

---

# IHK Abschlussprüfung Sommer 2018

---

Entwicklung eines Softwaresystems

Thema: Netzplanerstellung

Martin Leonard Haufs

1. Vorstellung des Ausbildungsbetriebes
2. Eigene Projekte
3. Aufgabenstellung
4. Klassenstruktur
5. Algorithmen
6. Implementation

# 1) Ausbildungsbetrieb

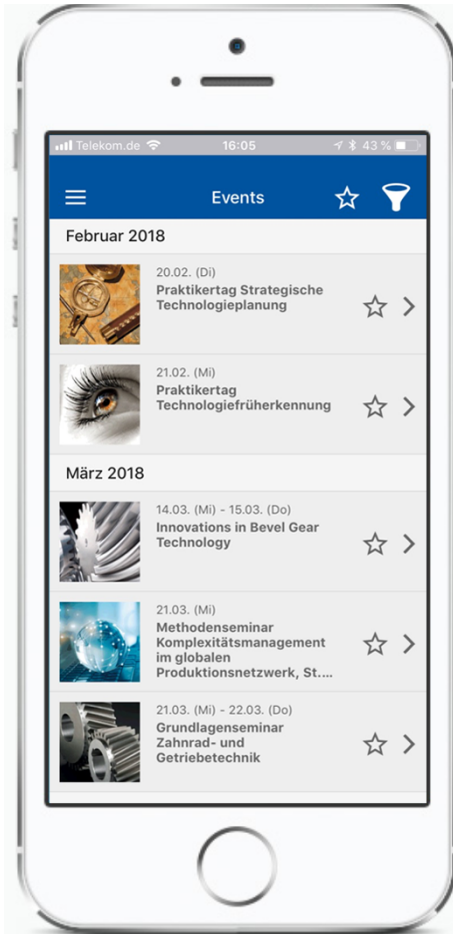
---

## ■ Werkzeugmaschinenlabor (WZL)

- Institut der RWTH Aachen
- Abteilung für Werkzeugmaschinen
  - Prof. Dr. Brecher
  - Konstruktion und Berechnung von Produktionsanlagen
  - Maschinenelemente und –komponenten
  - Steuerungstechnik und Automatisierung
  - Informationstechnik und –management
  - Mensch-Maschine-Interaktion
  - Automatisierung



## 2) Meine Projekte



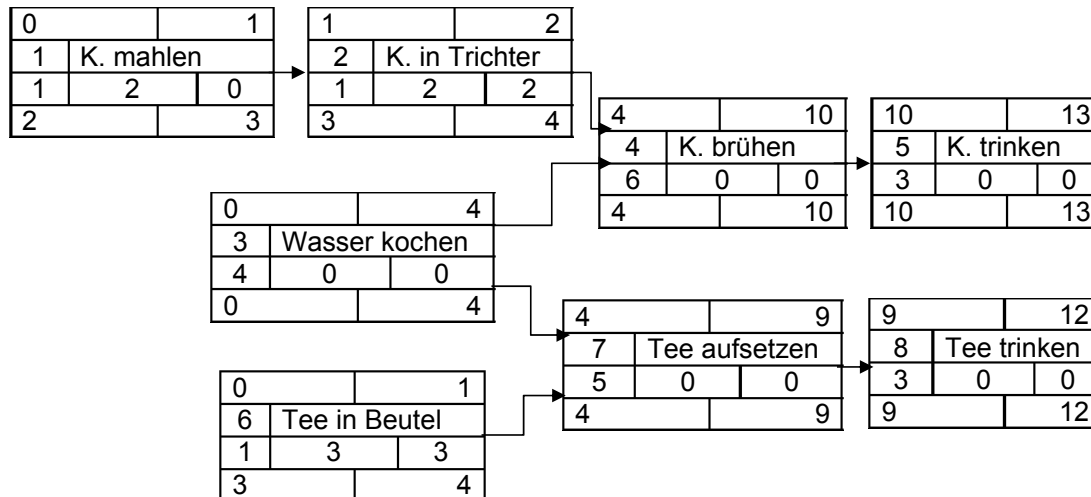
### Hybride mobile Anwendungsentwicklung

- Tagungsapp für das Aachener Werkzeugmaschinen-Kolloquium (AWK)
  - Appcelerator Titanium SDK
  - Entwicklung nativer iOS-Module für Titanium in Objective C
- Messeapp für den DGSV Kongress
  - Weltkongress der Sterilgutversorgung
  - Technologie: Ionic 1 mit AngularJS
- Eventapp für das WZL-Forum
  - Ionic 1 mit Typescript
- Neukonzeption als Multieventapp
  - Umstellung von Ionic1 auf Ionic 2



