gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

Richtlinien (Konventionen) für die Erstellung von Java Quellcode

1 Dateiorganisation

Jedes JAVA-Source-File enthält genau eine public-Klasse (private-Klassen dürfen in dieselbe Datei geschrieben werden).

Jedes Source-File beginnt mit einem Kommentar-Block folgenden Inhalts:

```
/* Kommentar zur Klasse
* Autor:
*
* Schnittstellen:
*
* Vererbung:
*
* Erstelldatum:
*
* Kurzbeschreibung:
*
*/
```

2 Einrückungen

Zeilenlänge: max. 80 Zeichen

Umbruch innerhalb eines Ausdrucks: Ist innerhalb eines Ausdrucks ein Zeilenumbruch notwendig, dann gelten folgende Tipps:
- Umbruch nach einem Komma
- vor einem Operator
- außerhalb von Klammerungen vornehmen

Die nächste Zeile so einrücken, dass der Zusammenhang zur vorherigen Zeile klar wird.

Blockklammern: geschweifte Klammern stehen im Allgemeinen in einer neuen Zeile. Öffnende und zugehörige schließende geschweifte Klammern stehen immer in der gleichen Spalte. Anweisungen nach einer öffnenden Blockklammer einrücken.

Beispiel:

```
if ((bedingung1 && bedingung2)
        || (bedingung3 && bedingung4))
{
    tuEtwas();
}
```

3 Kommentare

Block-Kommentare

werden verwendet zur Beschreibung von Dateien, Methoden, Datenstrukturen und Algorithmen.
Sie beginnen und enden mit einer Leerzeile:

```
/*
* Block-Kommentar
*/
```

Kommentare von einzelnen Zeilen

Beginnen in der Spalte, in der der folgende Text beginnt:

```
if (bedingung)
{
  /* Behandlung der Ja-Anweisung */
  ...
}
```

Nachfolgende Kommentare (trailing-Comments)

Sehr kurze Kommentare können auch in die selbe Zeile geschrieben werden.
Sie sollen aber weit genug von den Statements entfernt stehen.
Mehrere Kommentare dieser Art beginnen möglichst immer in derselben Spalte:

```
if (bedingung)
{
  status = true;          /* Spezialfall */  oder // Spezialfall
}
else
{
  status = false;         /* Normalfall */   oder // Normalfall
}
```

“banale” Kommentare vermeiden:

```
...
  return true;            /* true wird zurueckgegeben */
}
```

Zeilenende-Kommentar (//)
Nicht verwenden bei Kommentierung mehrerer aufeinanderfolgender Zeilen.

*Kommentare für die automatische Dokumentation mit javadoc /** */*

Das Java SDK bietet auch ein Tool (javadoc), um aus speziell formatierten Quelltext-Kommentaren eine Dokumentation der Software zu generieren. Ohne zusätzliche Informationen erstellt javadoc aus den „Dokumentationskommentaren“ im Quelltext eine brauchbare Beschreibung aller Klassen und Interfaces.

Ein Dokumentationskommentar beginnt mit /** und endet mit */ und ähnelt damit einem gewöhnlichen Kommentar. Er muss im Quelltext immer unmittelbar vor dem zu dokumentierenden Element platziert werden (einer Klassendefinition, einer Methode oder einer Instanzvariable). Er kann aus mehreren Zeilen bestehen. Die erste Zeile des Kommentars wird später als Kurzbeschreibung verwendet.

Innerhalb der Dokumentationskommentare dürfen neben normalem Text auch HTML-Tags vorkommen. Sie werden unverändert in die Dokumentation übernommen und erlauben es damit, bereits im Quelltext die Formatierung der späteren Dokumentation vorzugeben. Die Tags <h1> und <h2> sollten möglichst nicht verwendet werden, da sie von javadoc selbst zur Strukturierung der Ausgabe verwendet werden.

javadoc erkennt des weiteren markierte Absätze innerhalb von Dokumentationskommentaren. Die Markierung muss mit dem Zeichen @ beginnen und - abgesehen von Leerzeichen - am Anfang der Zeile stehen. Jede Markierung leitet einen eigenen Abschnitt innerhalb der Beschreibung ein, alle Markierungen eines Typs müssen hintereinanderstehen.

Häufig verwendete Markierungen in Dokumentationskommentaren

Markierung und Parameter	***Dokumentation***	***Verwendung in***
@author <name>	Erzeugt einen Autoreneintrag.	Klasse, Interface
@version <version>	Erzeugt einen Versionseintrag. Darf höchstens einmal je Klasse oder Interface verwendet werden.	Klasse, Interface
@see <reference>	Erzeugt einen Querverweis auf eine andere Klasse, Methode oder einen beliebigen anderen Teil der Dokumentation.	Klasse, Interface, Instanzvariable, Methode
@param <name> <description>	Parameterbeschreibung einer Methode	Methode
@return <description>	Beschreibung des Rückgabewerts einer Methode	Methode
@exception <classname> <description>	Beschreibung einer Ausnahme, die von dieser Methode ausgelöst wird	Methode
@deprecated <description>	Markiert eine veraltete Methode, die zukünftig nicht mehr verwendet werden sollte.	Methode

Zur Anwendung von javadoc siehe
https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

4 Deklarationen

Nur eine Variable pro Zeile deklarieren.
Zwischen Datentyp und Name steht ein Leerzeichen oder ein Tab:

```
int  iZahl;
int  iNummer;
String    szName;
```

Deklarationen nur am Anfang einer Methode.

Initialisierung
sollte sofort bei der Deklaration oder unmittelbar danach erfolgen, nicht erst vor der ersten Verwendung.
Ausnahme: vor der Initialisierung sind andere Berechnungen notwendig, bzw. die Initialisierung erfolgt durch Benutzereingabe.

```
int iInt = 0;
```

oder

```
int iInt;
iInt = 0;
```

5 Anweisungen

Jede in eine eigene Zeile.

```
i++;
j++;
```

nicht:

```
i++; j++;
```

logisch verknüpfte Bedingungen klammern:

```
if (a == b && c == d)    // vermeiden
if ((a == b) && (c == d))     // richtig
```

6 Namenskonventionen

Variablen
Variablennamen beginnen mit einer Kennung des Datentyps als Präfix. Nach dem Präfix wird mit einem Großbuchstaben weitergeschrieben.

Teilwörter werden mit einem Großbuchstaben begonnen. Unterstriche vermeiden.
Namen sollen aussagefähig, aber nicht zu lang sein.

Variablentyp	**Präfix des Variablennamen**	**Beispiel**
char	c	cZeichen
boolean	b	bWert
byte	by	byWert
String	s	sName
short	sh	sKurzeZahl
int	i	iZähler
long	l	lGrosseZahl
float	f	fKurzerFloat
double	d	dDoppeltGenauerFloat
array	a gefolgt vom Präfix der enthaltenen Datentypen	aiIntegerArray asStringArray
class	C	CMitarbeiter

Symbolische Konstanten:
Symbolische Konstanten werden mit Großbuchstaben geschrieben, Teilwörter werden mit Unterstrich abgetrennt:
static final int MAX_ANZAHL = 999;