

# Softwaretechnik 1 - 6. Tutorium

Tutorium 03

Felix Bachmann | 24.07.2017

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



- 1 Orga
  - Allgemeines
  - Feedback
- 2 Testen
  - KFO
- 3 Wiederholung und Klausuraufgaben
  - Planung & Definition
  - Entwurf
  - Implementierung
  - Testen
  - Abnahme, Einsatz & Wartung
- 4 Ende



# 6. Übungsblatt Statistik

## Allgemein



## Aufgabe 1 (Kontrollfluss-orientiertes Testen): Ø von 5+1

## Aufgabe 1 (Kontrollfluss-orientiertes Testen): Ø von 5+1



## Aufgabe 2 (Codeinspektion): Ø von 4



## Aufgabe 2 (Codeinspektion): $\emptyset$ von 4



## Aufgabe 2 (Codeinspektion): Ø von 4



## Aufgabe 3 (Parallelisierung von Geometrfy): Ø von 5

## Aufgabe 2 (Codeinspektion): Ø von 4



## Aufgabe 3 (Parallelisierung von Geometrfy): Ø von 5



## Aufgabe 4 (Alternative Parallelisierungsvarianten): $\emptyset$ von 6+3

## Aufgabe 4 (Alternative Parallelisierungsvarianten): $\emptyset$ von 6+3



Aufgabe 4 (Alternative Parallelisierungsvarianten):  $\emptyset$  von 6+3



Aufgabe 5 (Parallelisierungswettbewerb):  $\emptyset$  von 6

## Aufgabe 4 (Alternative Parallelisierungsvarianten): $\emptyset$ von 6+3



## Aufgabe 5 (Parallelisierungswettbewerb): $\emptyset$ von 6



# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- ① gegeben: zu testender Code



# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- 1 gegeben: zu testender Code
- 2 Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen

# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- 1 gegeben: zu testender Code
- 2 Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen
- 3 Zwischensprache  $\implies$  Kontrollflussgraph

# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- ① gegeben: zu testender Code
- ② Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen
- ③ Zwischensprache  $\implies$  Kontrollflussgraph
- ④ am Kontrollflussgraphen Testfälle finden:

# Kontrollflussorientiertes Testverfahren (KFO)

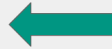
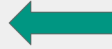
- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- 1 gegeben: zu testender Code
- 2 Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen
- 3 Zwischensprache  $\implies$  Kontrollflussgraph
- 4 am Kontrollflussgraphen Testfälle finden:
  - Anweisungsüberdeckung
  - Zweigüberdeckung
  - Pfadüberdeckung

## ■ Sprünge umwandeln

```
1  int a = 9;
2  System.out.println("Blahblah");
3  while(a == 9) {
4      int z = 0;
5      for(int i = 0; i <= 8; i++) {
6          z++;
7      }
8      int k = 0;
9      if(a == z + k) {
10         a = 8;
11     }
12 }
```



## ■ Sprünge umwandeln

```
1  int a = 9;
2  System.out.println("Blahblah");
3  if not (a == 9) goto 14;
4      int z = 0;
5      int i = 0;
6      if not (i <= 8) goto 10;
7          z++;
8          i++;
9      goto 6;
10     int k = 0;
11     if not (a == z + k) goto 13;
12         a = 8;
13 goto 3;
14
```

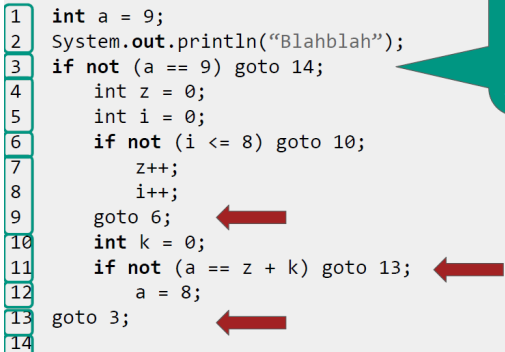
- Grundblöcke finden (Code bis goto ist ein Grundblock)

```
1  int a = 9;  
2  System.out.println("Blahblah");  
3  if not (a == 9) goto 14;  
4      int z = 0;  
5      int i = 0;  
6      if not (i <= 8) goto 10;  
7          z++;  
8          i++;  
9      goto 6;  
10     int k = 0;  
11     if not (a == z + k) goto 13;  
12         a = 8;  
13 goto 3;  
14
```