### 3) Aufgabenstellung

FAZ		FEZ	
Nr			
D		GP	FP
SAZ		SEZ	



- Erstellung eines Netzplans
- Netzplan – Netz aus Knoten
  - Dauer (D)
  - Frühester Anfangszeitpunkt (FAZ)
  - Frühester Endzeitpunkt (FEZ)
  - Spätester Anfangszeitpunkt (SAZ)
  - Spätester Endzeitpunkt (SEZ)
  - Gesamtpuffer (GP)
  - Freier Puffer (FP)
- Ermittlung der Dauer des Gesamtprojektes
- Ermittlung der Kritischen Pfade

## 4) Klassenstruktur - Datenmodell

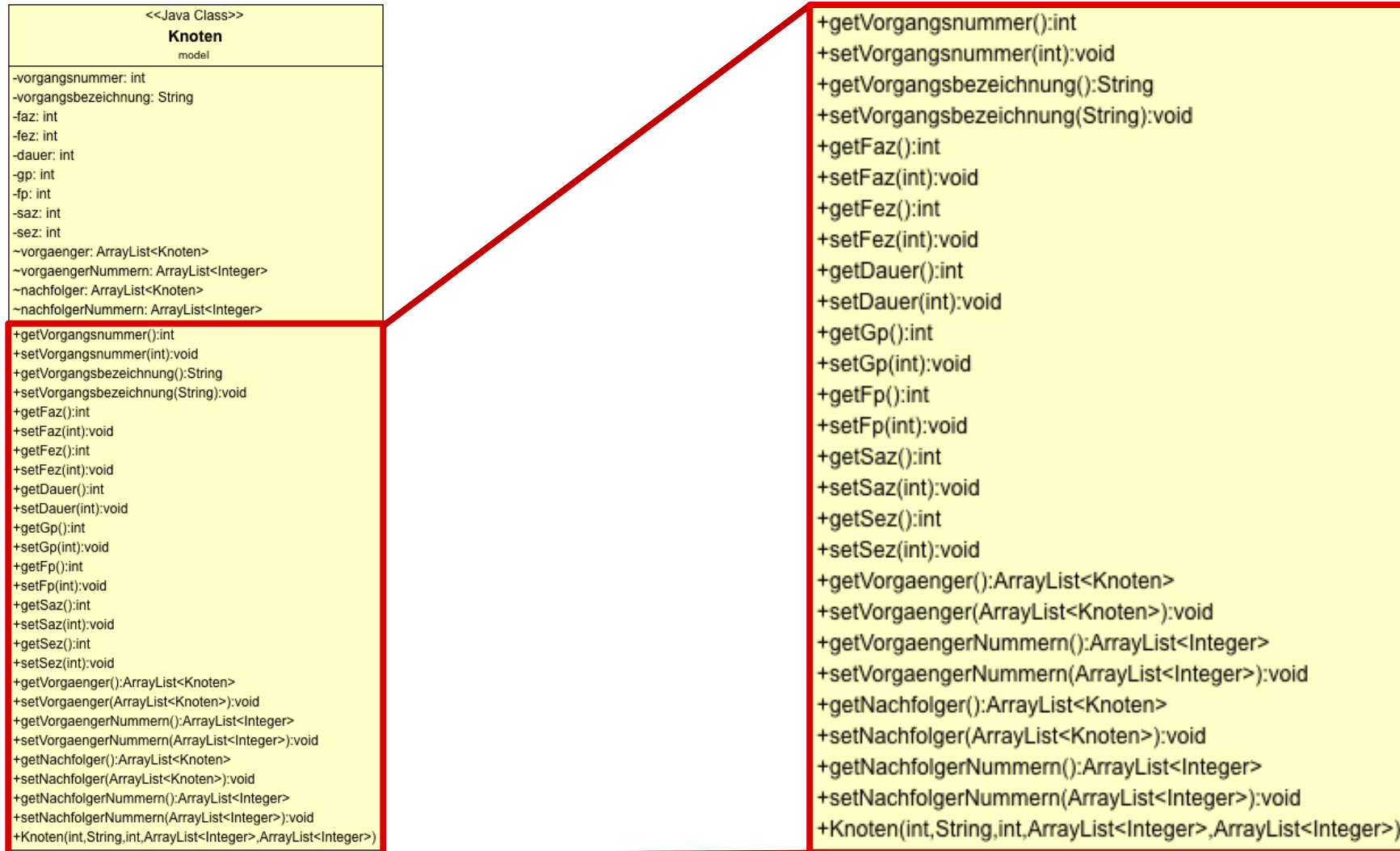
```
<<Java Class>>
Knoten
model

-vorgangsnummer: int
-vorgangsbezeichnung: String
-faz: int
-fez: int
-dauer: int
-gp: int
-fp: int
-saz: int
-sez: int
~vorgaenger: ArrayList<Knoten>
~vorgaengerNummern: ArrayList<Integer>
~nachfolger: ArrayList<Knoten>
~nachfolgerNummern: ArrayList<Integer>

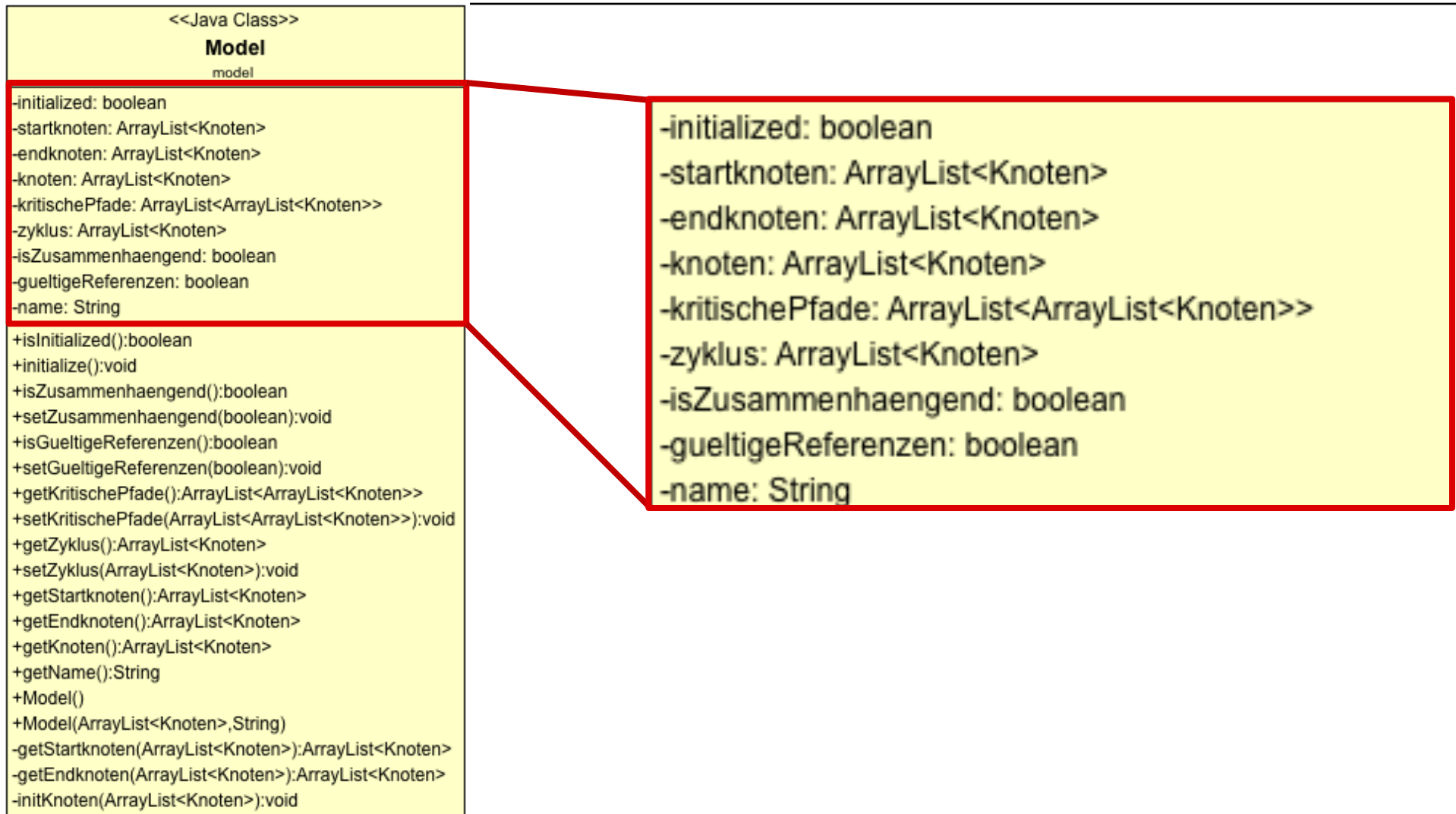
+getVorgangsnummer():int
+setVorgangsnummer(int):void
+getVorgangsbezeichnung():String
