. **Binärbäume:**

Eine Baumstruktur entsteht wenn jedes Objekt mehrere Nachfolger hat

* Genau ein Objekt wird innerhalb der Baumstruktur **nicht** referenziert (Wurzel)
* Von der Wurzel kann jedes Objekt über genau einen Weg erreicht werden

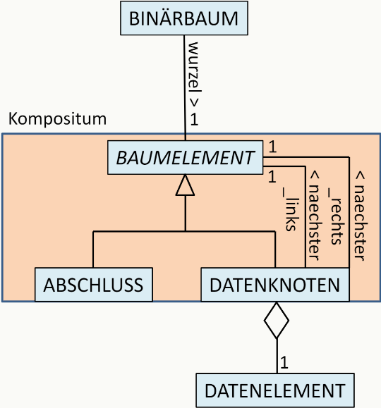
Bei Binärbäumen: Jedes Objekt hat **höchstens zwei** Nachfolger

Traversieren: Jeder Knoten muss 3 Aktionen ausführen:  
 sich selbst ausgeben (w), weiterreichen an den linken Teilbaum (LK) und weiterreichen an den rechten Teilbaum (RK)

Preorder: w-LK-RK Blätter sind ein Abschluss oder

Inorder: LK-w-RK haben zwei null Referenzen als

Postorder: LK-RK-w Nachfolger

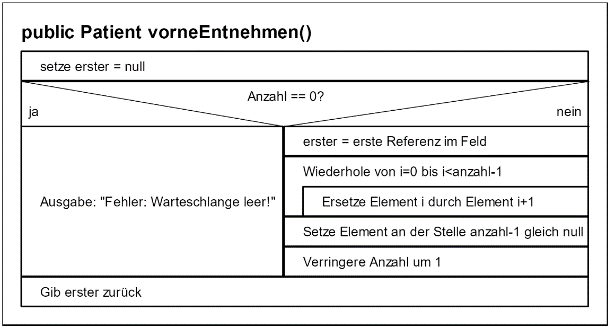
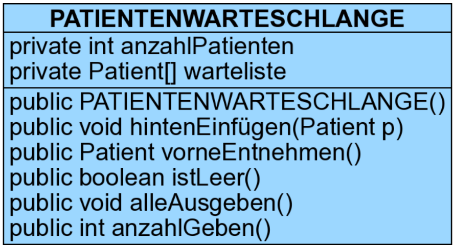


Kompositum ähnlich wie   
  
bei Liste

## HJ1\_1 Die Datenstrukturen „Warteschlange“ und „Stapel“

1. **Warteschlangen (queue)**

zb in Rechnernetzen die viele Anfragen bekommen

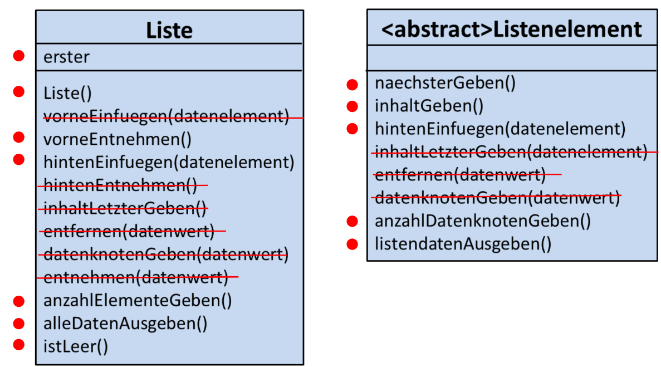
Grundprinzipien: FIFO (first in first out) -> Elemente in der Reihenfolge entnehmen, wie sie eingefügt wurden

Umsetzung zb mit einem Array:

* Problem:

Feldlänge statisch  
Nachrutschen bei entnehmen

* Lösung: Dynamische Datenstrukturen wie Listen:

Iterative Lösung mit einfach verketteter Liste aber besser: **Rekursive Liste**

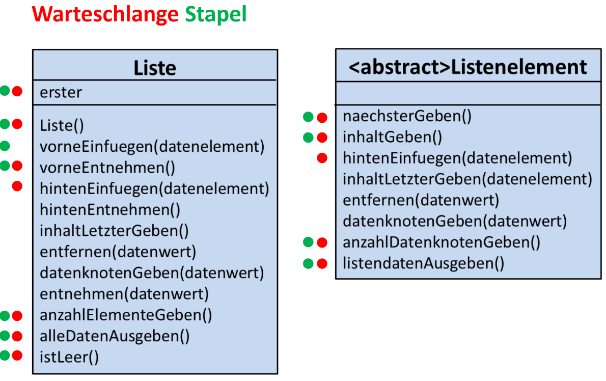
-> benötigte Einfügemethoden: hintenEinfügen, vorneEntnehmen **oder** hintenEntnehmen, vorneEinfügen

-> vorneEntnehmen & hintenEinfügen bei rekursiver Liste aber einfacher

1. **Stapel (stack)**

zb bei Solitaire; nur die oberste, zuletzt hingelegte Karte ist entnehmbar

Grundprinzipien: LIFO (last in first out)

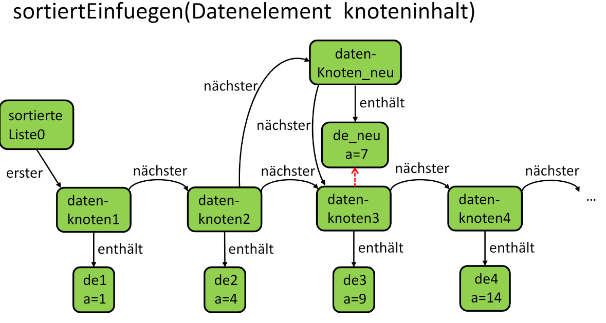
-> benötigte Einfügemethoden: vorneEinfügen, vorneEntnehmen **oder** hintenEinfügen, hintenEntnehmen

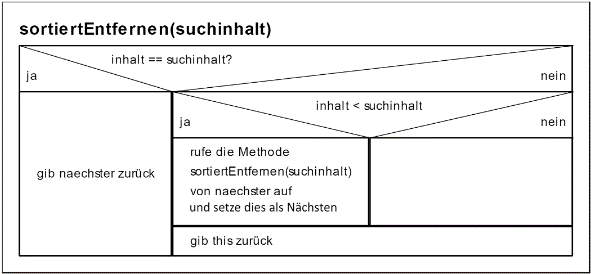
-> vorneEntnehmen & vorneEinfügen bei rekursiver Liste aber einfacher

## HJ1\_2 Die sortierte Liste

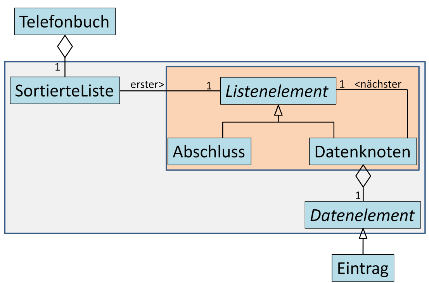
Die Datenwerte einer Liste können sortiert sein (zb alphabetisch, numerisch…)

Vergleich von Werten mit zb <>, mit eigen implementierten Methoden, oder bei Strings String.compareTo(string) -> (alphabetische Ordnung)

Beim Einfügen wird der Inhalt immer weitergegeben  
bis der Inhalt kleiner ist als der in der Liste.  
Hier wird dann eingefügt und der neue Datenknoten zurückgegeben.  
Wenn der neue Inhalt größer als alle anderen ist, fügt der Abschluss ein.

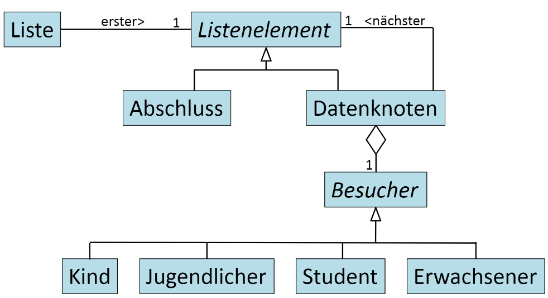


Beim Entfernen muss man nur so weit suchen, solange der Inhalt der Liste kleiner bleibt als der gesuchte Inhalt.   
Wenn der Inhalt größer ist als der Suchinhalt, kann der Suchinhalt aufgrund der Sortierung nicht mehr kommen



