Programmierrichtlinien

Vorwort:
Diese Richtlinien gelten für den Anteil des in Java implementierten Quellcodes während des Projekts „FreeSpace“. Es soll den Entwicklern eine leichteres Verständnis gegenüber dem Code anderer Teammitglieder ermöglichen um damit der Qualitätssicherung entgegenkommen.
Der Inhalt orientiert sich zum größten Teil an den Richtlinien des von den Entwicklern bekannten „Objektorientierte Softwareentwicklung“ Praktikums von Herrn Prof. Dr. Faßbender der FH Aachen.

1. **Allgemeine Richtlinien**:
   1. Programme müssen lesbar bleiben.

2. **Kommentierung**:
   1. Jede Datei mit Quellcode muss kommentiert werden
   2. pro Datei einige einleitende Zeilen, die die Funktionalität des Quelltextes beschreiben
   3. vor jeder Klassendefinition Kommentar zur Funktionalität der Klasse
   4. wenn aus einem Attributnamen nicht hervorgeht, was darin abgespeichert wird, muss dazu ein Kommentar in der Zeile vor dem Attribut verwendet werden
   5. vor jeder Methodenimplementierung Kommentar zur Funktionalität
   6. die Verwendung von javadoc ist zwingend vorgeschrieben für Public

3. **Formatierung**:
   1. genügend Freiraum für die Programmzeilen schaffen, die den Code deutlich hervorheben
   2. gewöhnliche Leerzeichen anstelle von Tabulatoren verwenden, da unterschiedliche Editoren Tabulatoren unterschiedlich interpretieren.
   3. Die Zeilenlänge sollte 78 Zeichen nicht überschreiten.
   4. Die Rümpfe von Blöcken, Klassen, Methoden und Schleifen werden jeweils um 2 Spalten eingerückt
   5. Die öffnende Klammer steht in der übernächsten Spalte hinter dem Methodennamen bzw. der Schleifenbedingung
   6. Die schließende Klammer steht in der Zeile unmittelbar hinter dem letzten Befehl des Blockes und 2 Spalten links der ersten Spalte des Blockes.

      Bsp.:
      ```
      class EinKlassenName { …
              for (int i = 0; i < 10; i++) {
                System.out.println("i= " + i);
              }
            }
      ```

   7. Jede Anweisung in extra Zeile.
   8. Bei Kontrollflussbefehlen if, else, for, while immer einen Block verwenden, auch wenn der Block nur aus einer oder keiner Anweisung besteht, da dadurch spätere Umprogrammierung nicht so schnell zu Fehlern führt.

      Bsp.:
      ```
      while ( Bedingung ) {
        // Empty !
      }
      ```

9. else immer unterhalb des zugehörigen if
10. Parameter bei Methoden in eine Zeile, wenn es passt, sonst untereinander jeder in eine extra Zeile. Schließende Klammer hinter letzten Parameter

Bsp.:
```
int plus(int summand1, summand2) {
  ....
}
AdjustmentListener remove(AdjustmentListener l,
                          AdjustmentListener oldl) {
  ...
}
```

11. Der Punktoperator für den Zugriff auf Objekt- oder Klassenvariablen bzw. -methoden, darf nicht durch Freiraum von dem Klassenname bzw. dem Ausdruck für die Objektreferenz oder dem Methoden- bzw. Variablennamen abgetrennt werden.
Bsp.: persons.whoAmI.myName
- try_catch-Klauseln sind wie folgt zu formatieren:

```
try {
  Anweisung;
  ...
} catch (Ausnahmetyp1 x) {
  Anweisung;
  ...
}
...
} catch (Ausnahmetypn x) {
  Anweisung;
  ...
}
```

4. **Bezeichnungen:**
   1. **Klassenname:**
      Klassennamen werden in UpperCamelCase geschrieben
      Beispiel: Character oder ArrayList
   2. **Methodennamen:**
      1. Methoden werden in lowerCamelCase geschrieben. Meistens sind deren Bezeichnungen Verben
         Beispiel: sendMessage, stop
      2. Unterstriche werden häufig bei Unittest Methoden verwendet um Stati zu unterscheiden. Pattern: test<Methodenname>_<status>
         Beispiel: testAdd_emptyQueue
   3. **Konstanten:**
      Konstanten werden in Großbuchstaben geschrieben
      Beispiel: static final string KLASSENNAME = „MeineKlasse“
   4. **Parameternamen**:
      1. Parameter werden in lowerCamelCase geschrieben

5. **Ausdrücke**:
   1. Shift-Operatoren als Ersatz für arithmetische Operationen sind zu vermeiden.
   2. Möglichst Klammern setzen.
   3. Nur eine Zuweisung pro Zeile, also nicht:

      Bsp.:
      ```
      a = (b = c + d) + e;
      ```
      stattdessen:
      ```
      b = c + d;
      a = b + e;
      ```

   4. Zahlenliterale sind in Ausdrücken zu vermeiden; stattdessen Konstante mit final definieren.

      Bsp.:
      ```
      final int ZEHN = 10;
      ```

6. **Anweisungen**:
   1. Zur Formatierung siehe Abschnitt 3
   2. Verwendung von for-Schleife immer dann, wenn eine Variable um eine konstante Größe erhöht (vermindert) wird und wenn vor Ausführung feststeht, wie oft sie durchlaufen wird.
   3. Verwendung von while-Schleife, wenn eine solche Variable nicht vorliegt.
   4. Verwendung von do/while-Schleife, wenn Abbruchbedingung erst nach einem Schleifendurchlauf ausgewertet werden kann.
   5. möglichst Verzicht auf continue.
   6. break immer dort einsetzen, wo benötigt.
   7. bei switch/case aufpassen, dass man breaks verwendet.

7. **Klassen**:
   1. Gruppenweise in der Reihenfolge ihrer Sichtbarkeit programmieren:
      1. public
      2. protected
      3. paketsichtbar
      4. private
   2. Innerhalb einer Gruppe folgende Reihenfolge:
      1. alle Attribute
      2. alle Konstruktoren
      3. alle Methoden

8. **Zugriffsrechte**:
   1. Zugriff auf public- und protected-Variablen vermeiden, da dies zu Schwierigkeiten führen kann.
   2. möglichst nur lesend auf Variablen zugreifen
   3. setter und getter einführen

9. **Codelänge und switches**:
   1. Programmcode von Methoden sollte maximal 2 Seiten lang sein.
   2. lieber switch mit sehr vielen Fällen als massenweise geschachtelte ifs.

10. **Javadocs**:
    1. **Kopf einer Klasse**

       Im Kopf der Klasse wird die Funktionalität und die grobe Struktur der Klasse

beschrieben. Außerdem werden die Standardeinträge für Copyright, Organisation, Autor und Version ausgefüllt.
Beispiel:

```
/**
 * <p>Überschrift: Struktur von Benutzern </p>
 * <p>Beschreibung: Diese Klasse definiert die grundlegende Struktur von
 *                  Benutzern.
 *                  Die Struktur eines Benutzers setzt sich zusammen aus:
 *                  - der UserId und
 *                  - dem Passwort
 *                  Da es nur um eine Struktur geht, werden lediglich die
 *                  beiden Standardmethoden equals und toString
 *                  implementiert.</p>
 *
 * <p>Copyright: Heinz Faßbender Copyright (c) 2003</p>
 * <p>Organisation: FH Aachen, FB05 </p>
 * @author Heinz Faßbender
 * @version 1.0
 */

public class Benutzer { ...
```

2. **Bedeutung der public-Attribute**:
   Vor jedem public-Attribut wird die Bedeutung des Attributs beschrieben, wenn der Name nicht selbsterklärend ist.
   Beispiel:

```
/**
 *  Attribut zur Speicherung der UserID:
 */
  public  String userId;
```

3. **Bedeutung der public-Konstruktoren**:
   Die Konstruktoren müssen nur kommentiert werden, wenn sie außer der Initialisierung noch andere Funktionalität liefern.
   Beispiel:

```
 /**
  *  liefert noch Ausgabe, dass er in Standardkonstruktor ist
  */
   public Benutzer() {
     System.out.println("Bin in Standardkonstruktor!");
   }
```

4. **Bedeutung der public-Methoden**:
   Die Kommentierung der public-Methoden ist besonders wichtig, da diese die Dienste des Objekts liefern und somit die Schnittstelle implementieren. Deshalb wird hier besonderer Wert draufgelegt. Außerdem ist die Bedeutung der Parameter jeweils mit dem Tag @param zu beschreiben.
   Beispiel:

```
/**
 *  Standardmethode
 *  @param benutzer liefert das Objekt, dessen Inhalte mit denen des
 *                  aktuellen Objekts verglichen werden sollen.
 */
  public boolean equals(Benutzer benutzer) {
    return (this.userId.equals(benutzer.userId) &&
            this.passWort.equals(benutzer.passWort));
  }

/**
 *   Standardmethode, die die Inhalte der beiden Attribute in der folgenden
 *   Form ausgibt: userId/passWort
*/

  public String toString() {
    return (userId + "/" + passWort);
  }
```