+setVorgangsbezeichnung(String):void
+getFaz():int
+setFaz(int):void
+getFez():int
+setFez(int):void
+getDauer():int
+setDauer(int):void
+getGp():int
+setGp(int):void
+getFp():int
+setFp(int):void
+getSaz():int
+setSaz(int):void
+getSez():int
+setSez(int):void
+getVorgaenger():ArrayList<Knoten>
+setVorgaenger(ArrayList<Knoten>):void
+getVorgaengerNummern():ArrayList<Integer>
+setVorgaengerNummern(ArrayList<Integer>):void
+getNachfolger():ArrayList<Knoten>
+setNachfolger(ArrayList<Knoten>):void
+getNachfolgerNummern():ArrayList<Integer>
+setNachfolgerNummern(ArrayList<Integer>):void
+Knoten(int,String,int,ArrayList<Integer>,ArrayList<Integer>)
```

```
-vorgangsnummer: int
-vorgangsbezeichnung: String
-faz: int
-fez: int
-dauer: int
-gp: int
-fp: int
-saz: int
-sez: int
~vorgaenger: ArrayList<Knoten>
~vorgaengerNummern: ArrayList<Integer>
~nachfolger: ArrayList<Knoten>
~nachfolgerNummern: ArrayList<Integer>
```

