

Programmierrichtlinien für die Programmiersprache

Java

(Stand: März 2007)

Die Programmierrichtlinien geben Hinweise zur Gestaltung und Dokumentation der Software und bewirken, dass das Programm einfacher zu lesen, zu testen, zu warten, zu portieren und zu bedienen ist. Solche Regeln sind vielleicht *nicht lebensnotwendig* für die kleinen Programme, die im Rahmen der Programmierausbildung an der Hochschule erstellt werden, aber zwingend, wenn größere Programme in einer Firma entwickelt werden. Jede Firma hat im Allgemeinen *ihre* Programmierrichtlinien, damit die Programme auf der einen Seite ein einheitliches Erscheinungsbild aufweisen (unabhängig vom jeweiligen Programmierer) und auf der anderen Seite einfach (zumindest teilweise) in anderen Projekten wieder verwendet werden können (Kostensparnis). Da jede Informatikerin und jeder Informatiker früher oder später mit solchen Regeln konfrontiert wird, basiert auch die Programmierung in meinen Lehrveranstaltungen auf Regeln, die die Freiheiten der Programmiersprache sinnvoll einschränken oder nicht unbedingt erforderliche Sprachmittel erzwingen.

Die hier aufgeführten Regeln und Empfehlungen sind bei der Entwicklung der Software einzuhalten bzw. zu berücksichtigen. **Falls ein Programm in erheblichem Umfang von diesen Regeln abweicht, wird es mit *nicht ausreichend* (5.0) bewertet.**

- 1) Das Programm soll nach Möglichkeit nur die von *Sun Microsystems* definierten Sprachmittel verwenden.
- 2) Jede Datei beginnt mit einem *Kopf*, der folgenden Aufbau hat:

```

/* Beschreibung des Dateiinhalts und wichtige Bemerkungen.
 *
 * Datei: <Dateiname>           Autor:   <Name>
 * Datum: <...>                 Version: <Versionsnummer>
 *
 * Historie: <WER hat WANN WAS geaendert / ergaenzt>
 *           Neue Eintraege kommen immer vor die alten Eintraege.
 *
 */
    
```

Die Felder *Version* und *Historie* dürfen bei kleinen Programmen fehlen.

- 3) Jede Klasse beginnt mit einem *Kopf*, in dem alle privaten und öffentlichen Daten und Methoden der Klasse sowie die Funktion der Klasse beschrieben wird. Jede Datei besitzt nur eine öffentliche Klasse (*public class*).
- 4) Jede umfangreichere Methode beginnt mit einem *Kopf*, der mindestens eine Beschreibung der Methode sowie der Ein- und Ausgabeparameter, des Rückgabewertes und der Fehlerausgänge enthält. Kleinere Methoden (weniger als 10 Zeilen Code) dürfen im Kopf der Klasse beschrieben werden.
- 5) Das Programm wird kommentiert. Es werden nur **sinnvolle Kommentare** verwendet, die nicht unmittelbar aus dem Programmtext erkennbar sind. Kommentare sollen das Programm leichter lesbar machen. Sie dürfen keine deutschen Sonderzeichen enthalten.

```

falsch:   int num = 0;           /* weise "num" den Wert 0 zu */
richtig:  int num = 0;           /* number of chars in buffer */
             String hostAddr;      /* name or IP address */
    
```

- 6) Alle Kommentare werden in der gleichen Sprache geschrieben. Englische Kommentare sind im Allgemeinen kürzer.
- 7) Kommentare hinter Anweisungen beginnen in der 41. Spalte und enden vor der 72. Spalte (**Tabulator verwenden!**). Ab der 73. Spalte steht ggf. das Kommentarendenzeichen. Alle Kommentarzeichen stehen untereinander. Längere Kommentare werden vor die Anweisung geschrieben (ab Blocktiefe bis zur 72. Spalte). Der Tabulator muss grundsätzlich auf **acht** Zeichen eingestellt werden. Kommentare hinter Anweisungen dürfen mit dem Kommentarzeichen `/*` oder `//` erstellt werden. Alle anderen Kommentare werden mit den Kommentarzeichen `/*`, `//` oder `/**` (Javadoc-Kommentar) erstellt.

