Styleguide

Version 1.0.0

Informatikprojekt WS 2010/2011

Freigabe am: 11.10.2010

Dokumenthistorie

Version	Datum	Status	Autor	Bemerkung
1.0.0	08.10.10	angelegt	extern	

Inhaltsverzeichnis

1 Vorbemerkung

Dieser Styleguide nutzt die Vorlage von Prof. Dr. Jörg Hettel und Prof. Dr. Thomas Allweyer in der Version 2.1 vom 29.09.2006.

Dieses Dokument beschreibt eine Reihe von Standards und Richtlinien zum Entwickeln von J2SE-Anwendungen im Rahmen des SWT-Projekts an der Fachhochschule Kaiserslautern am Standort Zweibrücken.

Diese sollen dafür sorgen, dass von den verschiedenen (Projekt-) Teams verfasster Code einheitlich und somit im Stil konsistent ist und bleibt.

Ein konsistenter Stil:

- erhöht die Lesbarkeit und damit die Wartbarkeit von Code
- erleichtert den Austausch von Code zwischen verschiedenen Programmierern
- spart Entwicklungszeit

Dieser Styleguide deckt nicht alle möglichen Situationen ab, ist jedoch auch für die Java-Entwicklung im Allgemeinen gültig.

2 Allgemeine Vorgaben

In diesem Kapitel werden auszugsweise die allgemeingültigen Richtlinien aus [RED00] wiedergegeben.

2.1 Quellcode-Dateien

Eine Java-Datei sollte nur eine öffentliche (public) Klassen- oder Interfacedefinitionenthalten, kann jedoch beliebig viele nichtöffentliche Support-Klassen, bzw. -Interfacesenthalten. Quelldateien sollten kleiner als 800 Zeilen gehalten werden, da größere Dateien schwierig zu warten sind. Die Quelldateien werden in einem CVS-Repository gehalten. Lesen Sie dazu bitte die entsprechende Dokumentation.

2.1.1 Namenskonventionen für Quell-Dateien

Die Dateien werden in englischer Sprache benannt. Dabei wird folgende Form verwendet:

```
ClassOrInterfaceName.java
```

Bei der Implementierung von Patterns muss der Patternname in den Klassennamenintegriert werden. Bsp:

```
DSAccesSingelton.java
```

2.1.2 Aufbau der Quell-Dateien

Eine Java-Quelle sollte folgende Elemente in der angegebenen Reihenfolge enthalten:

1. Copyright/Autor/Version/History Block-Kommentar
2. Package-Deklaration
3. Import-Deklarationen
4. eine oder mehrere Klassen/Interface-Deklarationen

2.1.3 Copyright/ID Block-Kommentar

Jede Quellcode-Datei sollte mit einem Block-Kommentar beginnen, der Versionsinformationen und eine Copyright-Bemerkung enthält.

Die Versionsinformationen sollten folgendes Format haben:

```
Dateiname Version Datum
```

Dies lässt sich durch Nutzung von CVS mit Hilfe von “keyword substitution” automatisieren. Der komplette Kommentar-Block sieht dann wie folgt aus (Bsp.):

```
/*
 * $Id$
 *
 * Copyright (c) 2006 Fachhochschule Kaiserslautern
 *                    Standort Zweibruecken
 * All Rights Reserved
 *
 * Code-Verantwortlicher: ...
 */
```

Wichtig: Im Projekt gibt es für jede Quellcode-Datei ***einen*** **Code-Verantwortlichen**, der als Autor namentlich genannt werden muss!

2.1.4 Package Deklaration und weitere Verzeichnisse

Die Java-Paketstruktur sieht wie folgt aus:

```
de.fhzw.informatik.swtprojekt.ws0607.gruppenname.<Anwendung>
```

Unter der Anwendung gibt es dann verschiedene Subpackages:

```
.db         // Alle DB-Zugriffsklassen
.view       // View-relevante Klassen (Swing-Komponenten)
.model      // Modellklassen (Anwendungsschicht)
.core       // Von verschiedenen Klassen benötigte Klassen bzw.
               Interfaces
.util       // Hilfklassen
.test       // JUnit Testklassen
```

Paketnamen sind in Englisch zu vergeben. Weitere Packages können bei Bedarf angelegtwerden.