# Klassenstruktur - Datenmodell



# Klassenstruktur - Datenmodell

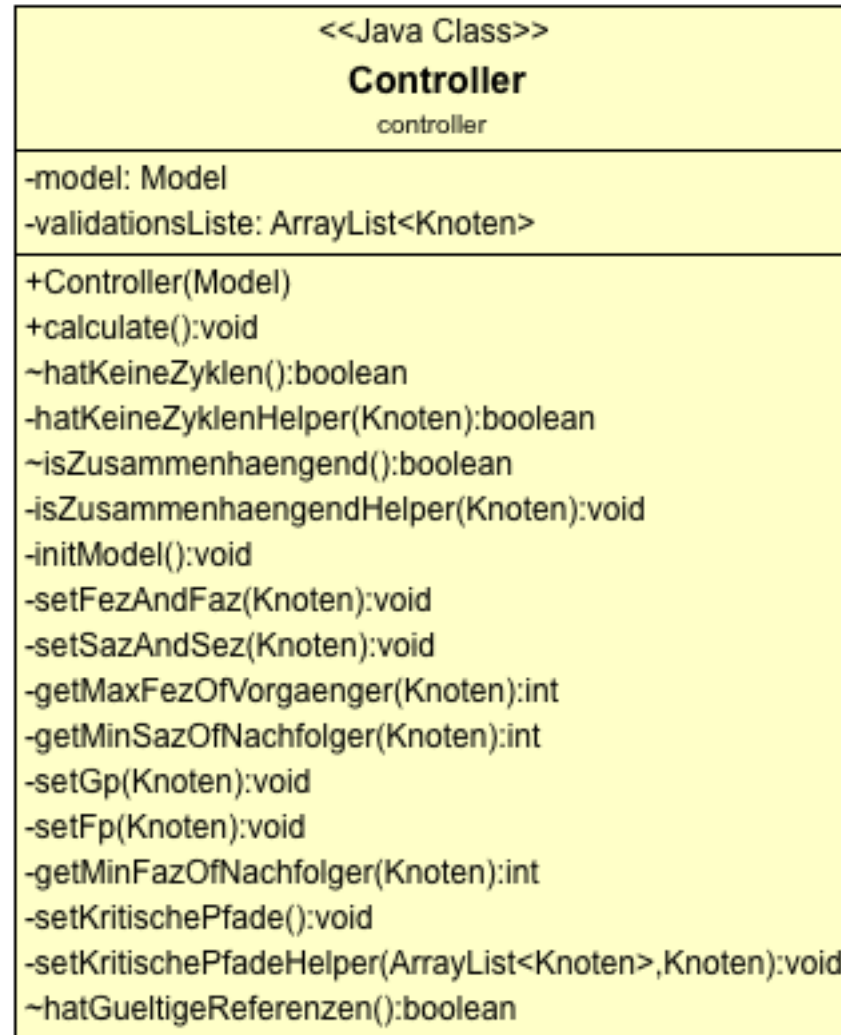




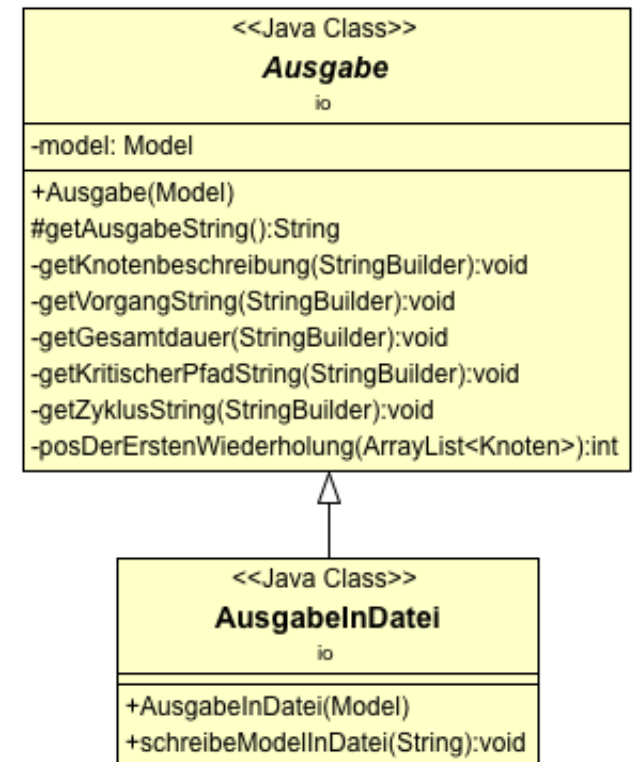
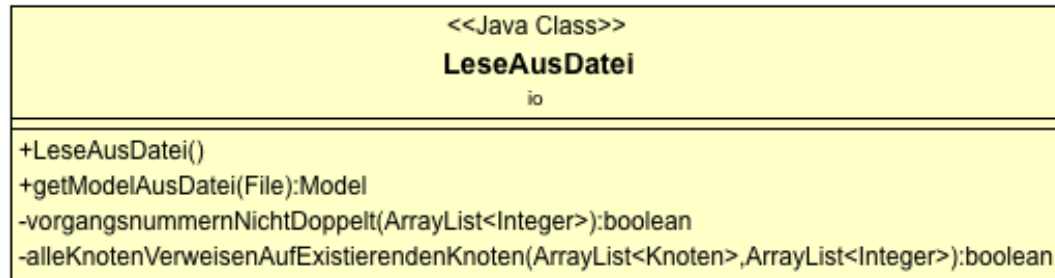
# Klassenstruktur - Datenmodell



