3. Aufgabenblatt
(Dokumentationskonzept)

Inhaltsverzeichnis:

Dokumentationskonzept

JAVA-Doc

Die quelltextnahe Dokumentation muss sich zwingend an die JAVADOC-Konventionen halten. Das beinhaltet eine aussagekräftige Erläuterung der Klassenvariablen durch */** Beschreibung */* . Mittels JAVA-Doc lässt sich relativ automatische eine Quellcodedokumentation erstellen. Weiterhin gibt es sogar die Möglichkeit via JAVA-Doc große Teile der Benutzerdokumentation automatisch zu erstellen. Allerdings bedarf dies dann noch starken Anpassungen.

Eine Klassen oder Methodendokumentation hat folgendes Layout:

```
/**
 * TEXT (über Mehrere Zeilen möglicherweise)
 *
 * TAGS
*/
```

Um in JAVA-Doc bestimmte Variablen zu befüllen verwendet man den den Namen der Variable mit einem vorangestellten “@” (z.B. @param, @see). Erweiterungen von JAVA-Doc lassen auch noch weitere Variablen zu.

Da die JAVA-Doc standardmäßig eine HTML-Ausgabe erzeugt, ist es evident, dass man innerhalb der Quellcodedokumentation HTML-Tags verwenden kann.

Benutzerdokumentation

Die Benutzerdokumentation einer Applikation sollte vor allem in Zusammenarbeit mit den Personen geschrieben werden, welche die GUI entwickeln und bearbeiten. Dabei kann und sollte zum Teil auf die Quellcodedokumentationen zurückgegriffen werden. Das Hauptaugenmerk liegt hier zweifelsohne bei der hervorragenden Verständlichkeit der Dokumentation (im Gegensatz zur Quellcodedokumentation). Auch dürfen sog. Screenshots auf keinen Fall fehlen, ebenso wie konkret beschriebene Anwendungsfälle.

Das Junit Framework

Bei Junit handelt es sich um ein Framework, das bei der Erstellung von automatischen Softwaretest helfen soll und so zahlreiche Erleichterungen bietet.

Pro erstellter nicht-ausschließlichen Bean-Klasse sollte ein Testfall erstellt werden. Ein sog. Testcase. Dieser wird erzeugt, indem man die Klasse *TestCase* des von Junit erweitert. Anschließend werden die Methoden erstellt, die die eigentlichen Tests durchführen. Diese haben immer mit „test“ zu beginnen, da sie nur dann von Junit als Testmethoden anerkannt werden. Mit der „assertXXX“-Methode wird überprüft, ob die Methode den gewünschten Wert liefert. So wirft „assertEquals(object1, object2) eine Exception, wenn die beiden Objekte nicht übereinstimmen. Weiterhin muss bei der Verwendung von Junit mit sog. Mocks gearbeitet werden. Mocks sind Trojaner, die noch nicht fertig gestellte Klassen simulieren.

Tests lassen sich zu sog. Test-Suiten zusammenfassen. Damit kann man mehrere Testfälle nacheinander ausführen.

Für ein solides Testen des Quellcodes ist unabdingbar, zu wissen, wie viel Prozent des erstellten Codes bereits getestet wurden. Dazu verwendet man das Programm „Clover“ (http://www.cenqua.com/clover/). Die Abdeckung des Codes gibt natürlich keine Auskunft darüber, wie gut oder wie schlecht die erstellten Tests sind. Aber sie gibt einen Hinweis darauf, welche Klassen und Methoden noch getestet werden müssen.

Es besteht auch die Möglichkeit einer Test-getriebenen Entwicklung (*Test-Driven-Development* oder kurz *TDD*). Dabei wird der Test vor der Klasse geschrieben, die die Klasse benutzen soll. Dieses Konzept ist allerdings nicht immer anwendbar und birgt die Gefahr, dass man die Test vorrangig behandelt.

Java-Code Regeln

Es gibt eine Unzahl von Regeln, die den „Java-Knigge“ ausmachen. Die wichtigsten seien hier zusammengefasset.

Jeder vergebener Name (Packet, Klasse, Methode, Variable) muss beschreibend sein, für das, für was er steht. Klassennamen beginnen mit einem Großbuchstaben. Methoden und Variablennamen mit einem Kleinbuchstaben. Die vergebenen Namen sind auf Englisch zu schreiben.

In Schleifen sollten im Kopf (der Schleife) alle innerhalb der Schleife benötigten Zählvariablen festgelegt werden. (z.B. *for (int n = 0, i = array.length; n < i; n++)*)

Jede nichttriviale Codezeile bedarf eines Kommentars. Die Dokumentation erfolgt für alle erstellten Methoden und die darin vorkommenden und übergebenden Variablen. Wenn eine *Exception* geworfen werden kann, so ist zu beschreiben, wann dies geschehen kann. Noch offene Quellcode-Teile können mit „//TODO Beschreibung“, noch zu verbessernde können mit „//FIXME Beschreibung“ beschrieben werden. Eine Beschreibung der Klasse ist bei nicht-Bean-Klassen sinnvoll.

Jede nichttriviale Änderung des Codes bedarf der Anpassung und anschließenden Neuausführung des dazugehörigen Testes.

Verwendete Klassen aus externen Paketen sind im import-Bereich zu deklarieren. Die Klassen sollten nicht durch direkte Angabe des Pfades, sowie des Klassennamens aufgerufen werden. Es sollten keine ganzen Pakete via * importiert werden, sondern jede importierte Klasse einzeln.

Bei der Benennung von Klassen, die ein *Interface* implementieren ist der Name des implementierten *Interfaces* an den eigentlichen Klassennamen anzufügen (z.B. xyzActionListener). Bei Vererbung ist unter Umständen genauso zu verfahren (z.B. abcException).

Nach der Verwendung einer geschweiften, öffnenden Klammer ist ein der darauf folgende Quellcode um 4 Leerzeichen nach links zu verschieben. Umgekehrt gilt das Selbe nach rechts.

Nachdem und bevor man eine neue Variable in einer Methode anlegt ist eine Leerzeile einzufügen (am Anfang einer Methode kann die Leerzeile weggelassen werden.

Nach und vor Zuweisungsoperationen („=“, „+“, ...) ist ein Leerzeichen anzufügen.

Das verkürzte IF-THEN-ELSE-Schema („ x ? y : z“) ist nur bei kurzen, trivialen Ausdrücken zu verwenden. Eine Ausnahme in Bezug auf die Trivialität (nicht auf die Kürze) bilden *this(..)*-Aufrufe.

Vor Beginn des Eigentlichen Quellcodes kann optionalerweise eine eine Lizenzbestimmung hinterlegt werden. Falls eine Lizenz gewählt werden sollte, ist dies sogar stark zu empfehlen.

Konstruktoren erzeugen das komplette Objekt. Das bedeutet insbesondere den Verzicht auf sog. *Load*-Methoden.

Die im Abschnitt “JAVA-DOC” genannten Punkte sind einzuhalten. Es ist also also das komplette JAVA-Doc-Sprektrum abzugreifen. Einschränunk auf folgende Tags: param, author, version, date, throws. In der Dokumentation der Klasse ist “author”, “version” und “date” als Tags zu verwenden.

Die Funktion der Klasse sowie aller Methoden ist zu dokumentieren (Siehe “JAVA-Doc”)

Methodenlokale Variablen sind wie Klassenlokale Variablen zu kommentieren.

Diese Regeln gelten für alle zu erstellenden Klassen.