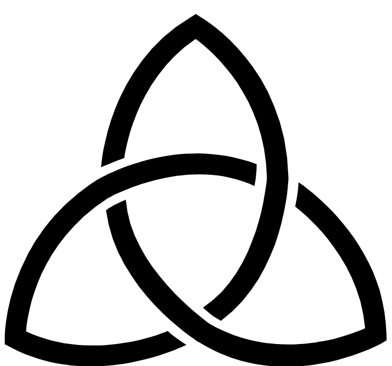
Sprachbeschreibung und Erweiterung

Worte, Sprachen, reguläre Ausdrücke, Automaten, BNF, Grammatik, Syntax-Diagramme, Spracherweiterungen do, for, break, switch

Formale Beschreibung von Programmiersprachen

- Programmiersprachen haben drei Facetten
  - **Syntax** :
    - Was ist ein textuell akzeptables Programm
    - Wird von jedem Compiler überprüft
  - **Semantik**
    - Was leistet ein bestimmtes Programm
      - Welchen Effekt haben die Anweisungen
      - Welchen Wert liefern die Ausdrücke
    - Möglichst mathematisch definierbar
    - Unabhängig von Rechner oder Betriebssystem
  - **Pragmatik**
    - Wie sollte man Programme in dieser Sprache schreiben
      - Erleichtert die Kommunikation zwischen Programmierern
    - Nicht verbindlich, man sollte sich aber dran halten, z.B.:
      - Großschreibung von Klassen, Kleinschreibung von Variablen/Feldern



©H. Peter Gumm, Philipps-Universität Marburg

Natürliche Sprachen

- Syntaktisch falscher Satz:
  - „Satz das ein langer ist sehr."
  - „Was erlauben Struuuntz ?"

- Syntaktisch richtig aber semantisch falsch:
  - „Das ist ein sehr langer Satz."
- Natürliche Sprachen sind zu
  - unpräzise und
  - ausdrucksstark

daher hat nicht jeder syntaktisch richtige Satz eine Semantik.

  - Beispiel: „Dieser Satz ist falsch."

©H. Peter Gumm, Philipps-Universität Marburg

Worte - Sätze

- Lexikalische Ebene
  - Definiert die Worte der Sprache
  - Analogie: Lexikon für nat. Sprache

- Grammatische Ebene
  - Definiert wie Worte zu korrekten Sätzen gefügt werden dürfen
  - In natürlichen Sprachen:
    - viele Regeln
    - viele Ausnahmen
  - In Programmiersprachen
    - wenige Regeln
    - keine Ausnahmen

©H. Peter Gumm, Philipps-Universität Marburg

Lexikalische Definition

- Die lexikalische Ebene definiert die Worte der Sprache
  - das was man im Wörterbuch oder im Lexikon findet

- Zwei gängige Möglichkeiten der Definition:
  - Reguläre Ausdrücke
    - Auch als Backus-Naur Form (BNF) bekannt
  - Automaten
    - Graphische Notation

©H. Peter Gumm, Philipps-Universität Marburg

Alphabete, Worte, Sprachen

- Ein *Alphabet* Σ ist eine Menge von Zeichen, z.B.:
  - das lateinische Alphabet (a, b, c, …, z, A, B, … , Z) ,
  - ASCII ,
  - UNICODE,
  - oder …
- Ein *Wort über Σ* ist eine beliebige Folge von Zeichen aus *Σ*, z.B.:
  - 011010111101 ist ein Wort über { 0,1 }
  - ahsodesukanee ist ein Wort über {a,b,c,…,z}
  - Die *Länge* eines Wortes ist die Anzahl der Zeichen
  - Das Wort der Länge 0 heißt *leeres Wort*.
    Man schreibt es als ε .
- Eine *Sprache über Σ* ist eine Menge von Worten über *Σ*
  - *Menge aller deutschen Worte* ist eine Sprache über dem lat. Alphabet
  - *Menge aller in Java erlaubten Variablennamen* ist Sprache über Unicode
  - *Menge aller float-Literale* ist eine ASCII-Sprache

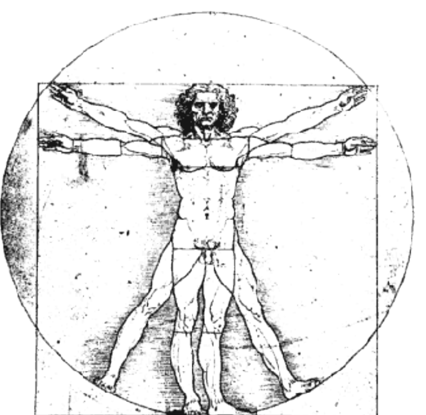
©H. Peter Gumm, Philipps-Universität Marburg