# Klassenstruktur - Controller



# Klassenstruktur – Eingabe & Ausgabe



# Algorithmus

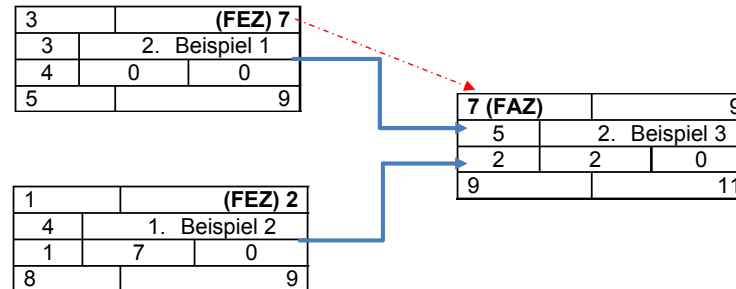
## a) Vorwärtsrechnung

- Startknoten hat FAZ = 0

Durchlaufen des Graphen von den Startpunkten → Endpunkten

- $FEZ = FAZ + \text{Dauer}$

- Größter FEZ eines Vorgängers ist FAZ des Nachfolgers



- Dauer (D)
- Frühester Anfangszeitpunkt (FAZ)
- Frühester Endzeitpunkt (FEZ)
- Spätester Anfangszeitpunkt (SAZ)
- Spätester Endzeitpunkt (SEZ)
- Gesamtpuffer (GP)
- Freier Puffer (FP)

# Algorithmus

## b) Rückwärtsrechnung

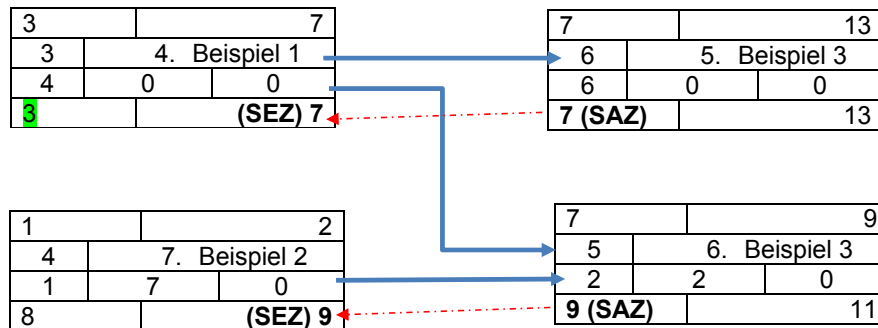
Durchlaufen des Graphen von den Endpunkten → Startpunkten

FAZ		FEZ	
Nr	Name		
Dauer	GP	FP	
SAZ		SEZ	

■ Endknoten haben SEZ = FEZ

■  $SAZ = SEZ - \text{Dauer}$

■ SEZ eines Knotens ist der minimale SAZ der Nachfolgeknoten



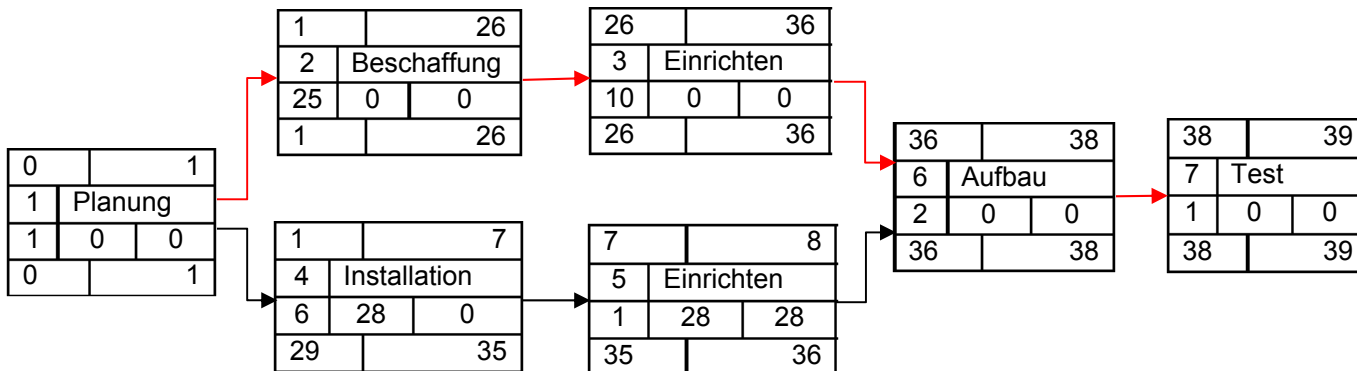
- Dauer (D)
- Frühester Anfangszeitpunkt (FAZ)
- Frühester Endzeitpunkt (FEZ)
- Spätester Anfangszeitpunkt (SAZ)
- Spätester Endzeitpunkt (SEZ)
- Gesamtpuffer (GP)
- Freier Puffer (FP)