Grundblöcke dürfen  
nur am Ende einen  
Sprung (goto)  
haben (müssen  
aber nicht)

- Grundblöcke prüfen (goto dürfen nur an Anfang eines Grundblocks verweisen)

```
1  int a = 9;  
2  System.out.println("Blahblah");  
3  if not (a == 9) goto 14;  
4      int z = 0;  
5      int i = 0;  
6      if not (i <= 8) goto 10;  
7          z++;  
8          i++;  
9      goto 6;  
10     int k = 0;  
11     if not (a == z + k) goto 13;  
12         a = 8;  
13     goto 3;  
14
```

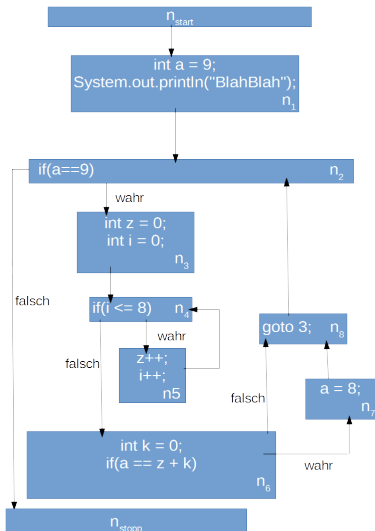




# KFO: Zwischensprache nach Kontrollflussgraph

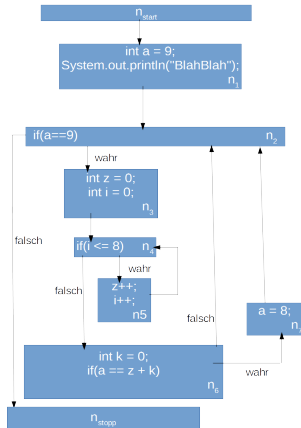
- Grundblöcke benennen
- Grundblöcke und Verzweigungen hinzeichnen
- Start- und Endzustand hinzufügen

# KFO: Zwischensprache nach Kontrollflussgraph



# KFO: Zwischensprache nach Kontrollflussgraph

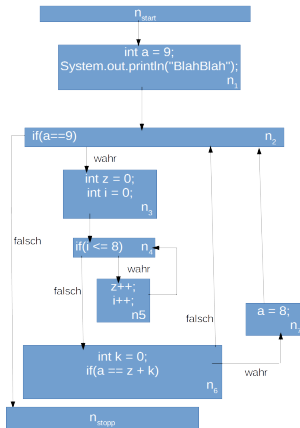
- goto-Knoten kann man auch weglassen



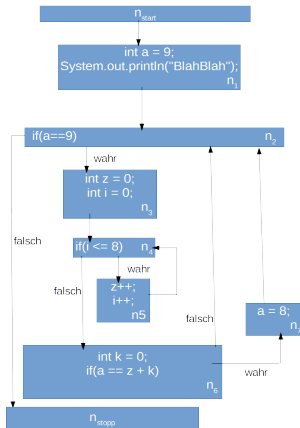
- Pfade finden, sodass jeder Grundblock traversiert wird

- Pfade finden, sodass jeder Grundblock traversiert wird  
⇒ Entdeckung nicht erreichbarer Code-Abschnitte

- Pfade finden, sodass jeder Grundblock traversiert wird  
⇒ Entdeckung nicht erreichbarer Code-Abschnitte
- aber: kein ausreichendes Testkriterium



## ■ Pfad für Anweisungsüberdeckung?



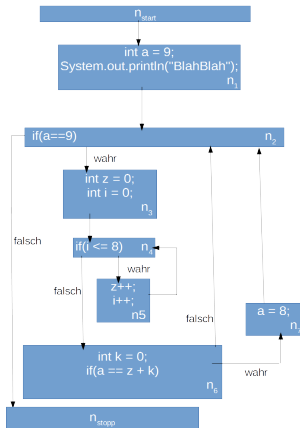
- Pfad für Anweisungsüberdeckung?  
 $(n_{start}, n_1, n_2, n_3, n_4, n_5, n_4, n_6, n_7, n_2, n_{stopp})$