Bemerkung: Da der Quellcode ein Bestandteil der Dokumentation ist, muss er im DIN A4 Format ausgedruckt werden. Bei einer Zeichendichte von 12 cpi (characters per inch; entspricht einer Zeichengröße von 10 pt (point)) können auf einer Zeile bei einem 3 cm großen linken und 2 cm großen rechten Rand max. 75 Zeichen gedruckt werden.

- 8) Mehrzeilige Kommentare werden in der folgenden Form geschrieben:

```

/* ...                               /**
 * ...                               * ...
 */                                  */
    
```

- 9) Programm und Kommentar werden **gleichzeitig** erstellt. Es ist **verboten**, zuerst das Programm zu erstellen und den Kommentar erst dann hinzuzufügen, wenn das Programm „läuft“.

- 10) Es wird entsprechend der Blocktiefe eingerückt. Einrücktiefe: genau **zwei** Zeichen je Block. Zusammengehörnde Schlüsselwörter bzw. Klammern stehen untereinander.

Beispiel:

```

if (...)
{
    Anweisung(en);
}
else
{
    Anweisung(en);
}

if (...)
{
    Anweisung(en);
}
else
{
    if (...)
    {
        Anweisung(en);
    }
}

switch (...)
{
    case ...:
        Anweisung(en);
        break;
    ...
    ...
    default:
        Anweisung(en);
        break;
}
    
```

Die geschweiften Klammern müssen auch dann verwendet werden, wenn nur eine Anweisung vorhanden ist.

- 11) Zwischen einem Funktionsnamen bzw. Schlüsselwort und der öffnenden Klammer sowie den einzelnen Ausdrücken in einer (Parameter-) Liste steht ein Leerzeichen.

Beispiel: `System.out.println ("..." + par1 + ...);`

- 12) binäre und ternäre Operatoren und Operanden werden durch ein Leerzeichen getrennt

Beispiele: `x = a + b;` `z = ((x < y) ? x : y);`

- 13) Die Größe von Anweisungen und Ausdrücken muss überschaubar sein.

- 14) Ausdrücke werden ggf. durch Klammern strukturiert (besser lesbar). Die Anweisung

```
while ((table[i][0] != c) && (i < tab_len))
```

ist verständlicher als die *hinreichende* Java Anweisung

```
while (table[i][0] != c && i < tab_len)
```

oder gar die *notwendige* Anweisung

```
while(table[i][0]!=c&& i<tab_len).
```

- 15) Die Größe der Klassen und Methoden muss überschaubar sein.

- 16) Alle Anweisungen enden vor der 72. Spalte.

- 17) Je Zeile ist nur eine Anweisung erlaubt.

- 18) Anweisungen mit Seiteneffekten müssen vermieden werden.

- 19) Deklarationen erfolgen so lokal wie möglich (Modularisierung, *Information Hiding*).

- 20) Namen müssen sorgfältig gewählt und als Kommentar erklärt werden. Aus dem Namen soll nach Möglichkeit die Bedeutung der Größe abgeleitet werden können. Zur Strukturierung der Namen wird eine Mischung aus Groß- und Kleinbuchstaben oder das Zeichen „_“ verwendet. Namen sollten nicht zu lang gewählt werden, da arithmetische Ausdrücke sonst leicht unübersichtlich werden. Deutsche Sonderzeichen sind in Namen verboten!

Beispiel:

```

servPort          /* Server Port          */
WAIT_TIME         /* waiting time before termination */
    
```

- 22) Klassen- und Schnittstellennamen beginnen mit einem Großbuchstaben und Methoden- und Variablennamen mit einem Kleinbuchstaben. Namen von Konstanten werden nur mit Großbuchstaben geschrieben. Der Name der Klasse, die die Funktion „main“ enthält, sollte auf „Main“ enden (z. B. „ProducerConsumerMain“).