Reguläre Ausdrücke

- dienen zur kompakten und präzisen Beschreibung der Wörter einer Sprache
- Reguläre Ausdrücke werden gebildet aus
  - den Zeichen eines Alphabetes Σ
  - den Operatoren
    - | (oder)
    - ▪ (Verkettung. Das Zeichen ▪ wird meist weggelassen)
    - * (Wiederholung, beliebig oft)
    - ? (optionaler Teil)
  - Klammern (und)
- Beispiele:
  - a (b | c)* : Ein a, gefolgt von beliebig vielel b's und c's
  - (public | private | static | protected | final) : Eines der Worte public, private, ...
  - [(a (, a)*)?] : [ ] oder [a] oder [a,a] oder [a,a,a], oder ...

©H. Peter Gumm, Philipps-Universität Marburg

Reguläre Definitionen



- Eine reguläre Definition ist eine Gleichung
  - < def > ::= regulärerAusdruck
    - Ein regulärer Ausdruck wird mit <def> abgekürzt.
    - <def> darf danach wie ein Zeichen verwendet werden
    - rekursive (zirkuläre) Definitionen nicht erlaubt !!

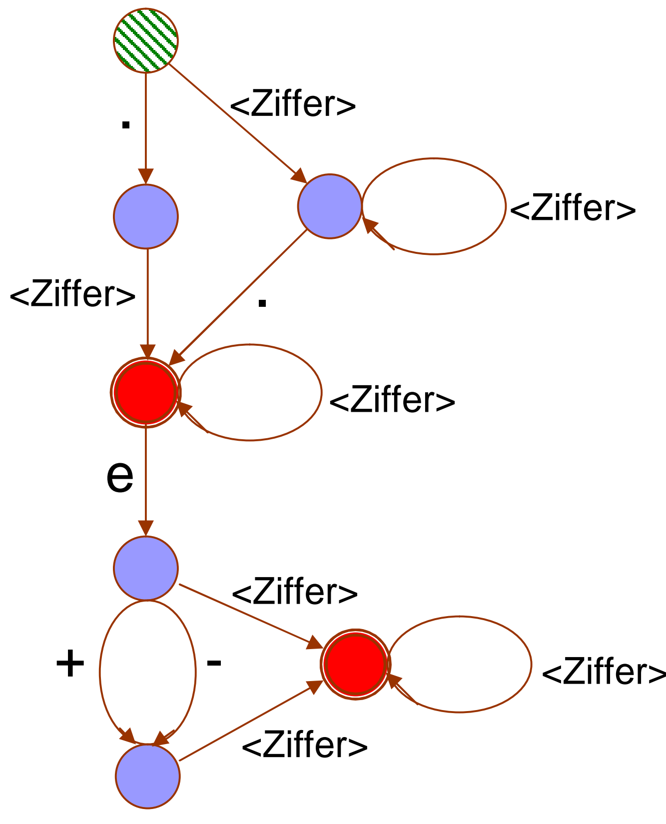
- Mit regulären Definitionen beschreibt man die zulässigen Worte einer Programmiersprache

- Beispiele:
  - <Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
    - Eine Ziffer ist eines der Zeichen 0 ... 9
  - <Zahl> ::= <Ziffer><Ziffer>*
    - d.h.: (eine <Zahl> ist eine nichtleere Folge von Ziffern)
  - <Buchstabe> ::= a | b | … | z | A | B | … | Z
  - <Bezeichner> ::= <Buchstabe> (<Buchstabe> | <Ziffer>)*
    - Ein <Bezeichner> ist eine Folge von <Buchstaben> und <Ziffern>, die mit einem Buchstaben beginnen muss.
  - <float-Literal> ::= <Ziffer>* . <Zahl> (e (+|-)? <Zahl>)?

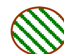
©H. Peter Gumm, Philipps-Universität Marburg

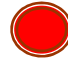
Automaten

- Mit Automaten definiert man die Worte einer Programmiersprache
  - Automaten sind Alternativen zu regulären Definitionen
- Automaten bestehen aus
  - Knoten
  - Pfeilen zwischen Knoten
    - Jeder Pfeil ist mit einem Zeichen beschriftet
  - Ein Knoten ist Anfangsknoten
  - Ein oder mehrere Knoten sind Zielknoten
- Ein Wort ist in der Sprache, wenn es einen Weg vom Anfangs- zu einem Zielknoten beschreibt
- Analogie:
  - Knoten = Städte
  - Pfeile = Einbahnstraßen mit Namen
  - Startknoten = Ausgangspunkt
  - Zielknoten = Fahrtziele
  - Wort = Instruktion (Folge von Straßennamen)