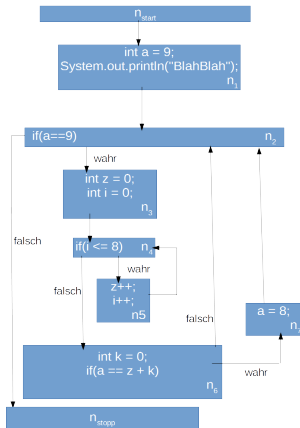
- Pfade finden, sodass jeder Zweig (=Kante) traversiert wird

- Pfade finden, sodass jeder Zweig (=Kante) traversiert wird  
⇒ Entdeckung nicht erreichbarer Kanten

- Pfade finden, sodass jeder Zweig (=Kante) traversiert wird  
⇒ Entdeckung nicht erreichbarer Kanten
- aber: Schleifen werden nicht ausreichend getestet



## ■ Pfad für Zweigüberdeckung?



## ■ Pfad für Zweigüberdeckung?

$(n_{start}, n_1, n_2, n_3, n_4, n_5, n_4, n_6, n_2, n_3, n_4, n_5, n_4, n_6, n_7, n_2, n_{stopp})$

- Finde **alle** vollständige, unterschiedlichen Pfade

- Finde **alle** vollständige, unterschiedlichen Pfade
- vollständiger Pfad = Anfang bei  $n_{start}$ , Ende bei  $n_{stopp}$

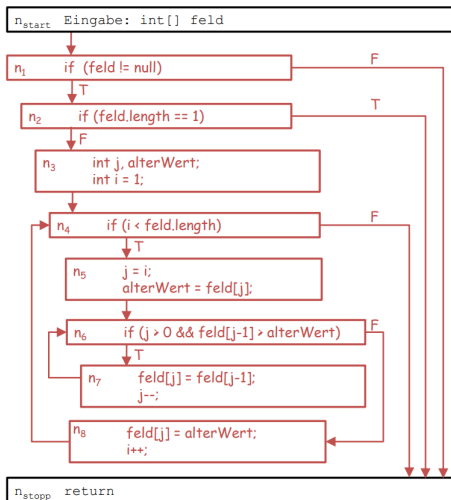
- Finde **alle** vollständige, unterschiedlichen Pfade
- vollständiger Pfad = Anfang bei  $n_{start}$ , Ende bei  $n_{stopp}$
- nicht praktikabel, da
  - Schleifen die Anzahl der möglichen Pfade stark erhöhen

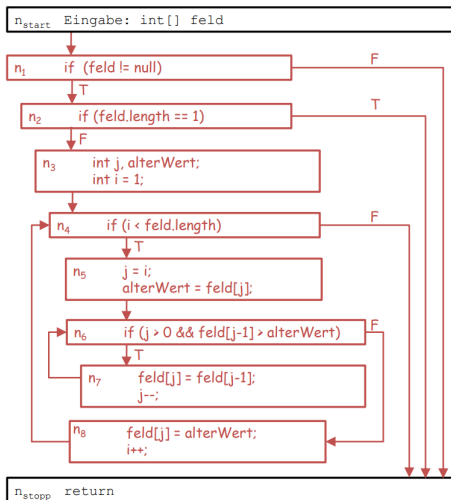


- Finde **alle** vollständige, unterschiedlichen Pfade
- vollständiger Pfad = Anfang bei  $n_{start}$ , Ende bei  $n_{stopp}$
- nicht praktikabel, da
  - Schleifen die Anzahl der möglichen Pfade stark erhöhen
  - manche Pfade nicht ausführbar sind (sich gegenseitig ausschließende Bedingungen)

```
01 public void sortiere(int[] feld) {  
02     if (feld != null) {  
03         if (feld.length == 1) {  
04             return;  
05         } else {  
06             int j, alterWert;  
07             for (int i = 1; i < feld.length; i++) {  
08                 j = i;  
09                 alterWert = feld[i];  
10                 while (j > 0 && feld[j - 1] > alterWert) {  
11                     feld[j] = feld[j - 1];  
12                     j--;  
13                 }  
14                 feld[j] = alterWert;  
15             }  
16         }  
17     }  
18 }
```

Erstellen Sie den Kontrollflussgraphen und geben Sie einen Pfad an, der Anweisungsüberdeckung erzielt.





Pfad:  $(n_{start}, n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_6, n_8, n_4, n_{stopp})$

- Ich kenne die Klausur auch nicht!