2.1.5 Import Deklarationen

Import-Deklarationen sollten voll qualifiziert erfolgen wegen:

- möglichen Namenskonflikten
- der besseren Lesbarkeit (welche Klassen/Interfaces werden genutzt?)

Beispiel:

```
java.lang.*      //schlecht
java.lang.String // o.k.
```

Tipp: Dieses Feature kann meist im Entwicklungstool eingestellt werden.

2.1.6 Klassen-/Interface-Deklarationen

Eine Datei sollte nur eine Klassen-/Interface-(=Typ-)Deklarationen enthalten. Sie sollte nur eine öffentliche Typ-Deklaration enthalten, der ein JavaDoc-Kommentar vorausgehen sollte, das die Funktion und Parameter (@param Tag) beschreibt. Nichtöffentlichen Typen sollte ein normaler Kommentar zur Beschreibung vorangestellt werden.

2.2 Namenskonventionen

2.2.7 Sprachen

Jeglicher Programmcode (auch in Konfigurationsdateien) ist in Englisch zu halten. Kommentare, Dokumentationen und Exceptions der Geschäftslogik sind Deutsch zu halten. Umlaute sind, außer in auszugebenden Strings, durch ae, oe, ue zu ersetzen, ß durch ss.

2.2.8 Klassennamen

Jeweils der erste Buchstabe jedes enthaltenen Worts wird in Großbuchstaben gehalten. Sind mehrere Worte im Namen enthalten, werden diese direkt aneinander geschrieben.

Werden Design Pattern benutzt ist der jeweilige Pattern-Name im Klassen-/Interfacenamen zu verwenden. (z.B. DBFacade)

2.2.9 Benennung von Feldern

Variablennamen werden klein geschrieben. Ausnahme sind Namen, die aus mehreren Worten bestehen. Bei diesen beginnen die Folgeworte mit einem großen Buchstaben.

Beispiele:

```
boolean resizable;
char recordDelimiter;
```

Konstanten (static final) werden komplett in Großbuchstaben geschrieben.

2.2.10 Methoden-Benennung

Methoden-Namen beginnen mit Kleinbuchstaben gefolgt von einem Paar runden Klammern.

Die Namen sollten die Funktion der Methode beschreiben.

Beispiel:

```
showStatus()
```

3 Kommentare und Dokumenation

Java unterstützt drei Arten von Kommentaren: Dokumentationskommentare, Blockkommentare und einzeilige Kommentare. Diese werden in den nächsten Abschnitten beschrieben.

Hier einige allgemeine Richtlinien zur Nutzung von Kommentaren:

- Kommentare sollen dem Leser helfen die Funktion des Codes zu verstehen. Sie sollen den Leser durch den Programmfluss führen, speziell an Stellen die unklar sein könnten.
- Vermeiden Sie unnötige Kommentare, die offensichtlich sind:
  `i = i + 1;    //erhöhe i um 1`
- Irreführende Kommentare sind schlimmer als gar kein Kommentar
- Vermeiden Sie es Informationen in einem Kommentar aufzunehmen, die veralten könnten.
- Keine “kunstvollen” Kommentarboxen
- Temporäre Kommentare sollten mit “XXX:” markiert werden, um sie später leichter wieder finden zu können.

3.1 Dokumentationskommentare (JavaDoc)

Sollen hier nur anhand eines Beispiels verdeutlicht werden. Wichtig ist, dass diese

Kommentare mit /** beginnen und mit **/ enden.

Beispiel:

```
/**
 * Hier steht ein beschreibender Text
 *
 * @param bar Beschreibung des Parameters
 * @return Beschreibung des Rückgabewertes
 * @exception name Beschreibung der Exception
 * @see foobar
 **/
public boolean foo(String bar) throws Exception {
```

3.2 Blockkommentare

Blockkommentare werden für den Copyright/ID-Kommentar und zum Auskommentieren von Code benutzt. Sie beginnen mit /* und enden mit */.

