Java

Programmierkonventionen

Autor:
Rudolf Histel

Erstellt / Aktualisiert: 23.12.2021

Einleitung

Dieses Dokument beschreibt Programmierprinzipien der Softwareentwicklung und die zusammen gefassten Programmierrichtlinien für die Programmiersprache Java.

Diese basieren auf den offiziellen
„Code Conventions for the Java TM Programming Language Revised April 20, 1999“:

- https://www.oracle.com/java/technologies/javase/codeconventions-contents.html
- https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Ich möchte hier noch einmal ganz klar betonen, dass dies meine persönlichen Vorgaben und Adaptionen sind die in meinen Kursen und Lehrveranstaltungen und Lehrmaterialien genutzt werden.

Diese setzt sich aus meiner Praxiserfahrungen als Senior Java Entwickler und verschiedener Literaturen zusammen. Sehr ausschlaggebend für meine angestrebte Philosophie ist das Buch „Clean Code“ von Robert C. Martin. Dieses gilt in der Softwarentwicklungsbranche mittlerweile als unausgesprochener Standard, wenn es um die Programmierung geht.

Andere Unternehmen und/oder Entwickler haben vielleicht andere Vorgaben an die es sich dann zu halten gilt sollte man in deren Team arbeiten.

Es besteht immer ein Unterschied zwischen Theorie und Praxis. Ein Unterschied zwischen der Philosophie und der Realität wie Sie tatsächlich gelebt wird.

Für jede Regel, jeden Prozess gibt es einen Grund in allen fast allen Fällen auch erfragen kann.

Dieses Dokument hat zum Ziel jedem einen leichten und kompakten Einstieg in die Programmierung zu bieten.

Allgemeine Regeln und Prinzipien der Softwareentwicklung

Namensgebung

Jegliche Namensgebung sollte in aller erster Line, singend, klingend also aussagekräftig sein. Der Name der zur Benennung von etwas genutzt wird sollte immer so gewählt werden das er direkt aussagt was das Ding tut. Generische Namen wie zum Beispiel „button1“ sind in der Regel nicht aussagekräftig und sollte daher nicht verwendet werden.

ASCII-Tabelle – Keine Umlaute oder Sonderzeichen

Egal mit welcher Programmiersprache man programmiert, man bewegt sich im englischen Sprachraum. Der Grundzeichensatz den jeder Computer und jede Software versteht ist der „American Standard Code for Information Interchange“ kurz ASCII genannt. Die ASCII-Tabelle der ist der Grundzeichensatz aller modernen Zeichensätze, enthält jedoch nur Buchstaben und Zeichen die im englischen Sprachraum vorhanden sind. Im heutigen Zeitalter gibt es natürlich fortgeschrittenere Zeichensätze (z.B. UTF-8) die mittlerweile Flächendeckend eingesetzt werden. Dieses ermöglichen es zwar Umlaute und nicht englisch native Zeichen anzuzeigen, aber oftmals werden Informationen und Projekte in andere Systeme übertragen um hier Interpretierungs- / Parsingproblemen vorzubeugen sollte man sich bei jeglicher Namensgebung auf die Zeichen, Symbole und Buchstaben aus der ASCII-Tabelle begrenzen. Diese kann jedes System interpretieren und abhängig davon welchen genauen vielleicht auch lokalisierten oder angepassten Zeichensatz es nutzen mag.

Kurz: Keine Umlaute und oder Sonderzeichen verwenden, maximal ein Unterstrich.

Das KISS – Prinzip

Kurz: „Keep it super simple“. So einfach wie möglich und nur so komplex wie unbedingt nötig.

Das DRY – Prinzip

„Don‘t repeat your self!“ zu Deutsch „Wiederhole dich nicht“. Dient dazu Dubletten zu verhindern in dem man Codeteile die man öfters schreiben würde in einer Methode, Funktion, Klasse, oder passendes Konstrukt zentralisiert wird. Das erlaubt für eine modulare Pflegbare Nutzung.

