Quelltextkonventionen für Java

Programmieren 1

Richtlinien für die Gestaltung von Java-Quelltexten

1 Warum Quelltextkonventionen?

„Eine Quelltextzeile wird nur einmal geschrieben, aber hundertmal gelesen.“ Dieser Satz veranschaulicht die weitgehend etablierte Erkenntnis, dass ca. 80% der Kosten von Software in die Wartung fließen. Nur sehr selten sind der Autor eines Programms und der Wartungsprogrammierer dieselbe Person. Quelltextkonventionen erleichtern deshalb die Wartung, denn eine einheitliche Gestaltung erlaubt es dem Leser, sich auf die wesentlichen Aspekte eines Quelltextes zu konzentrieren.

Für PR1 gilt: Nur ein Quelltext, der wirklich lesbar ist – sich an die hier genannten Konventionen hält – kann angemessen bewertet werden.

Die hier aufgeführten Konventionen fassen überwiegend die englischsprachigen Java-Konventionen der Firma Sun Microsystems zusammen, die vor etlichen Jahren von Oracle übernommen wurden. (http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html)

2 Klassenstruktur

2.1 Vor einer Klassendefinition steht immer ein **Klassenkommentar**, der den Zweck der Klasse beschreibt. Vor jeder Methode einer Klasse steht stets ein **Methodenkommentar**, der den Dienst beschreibt, den diese Methode einem Klienten bietet. Diese Kommentare nennen wir **Schnittstellenkommentare**.

Ein Quelltextbeispiel zur Orientierung:

```
/**
 * Eine Klasse, die ein sehr einfaches Modell von Girokonten implementiert.
 *
 * @author Axel Schmolitzky
 * @version Oktober 2021
 */
class Girokonto
{
    // Der Saldo des Kontos in Cent
    private int _saldo;

    /**
     * Initialisiere ein Girokonto mit einem Eroeffnungssaldo.
     * @param eroeffnungssaldo der Eröffnungssaldo in Cent.
     */
    public Girokonto(int eroeffnungssaldo)
    {
        _saldo = eroeffnungssaldo;
    }

    /**
     * Zahle einen Betrag auf das Konto ein.
     * @param betrag der einzuzahlende Betrag in Cent
     */
    public void einzahlen(int betrag)
    {
        _saldo = _saldo + betrag;
    }

    /**
     * @return Saldo des Kontos in Cent.
     */
    public int gibSaldo()
    {
        return _saldo;
    }
}
```

2.2 Dieses Beispiel verdeutlicht die **Reihenfolge**, in der die Teile einer Klassendefinition aufgeführt sein sollten:

Exemplarvariablen
Konstruktoren
Methoden

2.3 Das Beispiel verdeutlicht auch, dass **diese Teile** gegenüber der öffnenden geschweiften Klammer der Klassendefinition **um 4 Zeichen eingerückt** sein sollten.

2.4 Inhaltlich verschiedene Abschnitte sollten **durch Leerzeilen voneinander getrennt** werden.

2.5 **Exemplarvariablen sollten immer `private`** deklariert und damit nur innerhalb der Klasse zugreifbar sein.

2.6 Es ist oft nützlich, **Exemplarvariablen speziell** zu **benennen**, im obigen Beispiel etwa mit einem Unterstrich beginnend. Durch eine solche Regelung ist aus dem Quelltext sehr leicht ersichtlich, an welchen Stellen auf Exemplarvariablen zugegriffen wird. Wenn dieser Konvention wie in Programmieren 1 gefolgt wird, sollte sie auch durchgängig eingehalten werden.

2.7 Eine Zeile sollte **nicht mehr als 70 Zeichen** lang sein – bis zu 80 sind im Extremfall zulässig. Dies ist eine Konvention, die in erster Linie der Lesbarkeit (am Bildschirm oder ausgedruckt) für Menschen dient.

3 Kommentierung

Kommentare sind technisch gesehen Abschnitte im Quelltext, die vom Compiler ignoriert werden. Sie sind also niemals Teil einer Programmausführung, sondern dienen ausschließlich der Dokumentation des statischen Quelltextes *für den menschlichen Leser*.