- 23) Gleiche Dinge müssen gleich behandelt werden, z. B. entweder nur deutsche oder nur englische Namen wählen. Mischungen sollen möglichst vermieden werden.
- 24) Speziallösungen sind zu vermeiden (möglichst Standardmethoden verwenden).
- 25) Defensiv programmieren (auch *unmögliche* Fälle abfangen). Eine *switch*-Anweisung erhält z. B. auch dann einen *default*-Zweig, wenn die augenblickliche Programmversion alle möglichen Fälle abdeckt. Auf diese Weise werden bei späteren Programmiererweiterungen Fehler u. U. schneller gefunden, wenn die Erweiterung auch in der *switch*-Anweisung berücksichtigt werden muss (aber nicht berücksichtigt worden ist).
- 26) Nicht zu früh optimieren!
- 27) Falls das Programm eine Eingabe benötigt, wird zuerst eine aussagefähige Eingabeaufforderung ausgegeben (z. B. *Umsatz (in Euro):* oder *linke Intervallgrenze:*).
- 28) Bei fehlerhaften Eingaben wird ein Hilfetext ausgegeben, der den Fehler und die gewünschte Eingabe erläutert, bevor die Eingabe erneut angefordert wird.
- 29) Das Programm wird kommentiert **und** dokumentiert. Die Dokumentation muss mit einem *normalen* Editor oder einem an der Hochschule Fulda *verfügbaren* Textverarbeitungssystem erstellt werden.
- 30) Die Dokumentation besteht aus einer Benutzerbeschreibung und einer Detaildokumentation. Die Detaildokumentation kann mit *Javadoc* erstellt werden.
- 31) Die Benutzerbeschreibung muss für sich allein, d. h. ohne den Rest der Arbeit, verständlich sein. Sie muss die Bedienung des Programms vollständig beschreiben. Es wird empfohlen, die Benutzerbeschreibung ggf. zum besseren Verständnis mit Beispielen zu versehen.
- 32) Die Detaildokumentation des Programms muss so beschaffen sein, dass alle Wartungsarbeiten (Fehlersuche, Programmiererweiterungen usw.) hinreichend schnell durchgeführt werden können. Bei der Detaildokumentation können sowohl die Benutzerbeschreibung als auch das Quellprogramm als vorhanden vorausgesetzt werden, so dass darauf Bezug genommen werden kann. Für die Detaildokumentation sind im Allgemeinen besonders wichtig:
 - a) Eine vollständige Beschreibung sämtlicher Daten und Algorithmen (Variablen, Konstanten, Tabellen, Arbeitsbereiche, Dateien usw.).
 - b) Ein Grobschema des Systems, z. B. ein grobes Ablaufschema, ein Klassendiagramm usw.
 Einzelabläufe können im Allgemeinen unmittelbar dem Quellprogramm entnommen werden. Bei Abläufen mit sehr vielen Kontrollstrukturen sind grafische Hilfsmittel (z. B. UML-Diagramme, Nassi-Shneiderman-Diagramme oder Flussdiagramme) nützlich.
 - c) Hinweise zur Arbeitsumgebung: Compilerversion, besondere Klassen oder Pakete. Auf alle Besonderheiten ist deutlich hinzuweisen.
- 33) Englische Ausdrücke sollen im laufenden Text nach Möglichkeit nicht verwendet werden (z. B. *File*, *Directory*, *Buffer* usw.; besser: *Datei*, *Dateiverzeichnis*, *Puffer*, *Zwischenspeicher*, usw.). Falls kein bekanntes deutsches Wort zur Verfügung steht, wird der englische Begriff in Anführungszeichen gesetzt oder kursiv geschrieben.
- 34) Es ist **verboten**, englische Wörter mit deutschen Vorsilben zu versehen oder zu steigern (z. B. *gescannt*, *gelinkt*, *geschiftet*, *Verclusterung*).

- 35) Die Funktionsfähigkeit des Programms **muss** präsentiert werden (beim Übersetzen des Programms darf der Compiler auch dann keine Warnungen ausgeben, wenn **alle** möglichen Warnungen in der strengsten Stufe aktiviert werden). Zur Präsentation müssen ggf. geeignete Beispiele vorbereitet werden.

