Einführung in JavaDoc

1 Einleitung und Motivation

Don't play with that gun. It's bad luck!

Bansho

Java hat drei Arten von Kommentaren; die einen sind die aus C bekannten mit /* und */ eingeschlossenen, die anderen die einzeiligen Kommentare aus C++ mit vorangestellten //. Die dritte sind die, die mit /** beginnen und mit */ enden. Die JavaDoc-Kommentare sind mehrzeilig, wie Kommentare in C.

Die Referenz „How to Write Doc Comments for the Javadoc Tool“ (http://java.sun.com/j2se/javadoc/writingdoccomments/index.html) und die Programmbeschreibung an (http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javadoc.html).

2 Das Format eines JavaDoc-Kommentars

I was thinking... – Don't think!

Brooklyn und Barabbas

JavaDoc-Kommentare werden in HTML geschrieben und müssen vor der jeweiligen Klasse, Attribut oder Methode stehen. Ein JavaDoc-Kommentar besteht aus zwei Teilen, eine Beschreibung gefolgt von einem oder mehreren Tags.

```
/**
 * Gibt den Wert „Wurzel aus -1“ als Datentyp Komplex zurück.
 *
 * @return Wurzel aus -1
 */
public static Komplex i() {
      return new Komplex(0, -1);
}

/**
 * Addiert zwei komplexe Zahlen und gibt das Ergebnis zurück
 *
 * @param a der erste zu addierende Operand
 * @param b der zweite zu addierende Operand
 *
 * @return das Additionsergebnis
 */
public static Komplex add(Komplex a, Komplex b) {
      return new Komplex(a.re + b.re, a.im + b.im);
}

/**
 * Addiert eine Komplexe Zahl zum Komplexen Objekt.
 *
 * @param a die zu addierende komplexe Zahl
 */
public void add(Komplex a) {
      this.re += a.re;
      this.im += a.im;
}
```

Am Beispiel ist zu sehen, daß der Kommentar mit /** beginnt. Die führenden Sternchen in jeder Zeile sind in neueren Versionen von JavaDoc nicht mehr nötig. In der letzten Zeile wird der Kommentar mit */ abgeschlossen. Ein Screenshot des Beispiels findet sich in Anhang A.

3 Schreibstil

You know the true hero doesn't think about what he's doing. He doesn't even know he's a hero. He does things, because he knows he has to.

Sergeant Nim

Der erste Satz eines JavaDoc-Kommentars kommt sowohl in den Index als auch in die Tabelle der Methodenzusammenfassungen. Er sollte eine kurze Zusammenfassung der Methode sein.

Hat eine Beschreibung mehrere Absätze, dann sind sie jeweils durch <*p*> zu trennen.

Idealerweise sollte eine Beschreibung so gut sein, daß man nur mit der Beschreibung zur Hilfe den beschriebenen Code neu implementieren kann.

In einigen Fällen kann man es sich sparen, JavaDoc-Kommentare abzuschreiben. Wenn eine Methode einer Klasse eine Methode der Superklasse überschreibt, dann erzeugt JavaDoc einen Verweis „überschreibt“ mit einem Link auf die überschriebene Methode. Das gleiche gilt, wenn eine Methode in einem Interface eine Methode eines Superinterfaces überschreibt. In einem dritten Fall, wenn eine Methode die Methode eines Interfaces implementiert, erzeugt JavaDoc einen Verweis „specified by“ mit einem Link auf die implementierte Methode.

Die Beschreibung endet mit der ersten Zeile die ein @ enthält. Zwischen der Beschreibung und den @-Tags sollte eine Leerzeile sein. Jeder JavaDoc-Kommentar kann nur einen Beschreibungsblock enthalten. Nach den Tags kann die Beschreibung nicht fortgesetzt werden.

In <*code*>...</*code*> Tags eingeschlossen werden Java-Schlüsselwörter, Namen von Variablen, Klassen, Methoden, Interfaces, Feldern und Argumenten sowie Beispielcode.

Verwende keine Klammern für die allgemeine Form von Methoden und Konstruktoren. Betrachten wir einmal die beiden *add*-Methoden im obigen Beispiel. Es gibt die zwei Formen *add(Komplex, Komplex)* und *add(Komplex)*.

> Die *add(Komplex, Komplex)*-Methode ist statisch; sie addiert zwei übergebene komplexe Zahlen und gibt das Ergebnis zurück.

Wenn man jedoch beide Formen der Methode meint, dann läßt man die Klammern ganz weg. Würde man sie dennoch verwenden, so könnte der Leser annehmen, es gäbe noch eine dritte *add*-Methode, die keine Parameter hat.

Statt Sätzen kann man der Kürze wegen einfach Satzglieder verwenden. Insbesondere der Eingangssatz der Beschreibung und die @*param*-Tags sind dafür geeignet.

Die Bechreibung sollte mit einem Verbausdruck anfangen: „Addiert zwei komplexe Zahlen...“ statt „Diese Methode addiert zwei komplexe Zahlen...“.