Automat für float-Literale in Java

 Startknoten

 Zielknoten

©H. Peter Gumm, Philipps-Universität Marburg

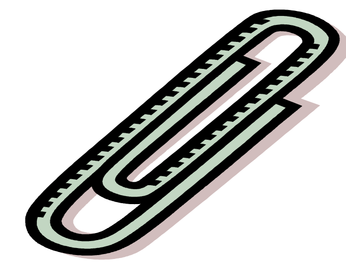
Beschreibung der Grammatik

- Drei Möglichkeiten:
  - EBNF (erweiterte BNF)
  - Kontextfreie Grammatik
  - Syntax-Diagramme

  - EBNF = BNF + Rekursion
  - Kontextfreie Grammatik = reguläre Definitionen + Rekursion
  - Syntaxdiagramme = Automaten + Rekursion

  - Beispiel: Korrekte Klammerung
    - Klammern treten als Paare auf
    - Innerhalb eines Klammerpaars herrscht wieder korrekte Klammerung

  Dies ist nicht durch BNF, reguläre Ausdrücke oder Automaten definierbar



Korrekte Klammerung:

(( ) (( )( ))) ( )

Falsche Klammerung:

(( ) ( ) ( ))) (( )

©H. Peter Gumm, Philipps-Universität Marburg

Kontextfreie Grammatik

- Gleichungen der Form
  - <def> ::= Ausdruck
- Rechts sind nur die Operatoren
  - ▪ Verkettung (wird nicht geschrieben)
  - | Alternative
  - ε (leeres Wort
  - Rekursion ist erlaubt !!
- Beispiele

```
<Klammerung> ::=    <Klammerung> <Klammerung>
                  | "(" <Klammerung> ")"
                  | ε
```

```
<expr> ::= <literal>
         | <identifier>
         | - <expr>            | +<expr>
         | <expr> + <expr>     | <expr> - <expr>
         | <expr> * <expr>     | <expr> / <expr>
         | <expr> % <expr>
         | ( <expr> )
```

Komplette Java-Syntax: http://java.sun.com/docs/books/jls/second_edition/html/syntax.doc.html#44467

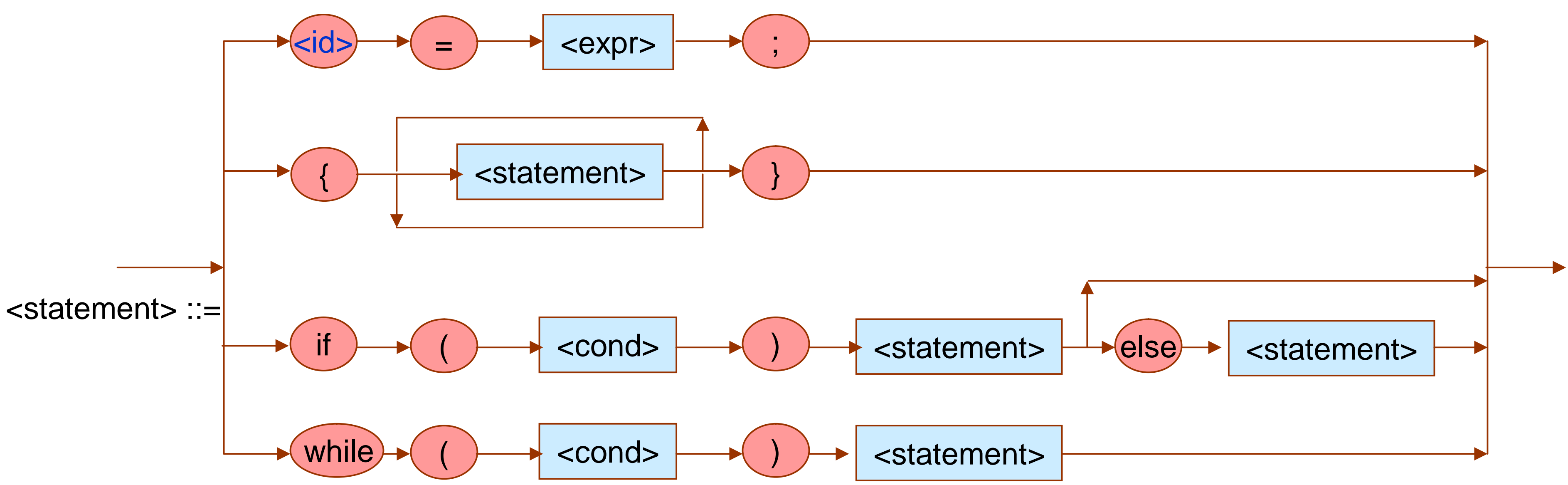
©H. Peter Gumm, Philipps-Universität Marburg