# Algorithmus

## c) Ermittlung der Zeitreserven und des kritischen Pfades

Durchlaufen des Graphen von den  
Endpunkten → Startpunkten

- $GP = SAZ - FAZ \Leftrightarrow GP = SEZ - FEZ$
- $FP = (\text{minimales FAZ der Nachfolger}) - FEZ$
- Kritischer Pfad:
  - Menge der Knoten mit  $GP = 0$  und  $FP = 0$



- Dauer (D)
- Frühester Anfangszeitpunkt (FAZ)
- Frühester Endzeitpunkt (FEZ)
- Spätester Anfangszeitpunkt (SAZ)
- Spätester Endzeitpunkt (SEZ)
- Gesamtpuffer (GP)
- Freier Puffer (FP)

# Algorithmus

- Initialisieren des Graphen
  - In allen Knoten Vorgänger und Nachfolger referenzieren

**initKnoten(List<Knoten> knoten)**

Für jeden Knoten k in knoten

Für jeden int vorgaengerNr in k.vorgaengerNummern

Für jeden Knoten k2 in knoten

$k2.vorgaengerNummer == vorgaengerNr$

T

F

Füge k2 zu k.vorgaenger hinzu

∅

Für jeden int nachfolgerNr in k.nachfolgerNummern

Für jeden Knoten k2 in knoten

$k2.nachfolgerNummer == nachfolgerNr$

T

F

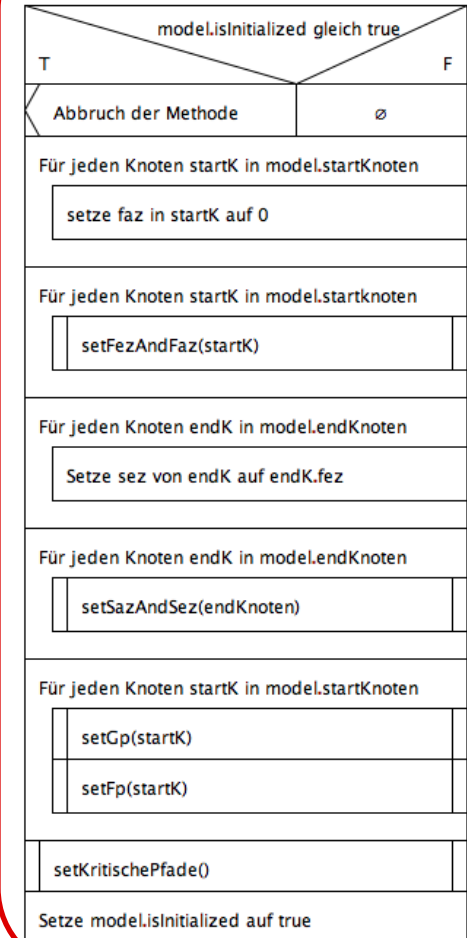
Füge k2 zu k.nachfolger hinzu

∅

# Algorithmus

- Vorwärtsrechnung
  - Frühester Endzeitpunkt (FEZ) und frühesten Anfangszeitpunkt (FAZ) berechnen
    - Rekursiver Baumdurchlauf ausgehend von Startpunkten
- Rückwärtsrechnung
  - Späteste Endzeitpunkte (SEZ) und Späteste Anfangszeitpunkte (SAZ) berechnen
    - Rekursiver Baumdurchlauf ausgehend von Endpunkten
- Ermittlung der Zeitreserven
  - Rekursiver Baumdurchlauf
- Berechnung der Kritischen Pfade
  - Backtracking

initModel(): void





# Algorithmus

## ■ Vorwärtsrechnung

- Frühester Endzeitpunkt (FEZ) und frühesten Anfangszeitpunkt (FAZ) berechnen
  - Rekursiver Baumdurchlauf ausgehend
  - von Startpunkten

- Startknoten hat FAZ = 0
- FEZ = FAZ + Dauer
- Größter FEZ eines Vorgängers ist FAZ des Nachfolgers