3.3 Einzeilige Kommentare

Ein einzeiliger Kommentar besteht aus // und dem Kommentartext. Dazwischen ist ein Leerzeichen. Der Kommentar muss die gleiche Einrückung habe wie der ihm folgende Code. Wenn der Kommentar zu mehreren folgenden Statements gehört, folgt auf das letzte Statement eine Leerzeile.

Ein Kommentar kann auch hinter einer Anweisung stehen. Dann sollte zwischen Anweisung und Kommentar mindestens ein Leerzeichen sein. Handelt es sich bei der kommentierten Anweisung um umgebrochene Zeilen, so ist im Anschluss eine Leerzeile einzufügen.

4 Klassen

Eine Klassen-Deklaration sieht wie folgt aus:

```
[ClassModifiers] class ClassName [Inheritances]
{
 ClassBody
}
```

ClassModifiers sind Kombinationen folgender Schlüsselworte in der angegebenen Reihenfolge:

```
public abstract final
```

Inheritances sind Kombinationen folgender Phrasen in der angegebenen Reihenfolge:

```
extends SuperClass
implements Interfaces
```

4.1 Aufbau des Klassenkörpers

Der Klassenkörper ist wie folgt aufgebaut:

1. static-Variablen-Deklarationen
2. Instanz-Variablen-Deklarationen
3. static initializer
4. static member inner class Deklarationen
5. static-Methoden-Deklarationen
6. instance initializer
7. instance constructor Deklarationen
8. instance member inner class Deklarationen
9. instance method Deklarationen

4.2 Methoden Deklarationen

```
[MethodModifiers] Type MethodName(Parameters)
                        [throws Exceptions] {
```

MethodModifiers:

```
public protected private abstract static final
synchronized native
```

Der komplette Code einer Methode muss auf eine Bildschirmseite (Standardauflösung: 1024x768, mit lesbarer Schriftgröße) passen.

4.3 Statements

- Pro Zeile nur eine Anweisung
- pro Zeile nur eine lokale Variablendeklaration
- eckige Klammern bei Array-Deklarationen hinter dem Namen des Arrays
- Rückgabewerte in der return-Anweisung nicht in Klammern, ausser es handelt sich um einen komplexen Ausdruck

Weiteres in [RED00].

4.4 Attributzugriffe

4.4.1 Instance-Member-Zugriffe

Memberzugriffe erfolgen immer mit this.

```
this.member();
```

4.4.2 Class-Member-Zugriffe

Statische Zugriffe erfolgen immer mit dem Klassennamen.

```
UsedClass.test();
```

5 DB-Zugriffe

Zum Zugriff auf die Datenbank werden in eigenen DB-Klassen im projektspezifischen .db Paket gekapselt. Die Klassennamen all dieser Klassen enden auf DB und lehnen sich an den Tabellennamen der jeweiligen Master-Tabelle an. Für die Tabelle Demotab also zum Beispiel: `DemotabDB .`

6 Allgemeine Richtlinien

Wird Code auskommentiert muss, muss der Grund durch ersichtlich sind. Wird Code nicht mehr benötigt, so ist er in der Regel zu entfernen (Vermeidung von dead-Code). Ebenfalls zu vermeiden ist die Duplizierung von Code. Code-Duplizierung lässt sich in der Regel durch geeignete Designs vermeiden. Benutzerdefinierte Vererbungshierarchien sollten nicht tiefer als drei Ebenen sein.

Vererbungshierarchien müssen im Code dokumentiert sein.

Law-of-Demeter (LoD): Reichweite nur zum Nachbarpaket. Also keine Aufrufe der Art:

```
DBClassInstance.getModelInstance().getCoreInstance().doSomething()
```

7 Literaturverzeichnis

[RED00]: Reedy, A.: Java Coding Style Guide, 2000,
http://developers.sun.com/prodtech/cc/products/archive/whitepapers/java-style.pdf

JavaDoc: http://java.sun.com/j2se/javadoc/

geprüft durch:

Name	Datum	Unterschrift

freigegeben durch:

Name	Datum	Unterschrift