Die Beschreibung von Klassen, Interfaces und Feldern kann das Subjekt weglassen: „Eine Schaltflächenbeschriftung“ statt „Dieses Feld ist eine Schaltflächenbeschriftung“.

Vermeide Abkürzungen!

4 Tags

I'm not dead yet.

GunHED UHED-507

Die Tags jeden Kommentares müssen in einer bestimmten Reihenfolge aufgeführt werden. Sie sind in dieser Reihenfolge beschrieben.

@author: Das @*author*-Tag ist optional. Sie können ein, kein oder mehrere @*author*-Tags verwenden. Werden mehrere @*author*-Tags verwendet, so sollten sie chronologische geordnet sein, der Erzeuger der Klasse zuerst.

> *@author Zaphod Beeblebrox*

@version: Statt einer eigenen Versionsnummer ist es auch möglich, die Versionsnummern von Sourcecodeverwaltungssystemen wie SCCS zu verwenden.

> *@version 1.02*

@param: Dem @*param*-Tag folgt zuerst der Name (nicht der Datentyp) des Parameters, gefolgt von einer Beschreibung des Parameters. Es hat sich eingebürgert, als erstes Hauptwort in der Beschreibung den Datentyp des Parameters zu verwenden, wobei davor noch ein Artikel („ein“, „der“, &c) stehen kann. Bei dem primitiven Datentyp int wird der Datentyp üblicherweise weggelassen.

Parameternamen fangen mit Kleinbuchstaben an um sie von Klassennamen zu unterscheiden. Ist die Beschreibung im @*param*-Tag ein Satzglied, dann fängt sie mit einem Kleinbuchstaben an, mit einem Großbuchstaben, wenn es ein vollständiger Satz ist.

Der Parametername nach dem *@param*-Tag wird nicht in <*code*>...</*code*>-Tags eingeschlossen.

@param Farbe die Farbe, die der Kreis haben soll.

@*return*: Für Methoden die nichts zurückgeben und Konstruktoren sollte kein *@return*-Tag angegeben werden. Bei allen anderen Methoden empfiehlt es sich etwas zu schreiben, auch wenn die Beschreibung schon alles enthält. Dadurch läßt sich diese Information leichter finden, wenn man nur danach sucht. Wenn möglich sollte man hier auch Rückgabewerte für Spezialfälle aufführen.

@return den gewürfelten Wert.

@*throws* und **@*exception***: Die beiden Tags sind synonym. Üblicherweise wird nur noch *@throws* verwendet. Für jede checked Exception (die in der throws-Klausel deklariert wird) sollte es ein *@throws*-Tag geben. Genauso sollte es für jede unchecked Exception ein *@throws*-Tag geben, wenn man davon ausgehen kann, daß der Aufrufer der Methode diese eventuell abfangen möchte.

@throws IOException wenn eine Ein- oder Ausgabe-Exception vorkommt.

@*see*: Hiermit wird ein Verweis auf ein anderes Element der Dokumentation hinzugefügt. Es gibt drei Arten von Verweisen. Die erste ist einfach ein String, der von Hochkommata begrenzt wird. Die zweite ist eine URL, wie sie in HTML verwendet wird: <*a href="spec.html#section"Java Spezifikation</a*>. Die dritte verweist auf andere JavaDoc-Elemente. Dabei kann man jedes gültige Programmelement in der Form *Paket.Klasse#Member* angeben. Zusätzlich kann noch eine Beschriftung angegeben werden. Gibt es keine Beschriftung, so wird der Verweis in der Form *Paket.Klasse.Member* angezeigt werden.

@see semester2.Graphobjekt#flaeche Fläche des Graphobjektes

5 Kommandozeilenoptionen

Odds don't mean anything when humans want something bad enough.

GunHED UHED-507

-*public*: Die Dokumentation wird nur für öffentliche Klassen und Member erstellt.

-*protected*: Die Dokumentation wird für geschützte und öffentliche Klassen und Member erstellt. Dies ist die Voreinstellung

-*package*: Die Dokumentation wird für Klassen und Member der Sichtbarkeiten Paket, geschützt und öffentlich erstellt.

-*private*: Die Dokumentation wird für alle Klassen und Member erzeugt.

6 Epilog

The GunHED Battalion has completed its mission.

GunHED UHED-507

Diese Einführung zeigt nur einige einfache Möglichkeiten von JavaDoc. Eine vollständige Dokumentation findet sich unter den beiden Internetlinks, die in der Einleitung genannt wurden. Dort gibt es auch weitere Ratschläge, wie man eine gute Dokumentation aufbauen kann.

Die sorgfältige Erstellung von Dokumentation wird oft als zeitraubend betrachtet. Langfristig ist sie jedoch fast immer hilfreich; in einer Menge von mehreren hundert Klassen und Dutzenden von Paketen ist es sonst unmöglich etwas zu finden. Gleichzeitig ist man während des Schreibens gezwungen noch einmal über die Klasse und Methode nachzudenken, was manchen Fehler aufdeckt, bevor er an den Kunden ausgeliefert wird.

7 Anhang A – Screenshot

add

```
public void add(Komplex a)
```

Addiert eine Komplexe Zahl zum Komplexen Objekt.

