

9. Programmierrichtlinien

Programmierrichtlinien

Code-Konventionen, Style Guides

**Anforderungen an die äußere Form
von Programmen
zur Erhöhung ihrer Lesbarkeit**

- Nicht Teil der Sprachdefinition
(keine Überprüfung durch den Compiler)
- Oft firmenintern
 - z. T. abweichend
 - trotzdem allgemein akzeptierte Prinzipien

Java-Programm: Probleme?

```
class S    {
    int x, y; P p, q;
    public void m()    {
        switch(s)    {
            case c1: d1();
                break;
            case c2:
                d2();    break;
            default: dA();
                break;    }    }
    }
```

Java-Programm: Was fällt auf?

```
class S{
    int x, y; P p, q;
    public void m()    {
        switch(s)    {
            case c1: d1();
            break;
            case c2:
            d2();    break;
            default: dA();
            break;    }    }
}
```

```
class Figur    {
    // nur Ausschnitt
    int x, y;
    Point pLinks, pRechts;

    public void zeichneFigur()    {
        switch(typFigur)    {
            case LINE:
                drawLine();
                break;

            case CIRCLE:
                drawCircle();
                break;

            default:
                drawAll();
                break;
        }
    }
}
```

Einrückungen und Leerzeilen

```
class Figur {
    int x, y;
    Point pLinks, pRechts;

    public void zeichneFigur() {
        switch(typFigur) {
            case LINE:
                drawLine();
                break;

            case CIRCLE:
                drawCircle();
                break;

            default:
                drawAll();
                break;
        }
    }
}
```

Namenswahl: aussagefähig

Variablen

- Beginn mit Kleinbuchstaben
- Teilworte beginnen groß
- kurz, aber mit Semantik, oft: Substantiv enthalten
- temporäre Variablen (z. B. Laufvariablen):
 - Typ int: i, j, k, m, n
 - Typ char: c, d, ch

`einKunde, actValue, farbe`

Konstanten

- in Großbuchstaben
- Teilworte durch '_' getrennt

`ROT, GRUEN, ROT_GRUEN, ROT_SCHWARZ`

Namenswahl (2)

Methoden:

- Beginn mit Kleinbuchstaben
- Teilworte beginnen groß
- sollten Verben beinhalten

suchen, kundenEintragen

Klassen:

- Beginn mit Großbuchstaben
- Teilworte beginnen groß
- sollten Substantiv sein

**Baum, Kunde, GeometrischeFigur,
Keyboard, System**

Layout: Anordnung des Programms

(Text- und Bildgestaltung)

Ausdrücke:

```
help = a * b;    i++;
```

A dashed rectangular box encloses the code snippet. Blue arrows point to the spaces between 'help' and '=', 'a' and '*', 'b' and ';', and between the two statements. Another blue arrow points to the space between the two statements.

- Operatoren von Operanden durch Leerzeichen trennen, (außer: unäre Operatoren)

```
if (kundenNr > NULL)
```

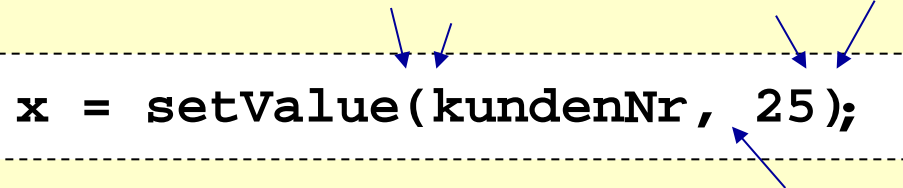
A dashed rectangular box encloses the code snippet. Blue arrows point to the spaces between 'if' and '(', 'kundenNr' and '>', and between '>' and 'NULL'. Another blue arrow points to the space between the closing parenthesis and the opening curly brace.