### setFezAndFaz(Knoten aktKnoten): void

Setze aktKnoten.fez auf aktKnoten.faz + aktKnoten.dauer

aktKnoten ist Endknoten  
(Länge von aktKnoten.nachfolger gleich 0)

T

F

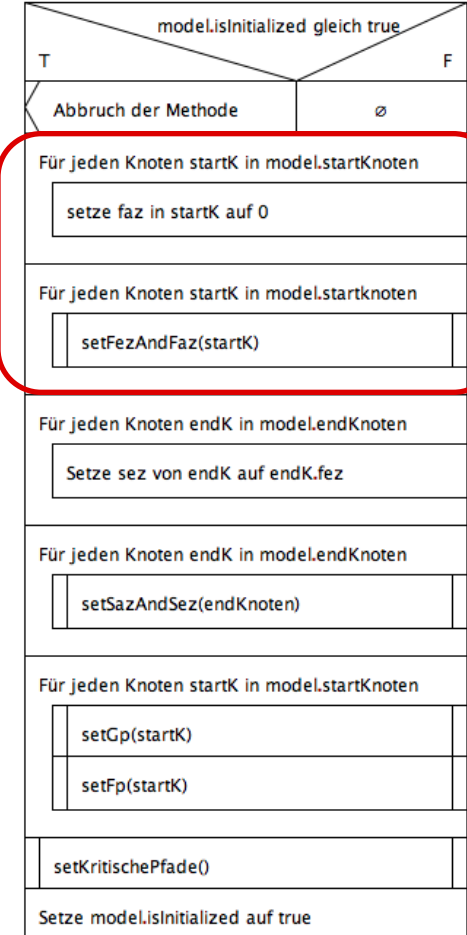
Setze faz von aktKnoten auf maximale fez der Vorgängerknoten von aktKnoten

Für jeden knoten nachfolger von aktKnoten.nachfolger

setze nachfolger.faz auf den maximalen fez der Vorgänger von nachfolger

setFezAndFaz(nachfolger)

### initModel(): void



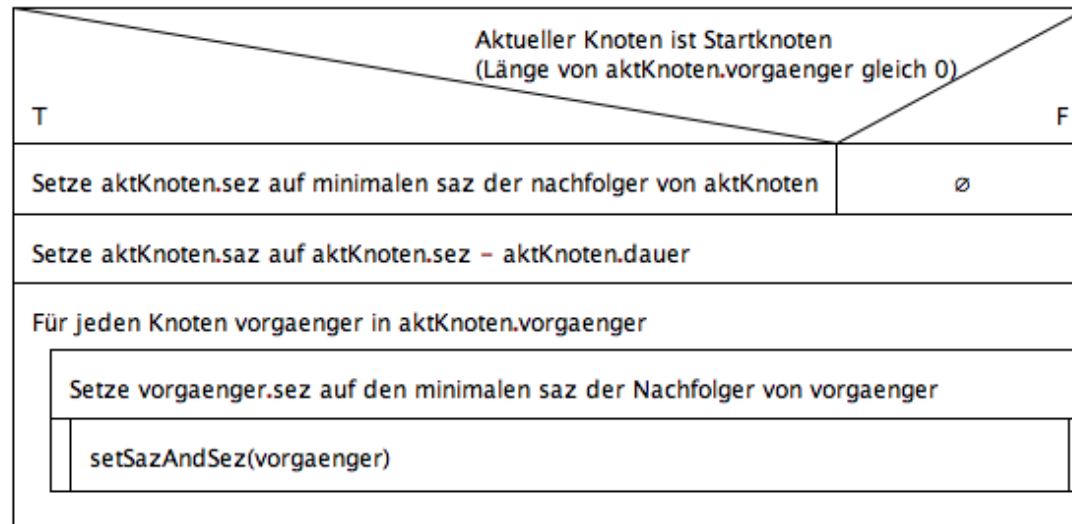
# Algorithmus

## ■ Rückwärtsrechnung

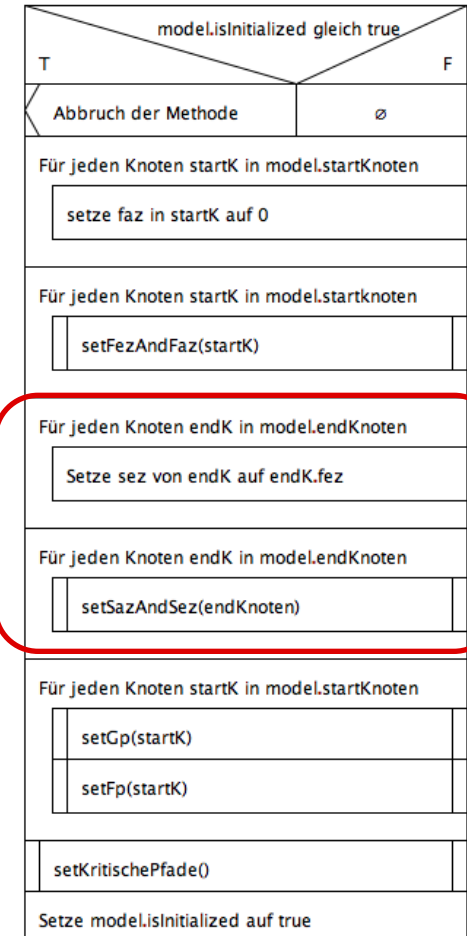
- Späteste Endzeitpunkte (SEZ) und Späteste Anfangszeitpunkte (SAZ) berechnen
  - Rekursiver Baumdurchlauf ausgehend von Endpunkten