## HJ1\_3 Heterogene Listen

Liste mit mehreren verschiedenen Klassen als Datenelement -> dafür braucht man eine abstrakte Oberklasse aller Datenelemente, um alle Datenelement Klassen ansprechen zu können



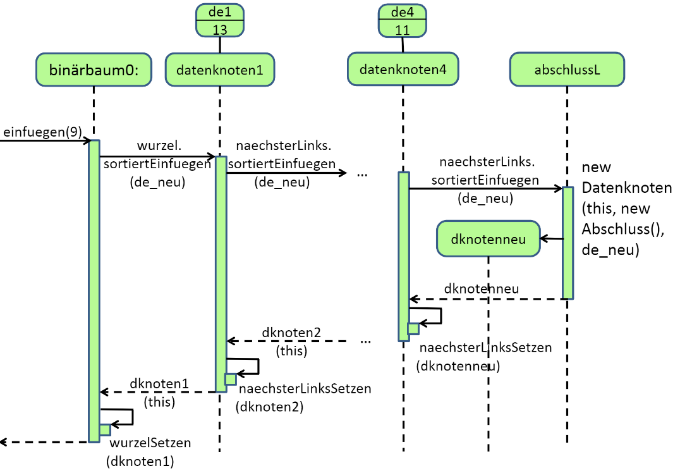
## HJ1\_4 Der geordnete Binärbaum

= binäre Suchbäume = BST (binary search tree)

Ein Binärbaum heißt geordnet, wenn der Schlüssel des LK kleiner und der Schlüssel des RK größer ist als die Wurzel

Einfügemethoden etc. sehr ähnlich zu Liste. Man muss nur den richtigen Teil-Ast unterscheiden:

public void sortiertEinfuegen(int i):



Datenknoten entfernen: (Pseudocode da es sehr kompliziert ist)

Suche im Baum nach dem Datenknoten  
 Wenn kein DK gefunden wurde: return  
 Wenn DK gefunden: ermittle Anzahl der Kindknoten  
 Anzahl 0: DK durch einen Abschluss ersetzen  
 Anzahl 1: DK durch den Kindknoten ersetzen  
 Anzahl 2: DK-Inhalt durch den kleinsten Inhalt im rechten Teil-Ast ersetzen; DK mit  
 dem kleinsten Inhalt im rechten Teil-Ast entfernen

# HJ 2

Gestrichen rip

# HJ 3

## HJ3\_1 Formale Sprachen (Referatsthema)

1. **Theorie**

Natürliche Sprache = zb Deutsch, Englisch…

Formale Sprache = zb Mathe, Chemieformeln, Programmiersprachen…

Alphabet ∑ := beliebige, endliche, nichtleere Menge, deren Elemente Zeichen oder Symbole genannt werden

Zeichenkette, Wort := endliche Folge von Zeichen des Alphabets

Leeres Wort Ɛ [„epsilon“] := Zeichenkette mit null Zeichen

Wortmenge ∑\* := Menge aller Wörter, die sich aus ∑ bilden lassen

∑ = {a,b}

∑\* = {Ɛ, a, b, aa, ab, ba, bb, aaa, …}

Teilmenge L von ∑\* := formale Sprache über dem Alphabet ∑

Wörter:

‚2+3=5‘ syntaktisch korrekt  
‚=+1-‘ syntaktisch falsch  
‚2+3=6‘ syntaktisch korrekt, aber semantisch (von der Bedeutung her) nicht sinnvoll

1. **Erzeugung**

S.13 Wetterkarte: ∑ = { ☀; ❄; 🌨; 1; 2; 3; 4; 5; 6; 7; 8; 9; 0; -; °C, %, /}

Produktionsregeln:

<S> -> <W> <T> ‘/‘ <L>  
<W> -> <WS> | <WS> <WS> | <WS> <WS> <WS>

<WS> -> ‚☀‘ | ‚❄‘ | ‚🌨‘  
<T> -> ‚-‘ <Z> ‚°C‘ | <Z> ‚°C‘ | ‚0’ ‚°C‘

<Z> -> <ziffernichtnull> | <ziffernichtnull> <ziffer>

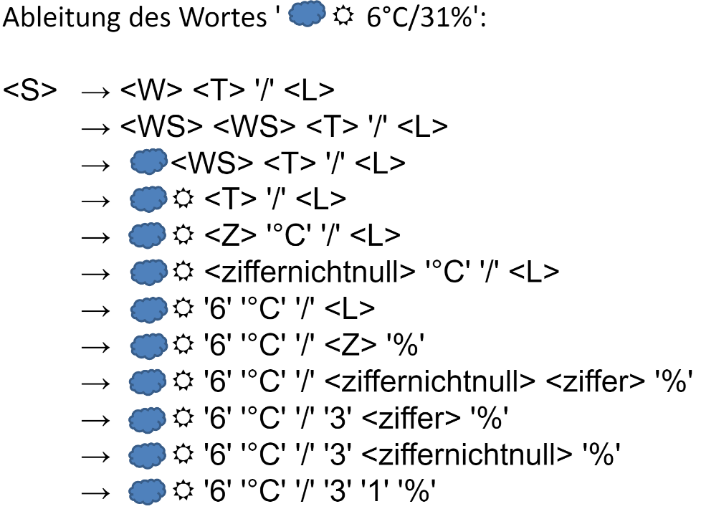
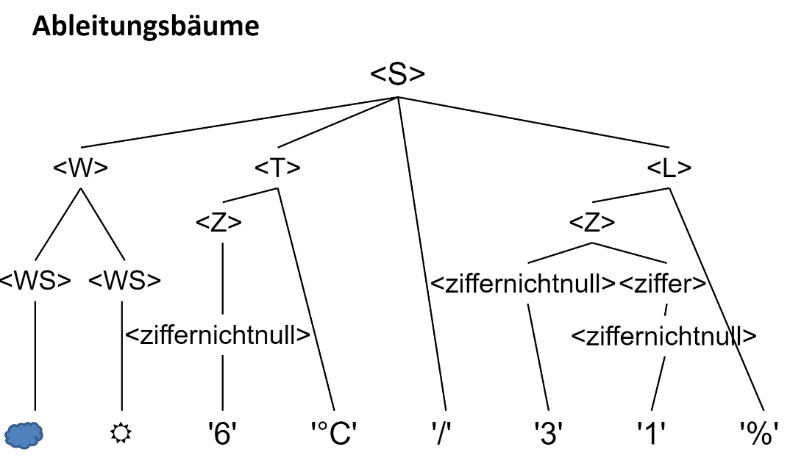
<ziffernichtnull> -> ‚1‘ | ‚2‘ | … | ‚9‘

<ziffer> -> <ziffernichtnull> | ‚0‘

<L> -> <Z> ‚%‘ | ‚0‘ ‚%‘

**Syntaktische Variablen** stehen in spitzen Klammern (zb <W>). Sie werden in der Ableitung eines Wortes ersetzt und heißen **Nichtterminale**. Sie gehören nicht zu den Symbolen der Sprache.

Alle Symbole, die zu ∑ gehören, werden nicht ersetzt (**Terminale**). Diese werden in Hochkommata gefasst (zb ‚1‘).

1. **Ableitungen**

Startvariable (zb <S>): syntaktische Variable, mit der alle Ableitungen einer Sprache beginnen

Wenn ein Wort, nur durch anwenden der Produktionsregeln (ausgehend von der Startvariable) erzeugt werden kann, gehört es zur Sprache, sonst nicht.