Syntaxdiagramme

- Äquivalent zu Kontextfreien Grammatiken
- <def> ::= Bahn-Diagramm
  - Engl.: railroad-diagram

- Bahn-Diagramme ähneln Automaten, aber:
  - Zustände tragen Beschriftung
    - rund: Worte (Terminale)
    - rechteckig: Definierte Begriffe (Nonterminale)
  - Pfeile ohne Beschriftung
  - Ein Eingang, ein Ausgang
  - Rekursion erlaubt

Beispiel: Syntaxdiagramm für <statement> (Anweisung), soweit bekannt

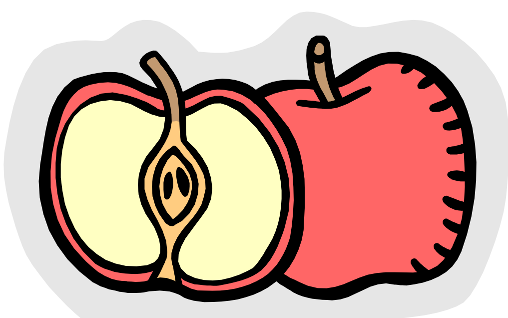


Komplette Java-Syntax: http://cui.unige.ch/db-research/Enseignement/analyseinfo/JAVA/statement.html

©H. Peter Gumm, Philipps-Universität Marburg

Erweiterung der Kernsprache

- Prinzipiell benötigt eine Programmiersprache nur
  - Zuweisung
  - Hintereinanderausführung (bzw. Blöcke)
  - Bedingte Anweisung
  - While-Schleife



- Jeder Algorithmus ist damit programmierbar

- Zusätzliche Konstrukte sind aber praktisch:
  - Fallunterscheidung (switch bzw. case)
  - do-Schleife (repeat)
  - for-Schleifen
  - loop .. Exit

- Wir definieren diese Konstrukte mit Hilfe der bereits bekannten, indem wir
  - Syntax
    - mittels kontextfreier Grammatik oder Syntaxdiagrammen
  - Semantik
    - durch gleichwertige Konstrukte der Kernsprache

  angeben

©H. Peter Gumm, Philipps-Universität Marburg

do-Schleife

Für Schleifen, deren Körper mindestens einmal ausgeführt werden.
Danach wird die Wiederholungsbedingung getestet

Syntax

```
do { <anweisung>* }
   while ( <bexpr> ) ;
```


Beispiel

```
import java.util.*;
public class Übung {

 static void eingabe(){
   Scanner ein = new Scanner(System.in);
   int x = -1;
   do {  System.out.println("Ihr Alter bitte");
         x = ein.nextInt();
      } while(x < 0 );
   System.out.println("Alter Knabe");
 }
```

Das Semikolon am Ende der do-Schleife ist notwendig !! IMHO: Ein bug im Sprach-Design.

©H. Peter Gumm, Philipps-Universität Marburg

do Schleife

- Klasse **javax.swing.JOptionPane** hat **statische** Methode
  - **showInputDialog**,

  um einen String einzulesen.
  - **JOptionPane.showInputDialog(** *<Aufforderung>* **)**
- Vorsicht beim String-Vergleich.
  - **==** statt **equals** liefert nicht das gewünschte Ergebnis !

```
static void rateMal(){
    String input, pass = "Ey, mach schon";
    do{
        input = JOptionPane.showInputDialog("Passwort");
      } while( ! pass.equals(input) );
    System.out.println("Willkommen, Meister");
 }
```