- Endknoten haben  $SEZ = FEZ$
- $SAZ = SEZ - \text{Dauer}$
- SEZ eines Knotens ist das minimale SAZ der Nachfolgeknoten

**setSazAndSez(Knoten aktKnoten): void**



**initModel(): void**



# Algorithmus

- Ermittlung der Zeitreserven
  - Rekursiver Baumdurchlauf

**setGp(Knoten aktKnoten): void**

Setze aktKnoten.gp auf aktKnote.saz – aktKnoten.faz

Für jeden Knoten nachfolger von aktKnoten.nachfolger

setGp(nachfolger)

**setFp(Knoten aktKnoten): void**

Setze aktKnoten.fp auf (minimales faz der Nachfolger von aktKnoten) – aktKnoten.fez

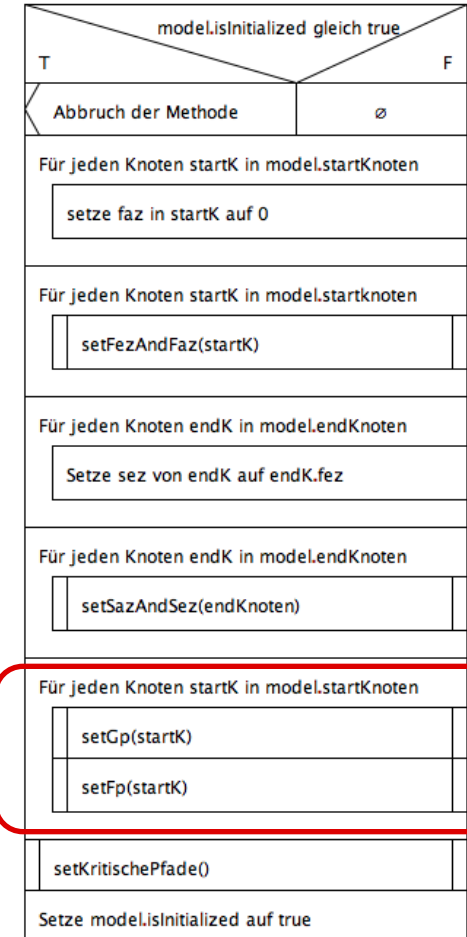
Für jeden Knoten nachfolger in aktKnoten.nachfolger

setFp(nachfolger)

$GP = SAZ - FAZ$

$FP = (\text{minimales FAZ der Nachfolger}) - FEZ$

**initModel(): void**



# Algorithmus

- Berechnung der Kritischen Pfade
  - Backtracking
  - Abbruchkriterium: Endknoten erreicht

Kritischer Pfad:  
Menge der Knoten mit  
GP = 0 und FP = 0

## setKritischePfade(): void

Initialisiere model.kritischePfade als leere Liste von Knoten

Für jeden Knoten startK in model.startKnoten

Initialisiere leere Liste von Knoten pfad

setKritischePfadeHelper(pfad, startK)

## setKritischePfadeHelper(List<Knoten> pfad, Knoten aktKnoten): void

aktKnoten ist Endknoten  
(aktKnoten.nachfolger hat Länge 0)

T

F

Füge aktKnoten zu pfad hinzu

Erstelle Kopie von pfad als pfadKopie

Füge pfadKopie zu model.kritischePfade hinzu

Return

∅

aktKnoten.gp == 0 und  
aktKnoten.fp == 0

T

F

Erstelle Kopie von pfad als pfadKopie

Füge aktKnoten zu pfadKopie hinzu

Für jeden Knoten nachfolger in aktKnoten.nachfolger

setKritischePfadeHelper(pfadKopie, nachfolger)

∅

## initModel(): void

model.isInitialized gleich true

T

F

Abbruch der Methode

∅

Für jeden Knoten startK in model.startKnoten

setze faz in startK auf 0

Für jeden Knoten startK in model.startknoten

setFezAndFaz(startK)

Für jeden Knoten endK in model.endKnoten

Setze sez von endK auf endK.fez

Für jeden Knoten endK in model.endKnoten

setSazAndSez(endKnoten)

Für jeden Knoten startK in model.startKnoten

setGp(startK)

setFp(startK)

setKritischePfade()

Setze model.isInitialized auf true

# Implementation

---

- Sprache Java
- Plattformübergreifend
- Shellscripte zum automatisierten öffnen der Tests