<A> -> ‚0‘ | ‚0‘ <A> : rekursive Produktion, theoretisch unendlich lange wörter

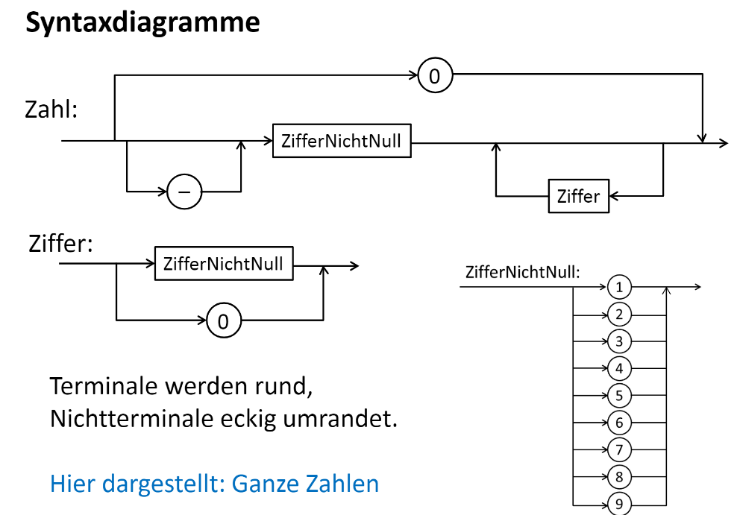
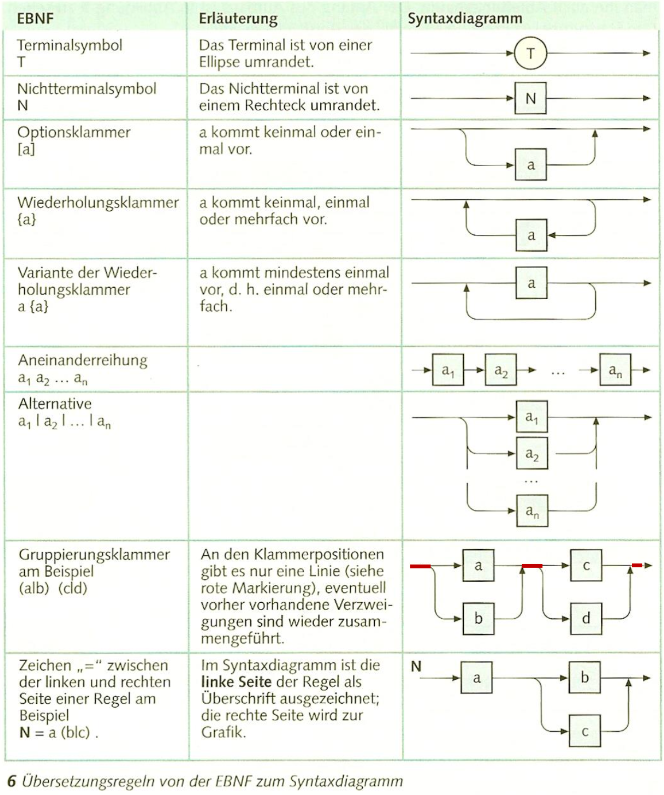
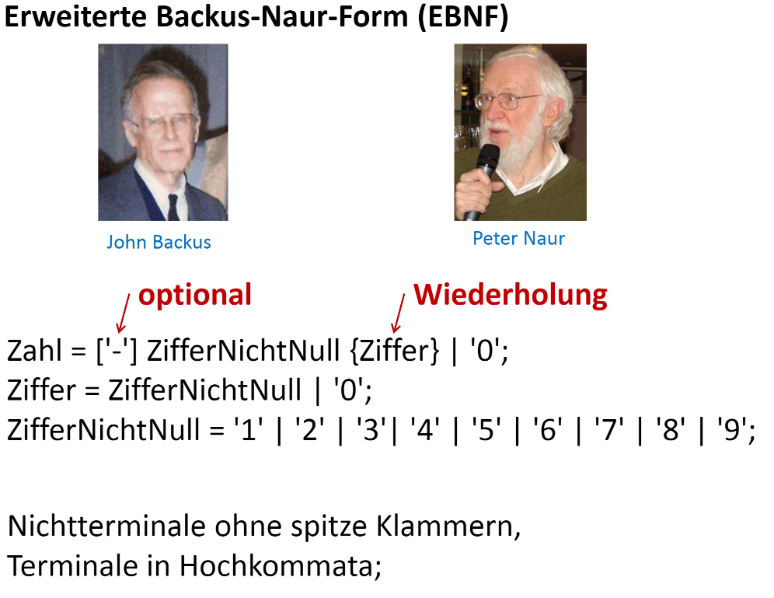
<A> -> b<B> | b verschränkte Rekursion