3.1 **Schnittstellenkommentare (für Klassen und Methoden)** sollten lediglich Auskunft über das „Was“ (einer Klasse, einer Methode) geben, nicht aber über das „Wie“. Sie sollten so formuliert sein, dass ein Klient der Klasse weiß, wie er die Klasse oder Methode (welche Parameter zu welchem Zweck etc.) benutzen soll und mit welchen Ergebnissen er bei einer Benutzung rechnen kann. Sie sollten jedoch nicht Details der Implementation verraten, die für einen Klienten irrelevant sind. Für Schnittstellenkommentare sollten die Java-Kommentarzeichen `/**` (Beginn) und `*/` (Ende) verwendet werden. Diese sollten nicht zum Auskommentieren längerer Programmabschnitte verwendet werden, da die Gefahr besteht, dass Zeilen nicht unmittelbar als auskommentiert erkennbar sind.

4 Blöcke, Deklarationen, Anweisungen

4.1 **Geschachtelte Blöcke** sollten stets um 4 Leerzeichen eingerückt sein.

4.2 **Blockklammern** (geschweifte Klammern) stets in **eigene Zeilen** schreiben.

4.3 **Pro Zeile sollte nur eine Deklaration oder Anweisung** stehen.

4.4 **Variablendeklarationen** sollten immer am Beginn des jeweiligen Blockes stehen, in dem Sie verwendet werden. Eine Mischung von Deklarationen und Anweisungen erschwert die Lesbarkeit.

4.5 Bei gewöhnlichen Auswahlanweisungen – `if`-Anweisungen – sollte ein ggf. vorhandenes `else` stets **in einer eigenen Zeile** stehen. Allg. gilt: Verwende immer Blockklammern bei `if`-Anweisungen, diese sind zwar bei genau einer auszuführenden Anweisung nicht notwendig, erhöhen jedoch die Lesbarkeit des Textes. Außerdem ist dies **robuster** gegenüber Änderungen: Wenn beim Eintreten der Bedingung nicht nur eine, sondern auch eine weitere Anweisung ausgeführt werden soll, kann die zweite leicht hinzugefügt werden, ohne dass Klammern eingefügt werden müssen.

4.6 **Boolesche Ausdrücke** sollten immer in der Art geklammert werden, dass sich jeweils nur eine Prüfung, oder eine Verknüpfung zweier Prüfungen innerhalb eines Klammerpaares befindet.

Beispiel:

```
if ((x == y) && (x == z))
{
    ....
}
else if ( ... )
{
    ....
}
```

5 Benennungsregeln

Namen (von Variablen, Typen, Klassen, Methoden, Variablen, etc.) werden im Umfeld von Programmiersprachen auch häufig **Bezeichner** genannt, orientiert am englischen Begriff *identifier*.

In Java sind Groß- und Kleinschreibung signifikant (Java ist „case sensitive“). Die Bezeichner `eins` und `Eins` beispielsweise sind deshalb verschieden.

5.1 **Bezeichner enthalten keine Unterstriche** – die einzige Ausnahme stellen die Exemplarvariablen dar, die mit einem Unterstrich beginnen sollen. Weiterhin sollten **Umlaute** (wie ä, ö und ü) (nicht nur) in Bezeichnern vermieden werden, da diese häufig plattformabhängig unterschiedlich dargestellt werden.

5.2 Die meisten Bezeichner (wie für Methoden und Variablen) beginnen mit einem Kleinbuchstaben, **weitere Wörter innerhalb eines Bezeichners** beginnen immer mit einem Großbuchstaben; diese Konvention wird auch kurz „**Camel Case**“ genannt. Lediglich Klassenbezeichner beginnen zusätzlich auch mit einem Großbuchstaben.

5.3 Für **Bezeichner von symbolischen Konstanten** (Konstanten, die **überall** im Quelltext dieselbe Bedeutung haben sollen, wie beispielsweise `PI`) gelten eigene Konventionen. Sie werden durchgängig aus Großbuchstaben gebildet, Teilwörter werden hier – und nur hier – mit einem Unterstrich voneinander getrennt. Exemplarkonstanten sind keine symbolischen Konstanten in diesem Sinne, da sie für jedes Exemplar einen anderen Wert haben können.

5.4 **Klassen, Methoden und Variablen sollten „sprechend“ benannt werden**, etwa mit ganzen Begriffen oder Kurzsätzen, die die Funktion des Bezeichneten deutlich machen. Als Daumenregel gilt vorläufig: Klassen werden üblicherweise mit Substantiven benannt, Methoden (als die „aktiven“ Einheiten in Objektsystemen) mit Verben. **Methoden** sollten stets nach der Dienstleistung benannt werden, die sie liefern oder die sie anbieten, können also auch substantivisch benannt sein.

Beispiele:

```
class Person {...}
public void aendereVornamenIn(String neuenVornamen) {...}
public String nameMitAnrede() {...}
```