In diesem Kapitel:

Programm-Dokumentation mit JavaDoc

Any member introducing a dog into the Society's premises shall be liable to a fine of £
Any animal leading a blind peson shall be deemded to be a cat..
- Rule 46, Oxford Union Society

Dokumentierende Kommentare werden in Java üblicherweise als *doc comments* bezeichnet. Sie dienen dazu, den Programmcode direkt zu dokumentieren. Der Inhalt dieser doc comments kann dann mit Hilfe eines Tools aus den Programmen extrahiert und formattiert werden, typischerweise mit Hilfe von HTML.

Doc Comments sind in der Regel knapp gehaltene Referenzdokumentationen. Die Referenzdokumentation beschreibt typischerweise den Kontrakt des dokumentierten Interfaces, der Klasse, des Konstruktors, der Methode, des Datenfeldes in einem Detaillierungsgrad, wie ihn der Programmierer benötigt um das entsprechende Konstrukt einsetzten zu können. Dieser Ansatz unterscheidet sich von einer Spezifikation, welche in der Regel umfangreicher und als separates Dokument vorhanden ist. Eine Spezifikation umfasst in der Regel mehrere Seiten Text für eine einzige Methode. Die Referenzdokumentation im Java Doc sollte im Wesentlichen ein oder zwei Paragraphen sein, eventuell noch kürzer. Die Spezifikation sollte also separat erzeugt und festgehalten werden. Aber es ist möglich, dass aus einem comment doc auf eine Spezifikation verwiesen wird.

Die Generierung der doc comments unterscheidet sich je nach Entwicklungsumgebung. Eine Möglichkeit, die Ihnen immer offen steht, ist es, das `javadoc` Hilfsprogramm einzusetzen, welches mit dem JDK mitgeliefert wird. Wegen dem Hilfsprogramm nennt man auch die generierte Dokumentation in der Regel *javadoc*, *JavaDoc* oder *Javadoc*.

Es ist wichtig, dass Sie wenigstens die gängigen Tags kennen, mit denen Sie die Programme dokumentieren können. Falls Sie JBuilder einsetzen, sehen Sie die generierte Javadoc gleich in der Entwicklungsumgebung. Sie müssen auch wissen, wie man den generierten Text interpretiert, wobei Sie darin sicher schon einige Erfahrung haben.

© J.M.Joller

1.1. Die Anatomie eines Doc Kommentars

Doc Kemmentare beginnen mit den drei Zeichen /** und enden beim nächsten */. Jeder doc comment beschreibt die Bezeichner, Methode,.... die gleich anschliessend folgt.

Beispiele:

```
/**
 * Title: Java Doc
 * Description: ein einfaches Beispiel für den Einsatz von JavaDoc
 * Copyright:    Copyright (c) 2000
 * Company:      abc
 * @author J.M.Joller
 * @version 1.0
 */
```

oder

```
/**
* Erledige, was das aufrufende Objekt erwartet. "Erwartet" wird gemäss dem
* ISO 7074-6 Standard definiert, auf Grund der Analyse des bisherigen
Verhaltens.
*/
public void dwin() throws IntentUnknownException;
```

Die * am Anfang der Zeile nach der Startzeile des Kommentars werden ignoriert, genauso wie Whitespaces (Tabs, ...) vor den Sternen.

Der erste Satz des Kommentars *(*`Erledige, was das aufrufende Objekt erwartet.`*)* fasst die Bedeutung das anschliessend beschriebenen Konzepts zusammen. Der erste Satz ist wie üblich der Text bis zum ersten Punkt. In praktischen Fällen sollten Sie also in diesem Bereich besonders vorsichtig dokumentieren und die Wortwahl sehr gut überlegen. Das obige Beispiel ist dafür kein gutes Beispiel!

Oft sind im Kommentar normale HTML Tags eingebettet. Damit kann man den Text besser formattieren, beispielsweise Tabellen erstellen, etwas hervorheben oder auf weitere Dokumentationen verweisen. Die Titel Tags (<h1>,....) sind reseviert für den generierten HTML Code, die braucht also javadoc intern bereits! An Stelle der Zeichen '<' und '>' sowie & sollten Sie jedoch die in HTML üblichen &lt und &gt sowie &amp verwenden. Sie können also nicht in HTML dokumentieren, weil der Kommentar erst durch javadoc formattiert wird.