<B> -> a<A> | a

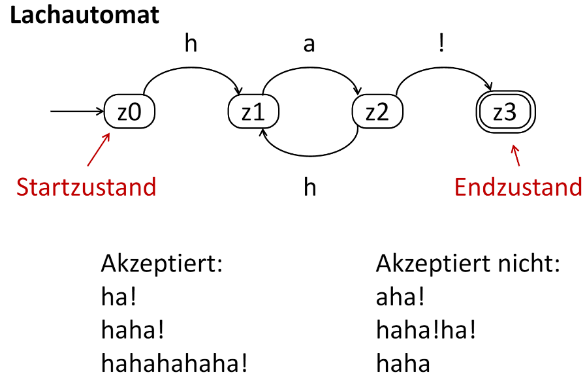
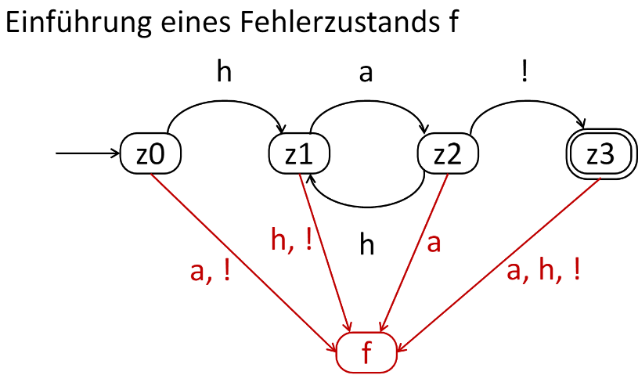
1. **Formale Grammatik**

Grammatik G = (V, ∑, P, S) <- Muss immer mit angegeben werden [sprich „VEPS“]

V := Menger der syntaktischen Variablen  
 ∑ := Alphabet  
 P: Menge der Produktionsregeln  
 S: Startvariable  
Dabei gilt: V ∩ ∑ = {} (V darf keine Nichtterminale enthalten, die Zeichen in ∑ sind)

1. **Notationsformen**

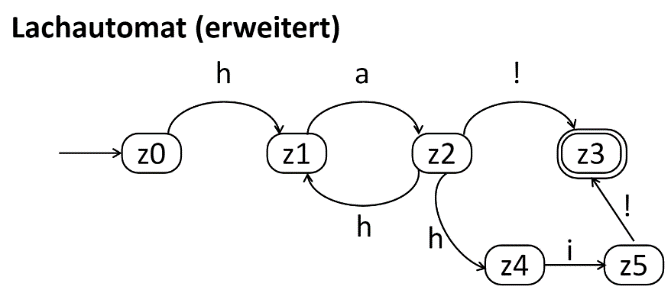
**Bei EBNF: Semikolon am Ende nicht vergessen**

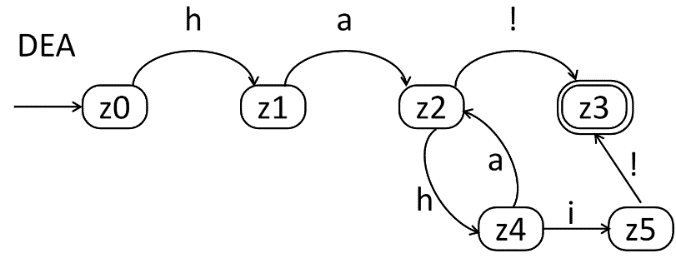
1. **Erkennende Automaten**

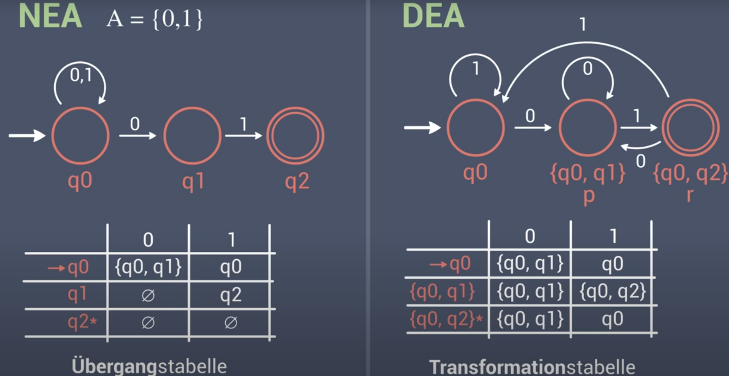
Ein Wort wird akzeptiert, wenn der Automat (beginnend im Startzustand) nach Abarbeitung des Wortes, in einem Endzustand ist.