Kurz: Vermeide Codedubletten und lege Wert auf zentralisierte Wiederverwendbarkeit

Single-Responsibility Prinzip / Separation of Concerns

Diese zwei Philosophien sind eigentlich synonym zu behandeln. Der Gedanke dabei ist, dass jeder Programmteil egal ob Methode, Funktion, Klasse, Interface, Record, Enum oder Datei einen bestimmten Zweck erfüllt und auch nur diese eine spezielle Aufgabe übernimmt.

Hier ein paar Beispiele:

- Eine Klasse „Hausmodel“ die ein Haus beschreibt, sollte nicht die Finanzbuchhaltung übernehmen.
- Die Personalverwaltung ist für das Ausstellen der Lohnzettel und das überweisen des Gehaltes zuständig, jedoch nicht für die Rechnungsverbuchung des Einkaufs.
- Eine Funktion die Rechnungen ausdruckt, kann gerne unterscheiden ob die Rechnung als PDF generiert wird oder an einen bestimmten Drucker gesendet wird, aber würde nicht die Produkte der Rechnung umbenennen oder mit einem Rabatt versehen.

Dieses Vorgehen verhindert das Erstellen von so genannten „Gott-Funktionalitäten“. Also „Die eierlegende Wollmilchsau auf Steroiden“. Ein Programmteil, eine Methode, eine Funktion die mehr macht als Sie muss oder gar versucht alle Aufgaben der Applikation zu übernehmen.

Kurz: Jeder Programmteil hat eine einzige Existenzberechtigung dieses darf nicht verletzt werden, in dem dieser Teil zu viel oder zu wenig der Aufgabe übernimmt.

Variablennamen

- Variablen beginnen immer mit einem Kleinbuchstaben.
- Variablen dürfen nicht nur aus ein Buchstaben bestehen. Ausgenommen sind Zählvariablen, wie zum Beispiel: „i“, „j“, „k“ oder „u“ … .
- Variablen sind direkt mit einem Wert zu initialisieren der im weiteren Programmablauf unwahrscheinlich ist.
- Es ist kamelSchreibweise ist zu verwenden, sprich jedes neue Wort innerhalb des Variablennamens beginnt mit einem Großbuchstaben.
- Keine Zahlen und Keine Unterstriche verwenden.
- Keine Sprachen miteinander vermischen.

Beispiele:

```
String productName           = "Rosen";
double productPrice          = 4.99D;
int    productAmount         = 5;
long   soldProductsWorldWide = 3000000000L;
```

NoGo’s:

```
String p1                    = "Rosen";
double p_PREIS_1             = 4.99D;
int    productAnzahl         = 5;
long   SoldProductsWorldWide = 3000000000L;
```

Konstanten

- Konstanten werden grundsätzlich komplett in Großbuchstaben geschrieben.
- Konstanten werden immer unter direkt unter dem Klassennamen definiert.
- Wie immer sind singende und klingende Namen zu verwenden.
- Keine Zahlen verwenden
- Einzelne Wörter sind mit einem Unterstriche (_) zu trennen.

Beispiel:

```
private static final String PRODUCT_NAME_LABEL_TEXT = "Produktname:\t";
```