Falls Sie in Ihrer Dokumentation ein '@' am Anfang einer Zeile haben müssen, muss dies als &#064 kodiert werden, sonst wird angenommen, dass es sich um einen Kommentartag handelt.

Es werden nur Kommentare analysiert, die direkt einer Klassendefinition, einem Interface, Methode oder Feld vorangestellt sind. Falls sich etwas anderes als Whitespaces oder Kommentare zwischen einem doc comment und dem, was dieser beschreibt, befindet, wird der Kommentar ignoriert!

© J.M.Joller

Beispiel: falls Sie einen Kommentar am Anfang der Datei schreiben und anschliessend eine Import oder eine Package Anweisung folgt, wird der Kommentar ignoriert.

Kommentare gelten immer pro Zeile. Falls Sie in einer Deklaration mehrere Felder definieren, gilt die Dokumentation für die gesamte Zeile. Sie sollten also bei der Verwendung von Kommentaren die Deklarationen sauber aufteilen.

Falls für geerbte Methoden keine Dokumentation erstellt wird, erbt diese Methode die Kommentare der Oberklasse. Sie sollten in den Fällen, bei denen Sie die Dokumentation aus der Oberklasse einfach übernehmen wollen in der verfeinerten Klasse einen Kommentar schreiben, aus dem hervorgeht, wo der Benutzer die Dokumentation findet und vorallem erwähnen, dass Sie die Dokumenattion nicht etwa vergessen haben.

Beispiel:

```
// Vererbung von Doc Kommentaren : siehe...
public void dwim() throws IntentUnknownException {
      //...
}
```

1.2. Tags

Doc Comments enthalten *Tags*, welche spezielle Informationen enthalten. Alle diese Tags beginnen mit einem @, wie beispielsweise @see oder @deprecated. Die Paragraphen mit diesen Tags werden speziell behandelt im generierten Dokument. Daraus resultieren spezielle Paragraphen in der Dokumentation, Verbindungen zu anderen Dokumenten oder weitere spezielle Anweisungen oder Hervorhebungen.

In diesem Abschnitt besprechen wir die meisten Tags, ausser @serial, @serialData und @serialField, welche mit der Objektserialisierung zusammenhängen.

Aller anderer Text (ausser den Tag Paragraphen) wird als Eingabetext für die HTML Generierung aufgefasst. Sie können diesen Text weiter formattieren, beispielsweise mit <p> für Paragraph, <pre> für vorformattierten Text und so weiter.

1.2.1. @see

Der @see Tag kreiert einen Querverweis auf eine andere Javadoc Dokumentation. Sie können irgend eine Grösse als Querverweis einsetzen, in der Regel wird es eine Klasse oder eine Methode einer Klasse sein, einfach mit deren Namen bezeichnet. Falls die Methode überladen ist, müssen Sie präziser adressieren, da Sie sonst eventuell zur Methode mit den falschen Parametern geführt werden. Sie können auch auf Interfaces oder Klassen im aktuellen Package verweisen. In diesem Fall mus der Link nicht voll qualifiziert sein.Falls Sie das Package wechseln, muss der Link voll qualifiziert sein.

Beispiel für @see:

- @see Attr
- @see com.magic.attr.Deck#DECK_SIZE
- @see Attribute Spezifkation

© J.M.Joller

Im ersten Falll wird auf ein Attr im selben Package verwiesen.
Im zweiten Fall wird auf eine Konstante in einem bestimmten Package verwiesen.
Im dritten Fall wird die Seite und der Anker auf den gesprungen werden soll, explizit aufgeführt.

1.2.2. {@link}

Der @see Tag wird typischerweise im Zusammenhang mit einem Hinweis am Ende der Dokumentation eingesetzt ("Siehe auch....").

Der Link Tag gehört zu einer Verbindung, die mitten im Text aufgeführt wird. Die Syntax des Tags ist

{ @link *package.class#member [label]*}

Der Unterschied zu @see ist wirklich nur der Ort bei dem Sie den einen oder anderen Tag einsetzen.

