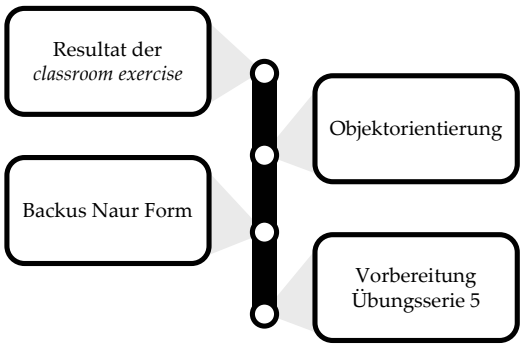
Montag, 24. November



Notenschnitt

3.63

mit 36½ Punkten

Auswertung

- ⅓ war genügend
- ⅜ unter der Note 3

- Definitionen ok
- Beispiele fehlten häufig

- Programmierteil schlecht

Was nun...

- Korrektur studieren
- mit Lösung vergleichen
- meine Skripte lesen
- Repetition in den nächsten beiden Übungsstunden

erster Blick auf ein riesiges Programm

Objektorientierung

Aufteilung des gesamten Problems in viele kleine einfachere Probleme

immer noch Monster, aber weniger gefährlich, wenn man sie (Klassen) im Griff hat...

Teilprobleme sind miteinander verbunden

Abstraktion

was muss die Toolbar alles können?

- Schaltflächensymbole anzeigen
- etwas machen, wenn Schaltflächensymbole angeklickt werden

was müssen Schaltflächensymbole alles können?

- markiert sein, wenn aktiv

Abstraktion in Eiffel

jede Einheit ist in Eiffel definiert durch eine Klasse

```
class TOOLBAR
feature
   -- was die TOOLBAR alles kann
   zeige_Schaltflaechensymbole is
        do
              -- tu' dies und das...
        end
   fuege_Schaltflaechehinzu( welche: SCHALTFLAECHE) is
        do
              -- entsprechende Dinge vornehmen...
        end
   -- usw.
end
```

Klassen und deren Objekte

jede Klasse ist eine Definition der Menge von möglichen Objekten:

- sie definiert, welche Zustände das Objekt annehmen kann
  z.B. Objekte der Klasse *INTEGER* können alle natürlichen ganzen Zahlen als Zustand annehmen
- sie definiert das Verhalten
  z.B. Objekte der Klasse *INTEGER* haben das Feature *out*, welches immer den Zustand des Objektes zurückgibt

Objekte erstellen

```
meine_Route : ROUTE
```

wir wollen jetzt, dass die Entity *meine_Route* auf ein neues Objekt der Klasse *ROUTE* zeigt:

```
create meine_Route
```


geht nicht, weil wir bei der Definition von *ROUTE* sehen:

```
class ROUTE create make_route
feature make_route( seg : ROUTE_SEGMENT ) ... usw.
```

ROUTE will also immer mit make_route kreiert werden:

```
create meine_Route.make_route( ein_Segment )
```

ein feature call

```
feature
   meine_Route : ROUTE
   ein_Segment : ROUTE_SEGMENT
   mach_etwas is
   do
      create meine_Route.make_route( ein_Segment )
   end
```

geht das?

ein_Segment zeigt noch auf kein Objekt – also auf *Void*

was passiert?

eine Vorbedingung wird verletzt

```
class interface ROUTE create make_route
feature make_route( seg : ROUTE_SEGMENT )
    -- create route with first route segment set to ‚seg‘
require
    seg /= Void
... usw.
```

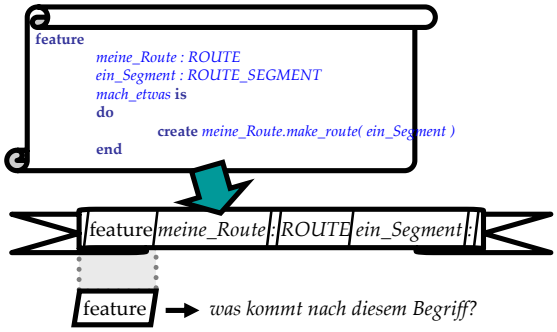
seg ist in diesem Aufruf gleich *Void*!

die Bedingung wird verletzt – das Programm hält an!

ein_Segment kreieren

```
feature
    meine_Route : ROUTE
    ein_Segment : ROUTE_SEGMENT
    mach_etwas is
    do
        create meine_Route.make_route( ein_Segment )
    end
```

Kompilieren



Backus-Naur-Form

John Backus

Peter Naur

Backus und Nauer haben sich beim Erstellen von Compiler mit einer Notation für Grammatiken beschäftigt