- Leerzeichen innerhalb / außerhalb von Klammern

Layout (2)

Methodenaufruf, -deklaration

```
x = setValue(kundenNr, 25);
```

A dashed rectangular box encloses the code line. Blue arrows point from above to 'setValue', 'kundenNr', and '25'. A blue arrow points from below to '25'.

```
private static setValue(int kunde, String s) {
```

A dashed rectangular box encloses the code line. Blue arrows point from above to 'setValue', 'kunde', and 'String s'. A blue arrow points from below to 's'. Another blue arrow points from below to the opening brace '{'.

Layout (3)

Anweisungen, insb. Blöcke: { ... }

C-Stil

```
while (a > b) {  
    ...  
}
```

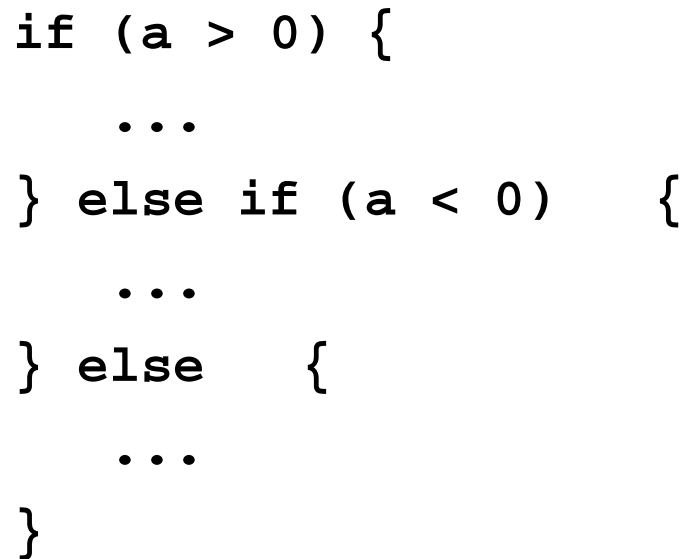
Pascal-Stil

```
while (a > b)  
{  
    ...  
}
```

Layout (4)

If – Anweisung

```
if (a > 0) {  
    ...  
} else if (a < 0) {  
    ...  
} else {  
    ...  
}
```

The code is enclosed in a dashed rectangular box. Two blue arrows point from the right side of the box towards the closing curly braces of the 'else if' and 'else' blocks.

Layout (5)

```
class Shapes {  
    int x, y;  
    Point pFirst, pSecond;  
  
    public void selectColor () {  
        switch (selectedShape) {  
            case LINE:  
                drawLine();  
                break;  
        }  
    }  
}
```

Leerzeile zwischen
Variablen und
Methoden

je Zeile höchstens
eine Anweisung

Layout (6)

Einrückungstiefe: 2, 3, 4

Zeilenlänge: < 80 Zeichen

zu lange Zeilen:

Methoden-
aufruf

```
x = setValues(kundenNr, kontoNr,  
              autoNr, fahrradNr);
```

8 Zeichen

nach dem
Komma

Ausdruck

```
nameV1 = nameV2 * (VAL1 + xxxx)  
         * (VAL2 + yyyy);
```

dieselbe
Spalte

Operator: neue Zeile

Sonstiges

- Kommentierung: ausreichend und aussagefähig
- Eine Variablendeklaration pro Zeile
- Methoden durch eine Leerzeile voneinander getrennt
- In Methoden: lokale Variablen und Anweisungen durch Leerzeilen getrennt
- In Methoden: verschiedene logische Abschnitte einer Methode durch Leerzeilen getrennt
(z.B. Eingabe, 1. Teilalgorithmus, ... Ausgabe)

Programmierrichtlinien: Adressen

- **JAVA Coding Conventions von Oracle (SUN)**

<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

- **JAVA Coding Standards von Ambysoft Inc**

<http://www.ambysoft.com/essays/javaCodingStandards.html>

**Tools zur automatischen
Überprüfung des Programmierstils**

Tool 'AssessStyle': automatische Stilüberprüfung

**Einbeziehung in das Praktikum in GdP
(in den letzten Jahren)**

*automatische Bewertung
der Praktikumsprogramme mit Punktabzügen*

→ jetzt: Plugin in Eclipse zur Stilprüfung