Beispiel für Link:

```
Ändert den Rückgabewert gemäss {@link #getValue}.
```

1.2.3. @param

Der @param Tag dokumentiert einen einzelnen Parameter einer Methode. Falls Sie @param einsetzen, sollten Sie pro Parameter eine Definition liefern, bzw. einen Link.

Syntax:

```
@param <Parametername> <Parameterbeschreibung>
```

Beispiel:

```
@param max Die maximale Anzahl Worte, die gelesen werden können.
```

1.2.4. @return

Dieser Tag dokumentiert den Rückgabewert eines Methodenaufrufes

Syntax:

```
@return <Beschreibung des Rückgabewerts>
```

Beispiel:

```
@return Die Anzahl Wörter, die gelesen wurden.
```


© J.M.Joller

1.2.5. @throws und @exception

Der @throws Tag dokumentiert eine Ausnahme, die von einer Methode geworfen wird. Falls Sie diesen Tag einsetzen, dann sollten Sie für jede Ausnahme, die geworfen werden kann einen Kommentar abgeben.

Der Tag @exception ist äquivalent zu @throws.

Syntax:

```
@throws <Name der Ausnahme> <Kurzkommentar>
```

Beispiel:

```
@throws java.io.IOException   Das Lesen aus dem Eingabestrom schlug fehl;
                              diese Ausnahme wird von Eingabestrom
                              weitergeleitet.

@throws UnknownName Der Name ist unbekannt.
```

1.2.6. @deprecated

Mit diesem Tag wird angezeigt, dass etwas veraltet ist, also unter Umständen in späteren Versionen nicht mehr unterstützt sein wird. Sie sollten aber in der Regel auch diese Programmteile so erhalten, dass alte Versionen des Programms problemlos weiterfunktionieren.

Der Tag ist also als Warnung gedacht, bei Neuentwicklungen Konstrukte einzusetzen, die unter Umständen durch bessere ersetzt wurden.

Syntax:

```
@deprecated <Hinweis>
```

Beispiel:

```
/**
* Führe das aus, was der Aufrufende erwartet. Die Erwartung wird in ...
* definiert.
* @deprecated Sie sollten besser dwishm einsetzen
* @see #dwish
*/
public void dwim() throws IntentUnknownException
```


© J.M.Joller

1.2.7. @author

Der @author Tag spezifiziert einen Autor des Programms.

Syntax:

```
@author <Name des Autors>
```

Beispiel:

```
@author Heinrich der Schreckliche
```

Sie können mehrere Autoren angeben, also mehrere @author Tags verweden.

1.2.8. @version

Dieser Tag dokumentiert die Version der Methode, Klasse oder was auch immer.

Syntax:

```
@version <Versionsnummer>
```

Beispiel:

```
@version 1.3
```

1.2.9. @since

Dieser Tag zeigt an. ab welcher Version diese Grösse zur Verfügung steht.

Syntax:

```
@since <Version>
```

Beispiel:

```
@since 1.2
```

Dieser Tag ist hilfreich beim Suchen von Änderungen im Programm oder in den Basisklassen.

© J.M.Joller

1.2.10. { @docRoot }

Die Dateien, welche von javadoc generiert werden, werden in einer Baumstruktur organisiert. Sie können dies leicht selber testen, indem Sie javadoc auf eine der Java Dateien anwenden, welche Kommentare enthält. Die genaue Anordnung der Dokumente im Dokumentenbaum ist allerdings nicht vorgegeben und hängt vom konkreten javadoc Tool ab. Es kann aber leicht passieren, dass Sie zu einem späteren Zeitpunkt weitere Äste in Ihren Baum eintragen möchten.

Der Tag { @docRoot } gestattet es Ihnen in Ihrem Dokument eine relative Position zu einem anderen Dokument anzugeben.

Syntax:

```
{ @docRoot } <Dokument [in der Regel eine HTML Seite]>
```

Beispiel:

```
Sie sollten den Kommentar in <a href="{@docRoot}/einfuehrung.html"> lesen
</a>
```

Es liegt aber an Ihnen, in diesem Fall dafür zu sorgen, dass im Root der generierten Dokumentation eine HTML Datei einfuehrung.html vorhanden ist.