Eingabe

Passwort

Ey, ma

OK Abbrechen

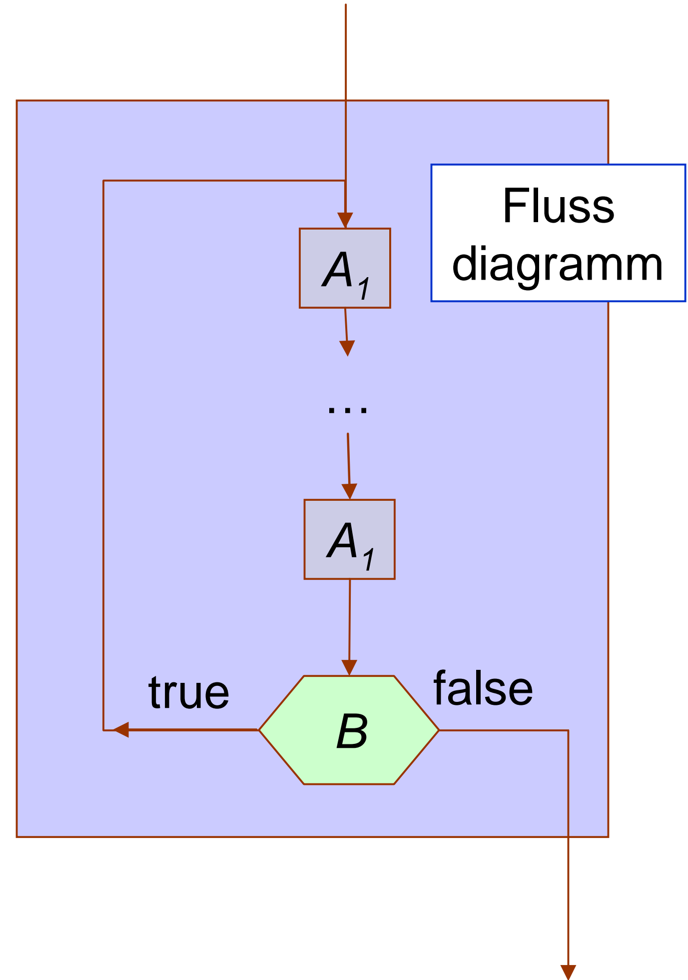
©H. Peter Gumm, Philipps-Universität Marburg

Semantik der do-Schleife

```
do { A1 ... An } while ( B )
```

=

```
{ A1 ... An
   while ( B ) {
     A1 ... An
   }
}
```



©H. Peter Gumm, Philipps-Universität Marburg

for-Schleife

Für Schleifen mit Zähler.

Beispiel

```
static void forTest(){
    for( int i=-40; i<=100 ; i += 10 )
        System.out.println(
            i +"°C = "+(32+9*i/5)+"°F");
}
```

Syntax

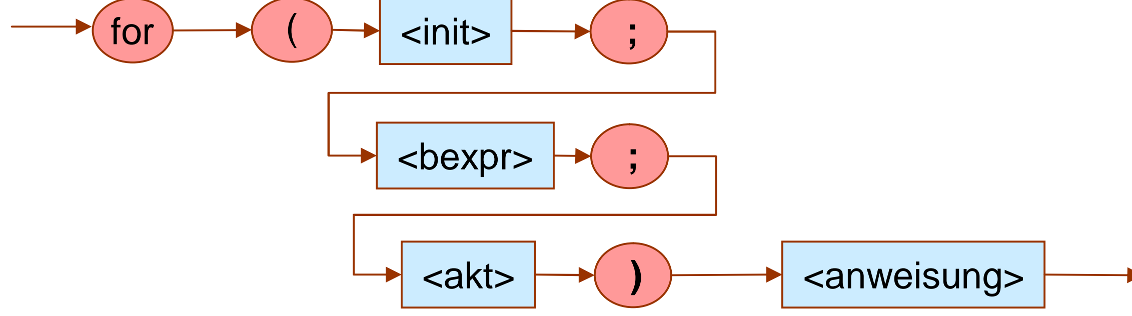
```
for( <init>; <bexpr>; <akt> )
    <anweisung> ;
```

<init> : Initialisierung

<bexpr> : Bedingung für Durchlauf

<akt> : Anweisungsausdruck

<anweisung> : der Körper



©H. Peter Gumm, Philipps-Universität Marburg

Geschachtelte for-Schleifen

- Ein *pythagoräisches Tripel* (a,b,c) besteht aus drei ganzen Zahlen mit $a^2+b^2=c^2$.
  - Beispiele: (3,4,5), (5, 12, 13), ….
- Aufgabe: Finde alle pythagoräischen Tripel mit a,b ≤100.
  - Lösungsidee: Teste alle Paare (a,b) mit a,b ≤ 100.
  - Verbesserung: Es genügt, solche mit a ≤ b zu betrachten.
  - Programm: Erzeuge die Paare mit geschachtelten for-Schleifen

```
class Pythagoras{

  static void alleTripel(int n){
    for (int a=1; a <= n; a++)
        for (int b=a; b <= n; b++){
            int c = ganzeWurzel(a*a+b*b);
            if (a*a+b*b==c*c)
                System.out.println("("+a+","+b+","+c+")");
        }
  }
  static int ganzeWurzel(int n){
    return (int)(Math.sqrt(n));
  }
}
```

Blue...