- Ich kenne die Klausur auch nicht!
  - ⇒ alles, was ich zum Inhalt der Klausur sage ist Spekulation
    - basierend auf Altklausuren

- Ich kenne die Klausur auch nicht!  
     $\implies$  alles, was ich zum Inhalt der Klausur sage ist Spekulation
  - basierend auf Altklausuren
- kein Anspruch auf Vollständigkeit der Wiederholung

- 1 Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen



- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens:
  - 1-2 Aufgaben zu UML-Diagrammen

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität
  - 1 Aufgabe zu Testen bzw. Qualitätssicherung

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](http://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität
  - 1 Aufgabe zu Testen bzw. Qualitätssicherung
  - 1 Aufgabe Rest (z.B. Objektorientierung, Abbott, Prozessmodelle...)

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](http://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität
  - 1 Aufgabe zu Testen bzw. Qualitätssicherung
  - 1 Aufgabe Rest (z.B. Objektorientierung, Abbott, Prozessmodelle...)
- $1/3 \pm \epsilon$  der Punkte reichen zum Bestehen

- Lastenheft, Pflichtenheft

- Lastenheft, Pflichtenheft
  - Phasen zuordnen



- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben
- UML-Diagramme

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben
- UML-Diagramme
  - Klassendiagramm

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben
- UML-Diagramme
  - Klassendiagramm
  - Aktivitäts-, Sequenz-, Zustandsdiagramm

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben
- UML-Diagramme
  - Klassendiagramm
  - Aktivitäts-, Sequenz-, Zustandsdiagramm
  - Anwendungsfalldiagramm

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben
- UML-Diagramme
  - Klassendiagramm
  - Aktivitäts-, Sequenz-, Zustandsdiagramm
  - Anwendungsfalldiagramm
  - Syntax kennen!

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben
- UML-Diagramme
  - Klassendiagramm
  - Aktivitäts-, Sequenz-, Zustandsdiagramm
  - Anwendungsfalldiagramm
  - Syntax kennen!
  - gegebenen Text in Diagramm umwandeln



- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen
  - Beispiele geben
- UML-Diagramme
  - Klassendiagramm
  - Aktivitäts-, Sequenz-, Zustandsdiagramm
  - Anwendungsfalldiagramm
  - Syntax kennen!
  - gegebenen Text in Diagramm umwandeln
  - bei Zustandsd.: Umwandeln hierarchisch  $\Leftrightarrow$  nicht-hierarchisch

## ■ Architekturstile

- Architekturstile
- **Entwurfsmuster**

- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen

- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen

- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen
  - Klassendiagramm hinzeichnen

- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen
  - Klassendiagramm hinzeichnen
  - aus Klassendiagrammen Entwurfsmuster erkennen

- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen
  - Klassendiagramm hinzeichnen
  - aus Klassendiagrammen Entwurfsmuster erkennen
  - Code für einfache Muster (Singleton... ) schreiben



- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen
  - Klassendiagramm hinzeichnen
  - aus Klassendiagrammen Entwurfsmuster erkennen
  - Code für einfache Muster (Singleton. . . ) schreiben
  - Code-Schnipsel auf mögliche Verbesserung durch EM untersuchen

## ■ UML-Abbildung

- UML-Abbildung
- **Parallelität**

- UML-Abbildung
- **Parallelität**
  - grundlegendes Prinzip

- UML-Abbildung
- **Parallelität**
  - grundlegendes Prinzip
  - in Java

- UML-Abbildung
- **Parallelität**
  - grundlegendes Prinzip
  - in Java
  - critical sections/ race conditions

- UML-Abbildung
- **Parallelität**
  - grundlegendes Prinzip
  - in Java
  - critical sections/ race conditions
  - deadlock

- UML-Abbildung
- **Parallelität**
  - grundlegendes Prinzip
  - in Java
  - critical sections/ race conditions
  - deadlock
  - Monitore, wait & notify



- UML-Abbildung
- **Parallelität**
  - grundlegendes Prinzip
  - in Java
  - critical sections/ race conditions
  - deadlock
  - Monitore, wait & notify
  - Semaphore

- Klausuren rechnen  $\wedge$  Folien anschauen
  - > Klausuren rechnen XOR Folien anschauen

Das war's dann wohl...

Viel Erfolg bei der Klausur und im weiteren Studium! :)

SIMPLY EXPLAINED