© J.M.Joller

1.3. Beispiel

```
package javadocbeispiel;

/**
 * Title: Attribut als Namen/Wert Paar
 * Description:  Ein <code>Attr</code> Objekt definiert ein Attribut als
ein
 * Name / Wert Paar, wobei der Name ein <code>String</code>
 * und der Wert ein beliebiges <code>Object</code> ist.
 *
 * Copyright:    Copyright (c) 2000
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 2.1
 * @since 1.3
 */
public class Attr {
  /** Der Attributname */
  private final String name;
  /** Der Attributwert */
  private Object value=null;

  /**
   * Kreiert ein neues Attribut mit dem gegebenen Namen und einem initialen
   * Wert von <code>null</code>.
   * @see Attr#Attr(String, Object)
   */
   public Attr(String name) {
    this.name = name;
  }

  /**
   * Kreiert ein neues Attribut mit dem gegeben Namen und initalen Wert.
   * @see Attr#Attr(String)
   */
   public Attr(String name, Object value) {
    this.name = name;
    this.value = value;
  }

  /**
   * Liefert den Namen des Attributs.
   */
   public String getName() {
    return name;
  }

  /**
   * liefert den Wert des Attributs.
   */
   public Object getValue() {
    return value;
  }

  /**
   * Setzt den Wert dieses Attributs. Ändert den Wert, den Sie durch
Aufrufen
   * von {@link #getValue}.
```


© J.M.Joller

```
   * @param newValue  Der neue Wert für dieses Attribut.
   * @return Der ursprüngliche Wert.
   * @see #getValue()
   */
   public Object setValue(Object newValue) {
    Object oldVal = value;
    value = newValue;
    return oldVal;
  }

  /**
   * Liefert eine Zeichenkette in der Form <code>name=value</code>.
   */
   public String toString() {
    return name+"='"+value+"'";
  }
}
```

Die Ausgabe konnte nicht ins Word importiert werden! Die generierte Struktur war zu komplex für die automatische Umwandlung.

Sie finden das Ergebnis auf dem Server / der CD.

© J.M.Joller

1.4. Das javadoc Hilfsprogramm

Die Programmdokumentation ist in der Regel keine vollständige Dokumentation. Oft müssen Sie auf externe Dateien für eine vollständige Dokumentation verweisen oder umgekehrt aus einem Dokument auf die javadoc.

Das javadoc Hilfsprogramm verfügt über einige Optionen, die Sie setzen können, um detailliertere oder kompaktere Beschreibungen Ihres Packages zu erhalten.

Syntax:

```
javadoc [ options ] [ package | source.java ]*
```

1.4.1. Beschreibung

javadoc parsed, analysiert die Deklarationen und Dokumentationen in einem Java Programmcode Set und produziert eine HTML Dokumentation, in der public und protected Klassen, Interfaces, Konstruktoren Methoden und Felder beschrieben werden.
j**avadoc** akzeptiert entweder eine Serie von Java Packages oder Dateinamen.
javadoc generiert eine .html Datei für jede .java Datei und jedes Packages, welches gefunden wird.

Zusätzlich wird ein Klassenbaum tree.html aufgebaut und ein Index aller Klassen, die gefunden wurden, AllNames.html.

1.4.2. Optionen

-public

zeigt nur public Members

-protected

zeigt nur protected Members. Dies ist die Standardeinstellung.

-package

zeigt nur Packages

-private

zeigt alle Klassen und Members.

-version

das @version Tag soll auch ausgewertet werden (Standard: nicht)

-author

das @author Tag soll auch ausgewrtet werden (Standard: nicht)

© J.M.Joller

-noindex

generiere keinen Package Index

-notree

generiere keinen Klassenbaum

-d *directory*

schreibe die HTML Seiten in dieses Verzeichnis.

```
javadoc -d C:\opus\public_html\doc java.lang
```

-verbose

zeige an, was konkret passiert

© J.M.Joller

© J.M.Joller