Erkennender Automat, Akzeptor := Automat, der ein Wort auf   
 Korrektheit überprüft

Endlicher Automat := Automat mit endlich vielen Zuständen  
 und Verarbeitung einer endlich langen  
 Zeichenkette

Wenn es mindestens einen Zustand gibt (hier z2), von dem aus ein Eingabezeichen eine Transition zu verschiedenen Folgezuständen auslösen kann, ist der Automat **nichtdeterministisch (NEA)**

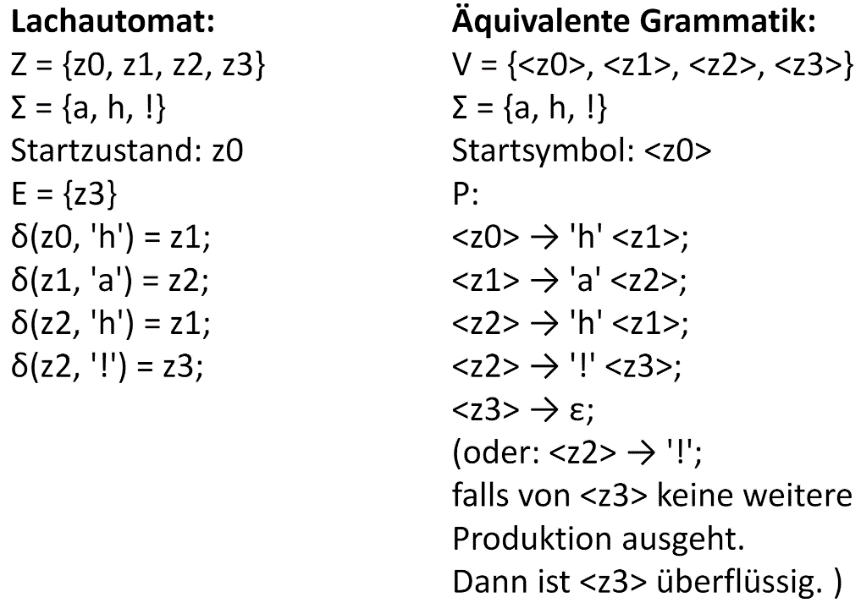
Wir implementieren aber nur DEA (deterministische, endliche Automaten)

Zu jedem NEA gibt es einen DEA, der die gleiche Sprache akzeptiert.

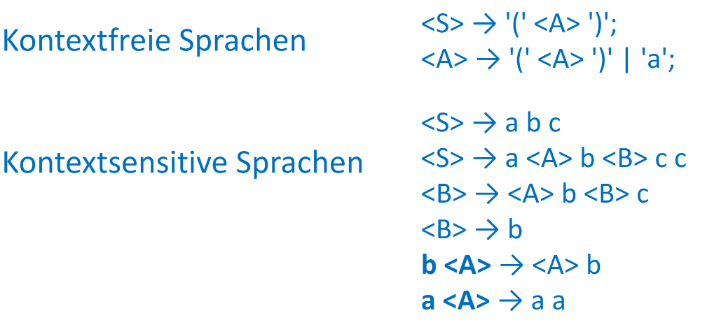
Definition DEA (**D**eterministischer **E**ndlicher **A**utomat)

1. Endliche Menge Z von Zuständen
2. Endliches Eingabealphabet ∑
3. Ein Startzustand
4. Menge E von Endzuständen
5. Zweistellige Übergangsfkt δ, die jedem Paar aus Zuständen und Eingabezeichen einen Folgezustand zuordnet