Options

```
(3,4,5)
(5,12,13)
(6,8,10)
(7,24,25)
(8,15,17)
(9,12,15)
(9,40,41)
(10,24,26)
(11,60,61)
(12,16,20)
(12,35,37)
(13,84,85)
(14,48,50)
(15,20,25)
(15,36,39)
(16,30,34)
```

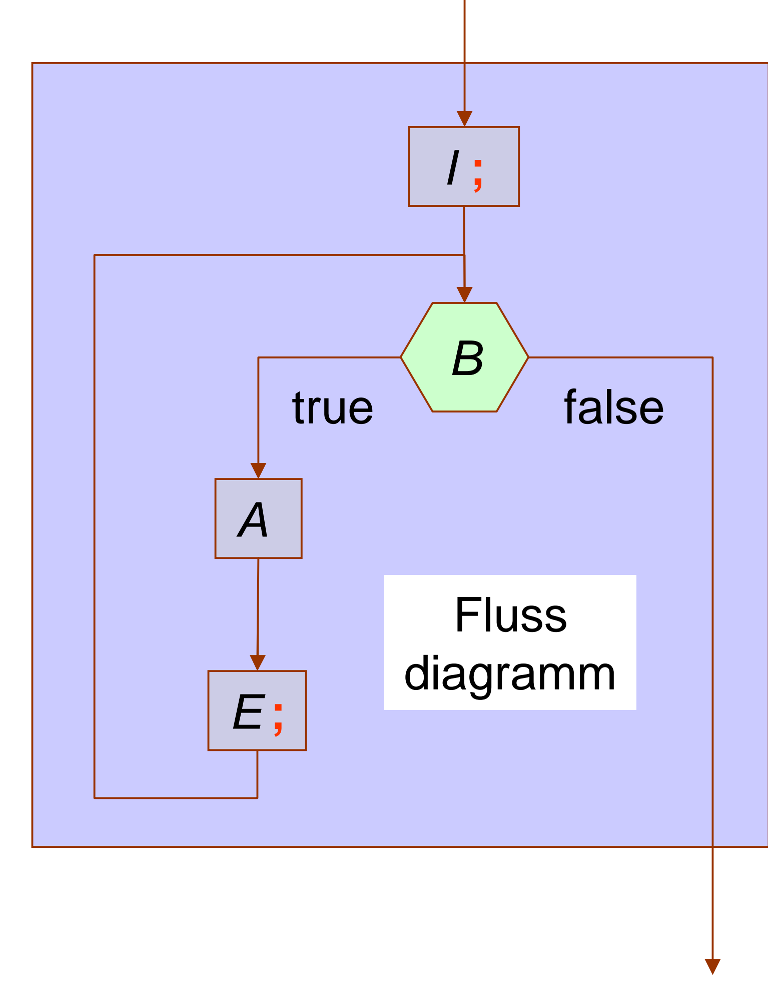

©H. Peter Gumm, Philipps-Universität Marburg

Semantik der for-Schleife

```
for( I ; B ; E )  A
```

=

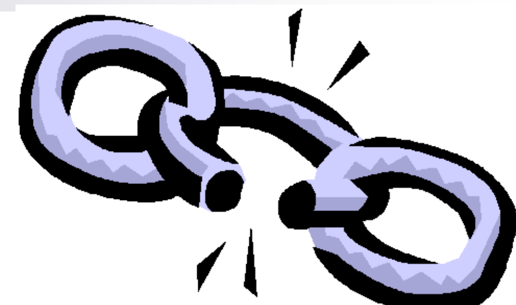
```
{  I ;
   while ( B ) {
      A
      E ;
   }
}
```



Offensichtlich gilt: `for( ; B ; ) A` = `while( B ) A`

©H. Peter Gumm, Philipps-Universität Marburg

break



Syntax

```
break;
```

Semantik

beendet sofort die Schleife bzw. die switch-Anweisung
Verwendbar in while, do, for–Schleifen
und in der switch-Anweisung

```
{ // server
  while(true){
    acceptRequests();
    if (timedOut())
              break;
    tueGutes();
  }
}
```

Pragmatik

vorzeitiges Verlassen einer Schleife
- z.B. wenn ein gesuchtes Element gefunden wurde
- wenn es keinen Sinn macht, den Rest des Körpers auszuführen