Parameters:
a - die zu addierende komplexe Zahl

---

add

```
public static Komplex add(Komplex a,
                          Komplex b)
```

Addiert zwei komplexe Zahlen und gibt das Ergebnis zurück

Parameters:
a - der erste zu addierende Operand
b - der zweite zu addierende Operand
Returns:
das Additionsergebnis

8 Anhang B – Ein vollständiges Beispiel

Party time, now, GunHED.

Brooklyn

```
package de.xyzzy.java.tools;

/*
 * Pair.java
 *
 * (c) 2005 Robert Klein
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted under the BSD License.
 */

/**
 * implementiert eine Klasse, deren Instanzen in einem TreeSet
 * nach der "nat&uuml;rlichen" Ordnung der Paarteile geordnet
 * sind, wobei das erste Paarelement das h&ouml;herwertige ist.
 *
 * @author Robert Klein
 * @version 1.0
 *
 * @param <A>
 *            der Typ des ersten Paarelements.
 * @param <B>
 *            der Typ des zweiten Paarelements.
 */
public class Pair<A extends Comparable<A>, B extends Comparable<B>>
           implements Comparable<Pair<A, B>> {
      public A e1;

      public B e2;

      /**
       * erzeugt eine Instanz mit den &uuml;bergebenen Parametern
       * als Inhalt.
       *
       * @param e1
       *            Erstes im Paar zu speicherndes Element
       * @param e2
       *            Zweites im Paar zu speicherndes Element
       */
```

```
public Pair(A e1, B e2) {
      this.e1 = e1;
      this.e2 = e2;
}

/**
 * vergleicht die aktuelle Instanz mit dem &uuml;bergebenen
 * Paar.
 *
 * @param other
 *            Paar, mit dem verglichen werden soll.
 * @return ein Wert <0, 0, oder >0, wenn die Instanz kleiner,
 *         gleich bzw. größer als das übergebene Paar ist.
 * @throws NullPointerException
 *             wenn das Paar mit dem verglichen werden soll
 *             nciht initialisiert ist.
 * @see java.lang.Comparable#compareTo
 */
public int compareTo(Pair<A, B> other) {
      if (this == other)
            return 0;
      if (null == other)
            throw new NullPointerException();
      if (0 != e1.compareTo(other.e1)) {
            return e1.compareTo(other.e1);
      }
      return e2.compareTo(other.e2);
}

/**
 * zeigt an, ob ein anderes Objekt gleich diesem ist. Diese
 * Methode implementiert die &Auml;quivalenzrelation:
 * <ul>
 * <li>Sie ist reflexiv: F&uuml;r jedes Paar
 * <code>x&lt;A, B&gt;</code> erh&auml;lt
 * <code>x.equals(x)</code> den Wert <code>true</code>.
 * <li>Sie ist symmetrisch: F&uuml;r alle Paare
 * <code>x&lt;A, B&gt;</code>, <code>y&lt;A, B&gt;</code>
 * gilt, da&szlig; <code>x.equals(y)</code> genau dann
 * <code>true</code> zur&uuml;ckgibt, wenn
 * <code>y.equals(x)</code>  <code>true</code>
 * zur&uuml;ckgibt.
 * <li>Sie ist transitiv: F&uuml;r drei Paare
 * <code>x&lt;A, B&gt;</code>, <code>y&lt;A, B&gt;</code>
 * und <code>z&lt;A, B&gt;</code>, bei denen
 * <code>x.equals(y)</code> und <code>y.equals(z)</code>
 * beide <code>true</code> zur&uuml;ckgeben, gibt
 * <code>x.equals(z)</code> auch <code>true</code>
 * zur&uuml;ck.
 * <li>Sie ist konsistent: F&uuml;r zwei Paare <code>x&lt;A,
 * B&gt;</code>
 * und <code>y&lt;A, B&gt;</code> gibt
 * <code>x.equals(y)</code> immer das gleiche Ergebnis
 * zur&uuml;ck, solange sich in <code>x</code> und
 * <code>y</code> keine Daten &auml;ndern, die f&uuml;r die
 * <code>equals()</code>-Methode verwendet werden.
 * </ul>
 *
 * @param other
 *            das Objekt, mit dem verglichen werden soll.
 * @return <code>true</code>, falls dieses Objekt gleich
 *         dem Objekt im Argment ist, <code>false</code>
 *         sonst.
 * @see Object#equals(java.lang.Object)
 */
public boolean equals(Pair<A, B> other) {
      if (this == other)
            return true;
      if (null == other)
```

```
                return false;
            return (e1.equals(other.e1) && e2.equals(other.e2));
        }

        /**
         * gibt einen String zur&uuml;ck, der das Objekt
         * repr&auml;sentiert. Hier gibt er die beiden Elemente in
         * einem Klammerpaar, durch ein komma voneinander getrennt
         * zur&uuml;ck.
         *
         * @return einen String, der das Objekt repr&auml;sentiert.
         *
         * @see Object#toString()
         */
        public String toString() {
            return "(" + e1.toString() + ", " + e2.toString() + ")";
        }
}
```