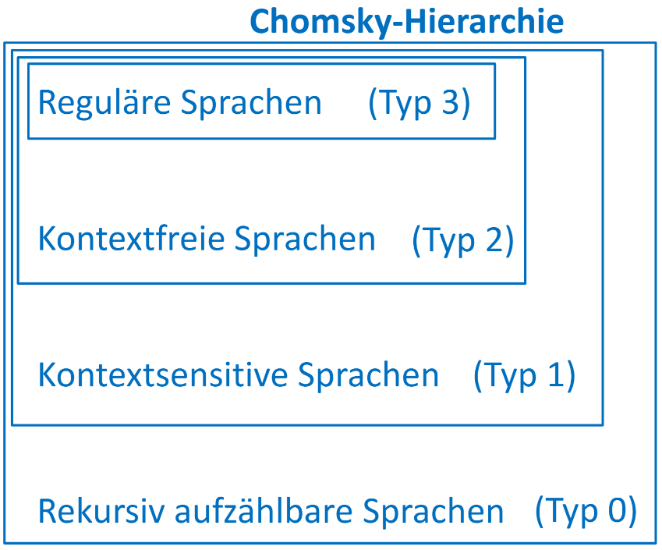
https://youtu.be/ArYI4dd7ffE



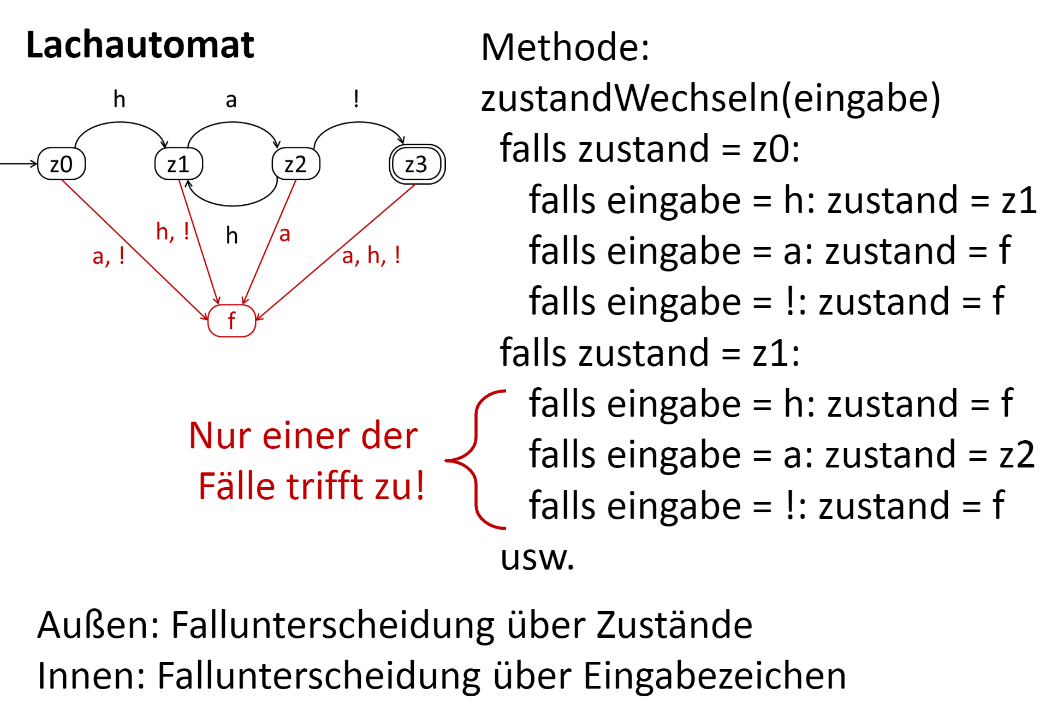
* Rechtslineare Grammatik

Reguläre Sprache := Jede Sprache, die durch einen endlichen Automaten erkannt werden kann

Zb Rekursive Produktionen können nicht erkannt werden, da sie potenziell unendlich lange Wörter bilden können



1. **Implementierung von endlichen Automaten**



* switch-Statement für Außen; if-Statements für Innen

# HJ 4

## HJ4\_1 Aufbau und Funktionsweise eines Rechners, Zustandsübergänge der Registermaschine

## HJ4\_2 Programmierung auf Maschinenebene

## HJ4\_3 Grenzen der Berechenbarkeit