©H. Peter Gumm, Philipps-Universität Marburg

continue

Syntax

`continue;`

Semantik

- kann von innerhalb einer Schleife aufgerufen werden
- startet nächste Schleifeniteration

```
/** Lese Zahlen von der Tastatur bis Eingabe == 0.
 *  Ignoriere Zahlen < 10 oder > 60
 *  Schreibe Durchschnittswert auf das Terminal
 */
public static void readComputeAverage(){
  java.util.Scanner in = new java.util.Scanner(System.in);
  System.out.println("Bitte Zahlen eingeben");
  int summe=0, zaehler=0;
  while(true){
     int n=in.nextInt();
     if(n==0)
        break;         // Fertig
     if(n<20 || n >60)
        continue;      // gleich nochmal
     summe+=n;
     zaehler++;
  } // end while
System.out.println("Durchschnitt ist " + (float)summe/zaehler);
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Fallunterscheidung

- **Syntax:**

  switch (<*expr*>) { <fall>* <default>? **}**

  wobei

  <fall> ::= **case** <wert> **:** <anweisung>*

  <default> ::= **default :** <statement>*

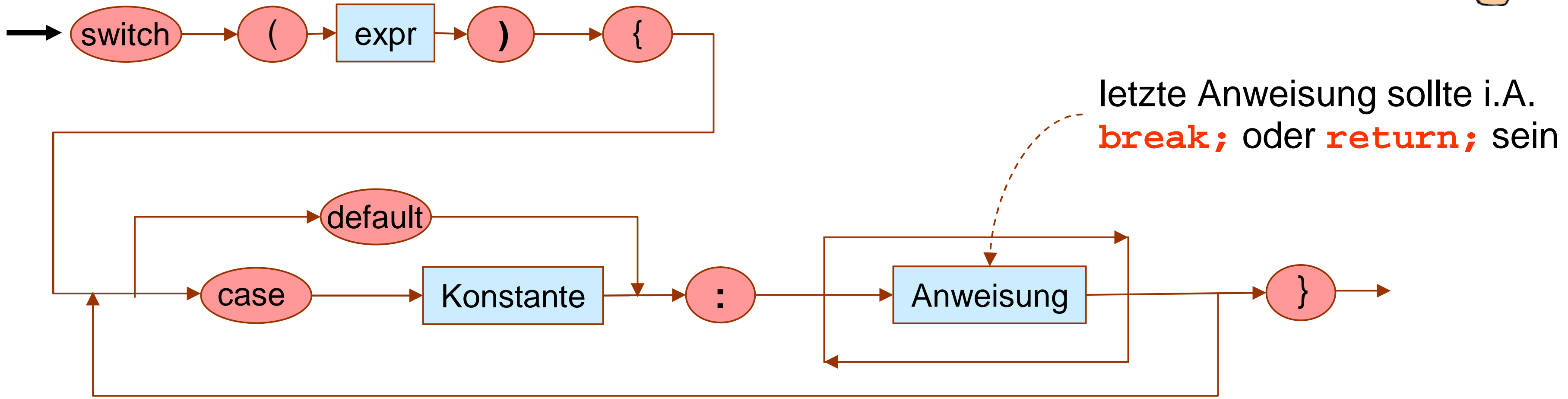
  - <expr> nur von Typ byte, short, int, long, char)

letzte Anweisung sollte i.A. `break;` oder `return;` sein

```
static boolean jaNein(char c){
    switch(c){
        case 'j' :
        case 'y' : return true;
        case 'n' : return false;
        default  : System.out.println("zählt als \"Nein\"");
                   return false;
    }
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Fallunterscheidung



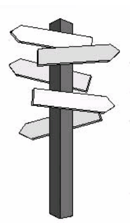
Semantik der switch-Anweisung :