Klassen

- Klassennamen müssen mit einem Großbuchstaben beginnen.
- Es sind sprechende Namen zu wählen
- Keine Unterstriche, Zahlen und Sonderzeichen

```java
/**
 * Jede Klasse muss ein kurzes, knackiges Beschreibungskommentar haben.
 */
public class ExampleClass {

   //region 0. Konstanten
   private static final String DEFAULT_VALUE_FOR_STRINGS  = ">noValueYet<";
   private static final int    DEFAULT_VALUE_FOR_INTEGERS = -1;
   //endregion

   //region 1. Decl and Init Attribute
   private String stringAttribute;
   private int    integerAttribute;
   //endregion

   //region 2. Konstruktoren

   /**
    * Standard Konstruktor
    * zum direkten initalisieren
    * der Attribute
    */
   public ExampleClass() {
      this.stringAttribute = DEFAULT_VALUE_FOR_STRINGS;
      this.integerAttribute = DEFAULT_VALUE_FOR_INTEGERS;
   }

   /**
    * Ueberladener Konstruktor zum direkten setzen der Attriubute
    * bei der Instanziierung
    *
    * @param stringAttribute  : {@link String} : Ein Text
    * @param integerAttribute : int : Eine Ganzzahl
    */
   public ExampleClass(String stringAttribute, int integerAttribute) {
      this.stringAttribute = stringAttribute;
      this.integerAttribute = integerAttribute;
   }

   //endregion

   //region 3. Getter und Setter

   public String getStringAttribute() {
      return stringAttribute;
   }

   public void setStringAttribute(String stringAttribute) {
      this.stringAttribute = stringAttribute;
   }

   public int getIntegerAttribute() {
      return integerAttribute;
   }

   public void setIntegerAttribute(int integerAttribute) {
      this.integerAttribute = integerAttribute;
   }

   //endregion

   //region 4. Methoden und Funktionen
   @Override
   public String toString() {
      return "ExampleClass{" +
            "stringAttribute='" + stringAttribute + '\'' +
            ", integerAttribute=" + integerAttribute +
            '}';
   }
   //endregion
}
```

Interfaces

- Interface müssen mit einem Großbuchstaben beginnen.
- Es sind sprechende Namen zu wählen
- Keine Unterstriche, Zahlen und Sonderzeichen

```java
/**
 * Kurzbeschreibung
 */
public interface ExampleInterface {

    //region 0. Konstanten
    String STRING_KONSTANTE = "Ein Text";
    //endregion

    //region 1. Methoden und Funktionen

    /**
     * Eine Methode die Dinge tut.
     * @param integerValue : int : Ganzzahl
     */
    void methodWithParameters(int integerValue);

    /**
     * Eine Funktion die Dinge tut.
     * @return double : Eine Kommazahl
     */
    double functionWithNoParameters();
    //endregion
}
```

Methoden und Funktionen

- Sprechende Namen verwenden
- Methoden und Funktionsnamen müssen mit einem Kleinbuchstaben beginnen
- Keine Umlaute und oder Sonderzeichen sowie Unterstriche.

Beispiel:

```java
/**
 * Methode keine Rueckgabetyp und kein Rueckabewert
 */
private void method() {
    System.out.println("Hello World");
}

/**
 * Funktion Rueckgabtyp ist int
 * @return integerValue : Rueckgabewert
 */
private int function() {
    System.out.println("Hello Java");
    return -1;
}
```

Syntax

- Immer wieder den Quellcode automatisch formatieren lassen.
- Benutzt Zeilenumbrüche und Leerzeilen, wir schreiben Quellcode und kein Aufsatz im Blocksatz
- Haltet Verzweigungen und Schleifen einfach und gut leserlich
- Verwendet in einer Methode oder Funktion nicht mehr als vier Parameter
- So einfach wie möglich und nur so komplex wie möglich
- Jede Klasse bekommt ein kurzer, beschreibender und aussagekräftiger Kommentar.
- Wenn statisch auf einzelne Arrayelemente zugegriffen werden muss, so sind Konstanten / finale Variable für den Index zu verwenden: personalakten[INDEX_AKTE_HANS] = „Wert“;
- Es sind keine Kommentare zur Trennung der des Quellcodes zu verwenden: //## -> das tun wir nicht
- Kontrolliert stets den Programmablauf und den des Prozesses, wo kommen Objekte her, wo gehen Sie hin und was machen wir damit.
- Arbeitet in kleinen Iterationen, Step by Step, und testet kontinuierlich euren Code.