Beispiel

Im folgenden wird ein kleines Programm vorgestellt, das mit Javadoc kommentiert wurde.

```
/**
 * You can automatically generate a documentation with "javadoc" if
 * you use javadoc-style comments. The comments can contain HTML-code.
 * <br><br>
 *
 * The documentation of a class, method, variable or constant (called
 * 'field' in Javadoc) must directly precede the corresponding
 * statement. If you don't use packages you will see "&lt;Unnamed&gt;"
 * in some places of the documentation. You can create a directory
 * "doc-files" containing images and complete HTML files whose content
 * would overwhelm the comments of a normal Java source file. You must
 * choose filenames that contain characters which are not allowed in
 * class names (e.g.: "-") to prohibit that "javadoc" will process
 * these files or the files in such directories. Javadoc will copy
 * the directory "doc-files" with all files to the destination
 * directory of your documentation so that you can use these files in
 * this document. Javadoc supports a lot more which you can read in
 * the corresponding manual page or on the web.<br><br>
 *
 * This program determines the <b>greatest common divisor</b> (gcd)
 * and <b>least common multiple</b> (lcm) of two natural numbers.
 * The greatest common divisor can be determined with Euclid's
 * algorithm. You can find a description of both algorithms on
 * wikipedia or in the book from Knuth.<br><br>
 *
 * <table>
 *   <tr valign="top">
 *     <td width="30%" align="left" valign="top">
 *       Class file generation:
 *     </td>
 *     <td width="70%" align="left" valign="top">
 *       <code>javac GcdLcmOneMain.java</code>
 *     </td>
 *   </tr>
 *   <tr valign="top">
 *     <td width="30%" align="left" valign="top">
 *       Usage:
 *     </td>
 *     <td width="70%" align="left" valign="top">
 *       <code>java GcdLcmOneMain</code>
 *     </td>
 *   </tr>
 *   <tr valign="top">
 *     <td width="30%" align="left" valign="top">
 *       Documentation:
 *     </td>
 *     <td width="70%" align="left" valign="top">
 *       <code>javadoc -d html_1 GcdLcmOneMain.java</code><br>
 *       <code>javadoc -d html_1 -author -version -private
 *         GcdLcmOneMain.java</code>
 *     </td>
 *   </tr>
 * </table>
```



```

        System.out.println ("You've only submitted one of two " +
                            "integer parameters.");
        b = DEFAULT_B;
        System.out.println ("I use my default 2nd integer " + b);
        break;
    case 2:                                // two parameters available
        try
        {
            a = Integer.parseInt (argv[0]);
        } catch (NumberFormatException e)
        {
            System.err.println ("Input parameter error");
            System.err.print  (" \" + argv[0] + "\" + " isn't an " +
                               "integer.");

            a = DEFAULT_A;
            System.err.println (" I use my default 1st integer " + a);
        }
        try
        {
            b = Integer.parseInt (argv[1]);
        } catch (NumberFormatException e)
        {
            System.err.println ("Input parameter error");
            System.err.print  (" \" + argv[1] + "\" + " isn't an " +
                               "integer.");

            b = DEFAULT_B;
            System.err.println (" I use my default 2nd integer " + b);
        }
        break;
    default:                                // no parameters or too many
        System.out.println ("I'm computing the greatest common " +
                            "divisor and least common multiple");
        System.out.println ("of two positive integers.");
        System.out.println (" Usage: java GcdLcmOneMain " +
                            "<1st number> <2nd number>");

        System.exit (0);
    }
    // check parameters
    if (a < 0)
    {
        System.out.println ("Input parameter error");
        System.out.println (" 1st number isn't positive. I use its " +
                            "absolute value.");

        a = Math.abs (a);
    }
    if (b < 0)
    {
        System.out.println ("Input parameter error");
        System.out.println (" 2nd number isn't positive. I use its " +
                            "absolute value.");

        b = Math.abs (b);
    }
    // compute greatest common divisor
    result = ComputeGcdOne.gcd (a, b);
    System.out.println ("a: " + a + "      b: " + b +
                        "      gcd (a, b): " + result);
    // compute least common multiple
    result = ComputeLcmOne.lcm (a, b);
    System.out.println ("a: " + a + "      b: " + b +
                        "      lcm (a, b): " + result);
}
}

```