Wenn ein case passt, werden die zugehörigen Anweisungen und die der folgenden Fälle ausgeführt - bis ein break oder return angetroffen

```
public static int tage(int monat, int jahr){
   int tage;
   switch(monat){
      case 2  : tage = schaltJahr(jahr)?29:28; break;
      case 4  :          // fallthrough
      case 6  :          //   -- " --
      case 9  :          //   -- " --
      case 11 : tage = 30; break;
      default : tage = 31;
   } // end switch
   return tage;
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Semantik der switch-Anweisung



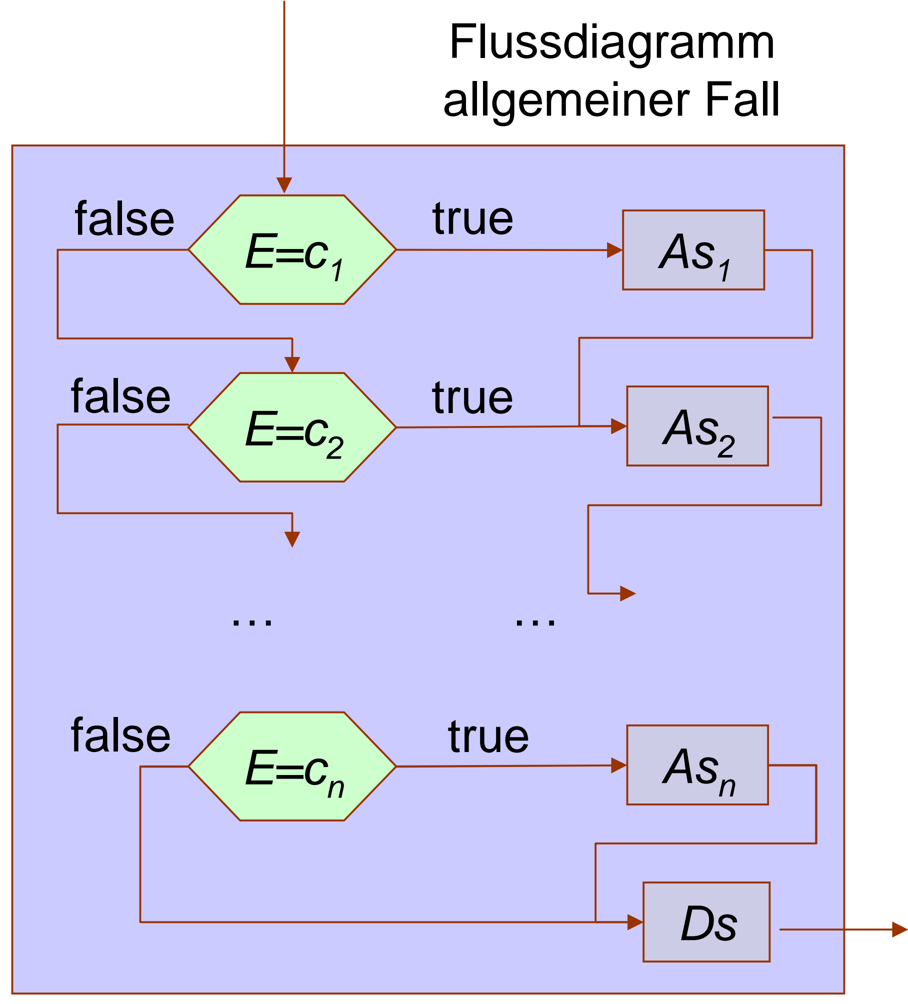
im Normalfall, wenn jeder Fall durch ein break beendet wird :

```
switch( E ){
  case c1  : As1  break ;
  case c2  : As2  break ;
     …
  case cn  : Asn  break ;
  default cn : Ds
}
```

=

```
{  if(  E==c1) As1
   else if (E==c2) As2
     ….
   else if (E==cn) Asn
   else  Ds
}
```

Flussdiagramm
allgemeiner Fall



wer denkt sich so einen Quatsch aus ?
… also bitte: immer schön `break;` verwenden

©H. Peter Gumm, Philipps-Universität Marburg

Anweisungsausdrücke

- Die in modernen Sprachen saubere Trennung
  - Ausdrücke : bezeichnen einen Wert
  - Anweisungen : verändern den Zustand

  wird in Java nicht durchgehalten

- Es gibt Ausdrücke, die den Zustand verändern
  - `x++;   System.out.println( "x = " + x++ + "... denkste");`

- Es gibt Anweisungen, die als Ausdrücke verwendet werden können
  - <id> = <expr> (ohne das Semikolon) ist ein „*Zuweisungsausdruck*"
    - der **Wert** ist der von **<expr>**
    - der **Effekt** ist der der **Zuweisung** <id> = <expr> ;

- Ein Semikolon macht aus einem einfachen Ausdruck eine Anweisung
  - x = 42
    - ist ein *Ausdruck*
  - x = 42 ;
    - ist eine *Anweisung*

- Das hat Java von C geerbt
  - Damit C-Programmierer nicht umlernen müssen

©H. Peter Gumm, Philipps-Universität Marburg

Weitere Anweisungsausdrücke

- Mit Seiteneffekt

(6) Ist v eine Variable, dann sind Ausdrücke:

v++, v--, ++v, --v

Beispiel: `x++ -  x++ == -1`

- Potentiell mit Seiteneffekt

(7) Sind E_1 vom Typ T_1 , …, E_n vom Typ T_n Ausdrücke und

T myFunction(T_1 x_1, …, T_n x_n) { … }

eine Methode mit Rückgabewert vom Typ T, dann ist der *Funktionsaufruf* ein Ausdruck vom Typ T:

myFunction(E_1,…,E_n)

Beispiel: `getKontoStand()-5`

©H. Peter Gumm, Philipps-Universität Marburg