Code-Style-Guideline

-

Klärung jeglicher Fragen in Bezug auf qualitativen Code

Version 1.5

Inhalt

0.1 Ziel des Dokumentes

In diesem Dokument wird genau beschrieben, wie Code in unserem Projekt auszusehen hat. Ebenfalls ist in dieser Guideline auch enthalten, wie vernünftige Kommentare und JavaDoc auszusehen haben. Angelehnt ist die Guideline an die Google Java Style Guide.

0.2 Changelog

Version	Datum	Änderung	Geändert von
1.0	4.11.2018	Initial	Lukas Boschanski
1.1	6.11.2018	Anpassung	Lukas Boschanski
1.2	6.11.2018	Erweiterung Namengebung	Lukas Boschanski
1.3	7.11.2018	Festlegung der Formatierung	Lukas Boschanski
1.4	8.11.2018	Finalisierung	Lukas Boschanski
1.5	10.11.2018	Formatierung	Steffen Sassalla
1.6	17.11.2018	Kapitel 1,9 Änderung und Kapitel „Lambdas	Lukas Boschanski

		und anonyme innere Klassen“ neu	
1.7	05.01.2019	Kommentierung von Klassen	Tim Dahm
1.8	21.01.2019	Anpassung	Lukas Boschanski

1. Dateien

- Benennung
  Generell ein case-sensitiver einzigartiger Name mit “.java” als Dateiendung. Falls es sich um eine Gui-Beschreibung in der Metasprache FXML handelt, so ist ebenfalls ein einzigartiger Name in case-sensitive zu verwenden mit der Dateiendung “.fxml“.
- Datei Encoding
  Standardcodierung ist UTF-8
- Spezielle Zeichen
  Tab und Spaces sind erlaubt, werden aber von unserem Formatierungstool dementsprechend gesetzt.

  Für spezielle Zeichen, die standardmäßig in Java sind (special escape sequences), benutzen wir diese Darstellung anstatt des Unicodes/Oktal-Darstellung. (z.B.: \n)

  Falls ein nicht-ASCII Zeichen dargestellt werden muss, verwenden wir die Unicode Version (z.B.: µ).

2. Dateistruktur

- Struktur jeder Datei
  Jeweils mit einem Zeilenumbruch getrennt in dieser Reihenfolge:
  - Lizenzangaben, falls nötig
  - Package Statement
  - Import Statement
  - Genau eine Top-Level Klasse

- Überladene Konstruktoren/Methoden
  Stehen ungetrennt hintereinander ohne Fremdcode dazwischen.

- Imports
  Wir benutzen, falls ein Import nötig ist, diesen **ohne** das Schlüsselwort static. Ausgenommen hiervon sind Unit-Test die mit JUnit erfolgen.
  Imports werden nach dem ersten Paketnamen getrennt durch Zeilenumbrüche.

3. Formatierung

Maximal 100 Zeichen pro Zeile. Falls es nicht möglich ist, können unter Umständen, wie z.B. bei einer URL, mehr Zeichen pro Zeile benutzt werden.
Es gilt, dass eine Operation pro Zeile stattfinden sollte. Danach folgt ein Zeilenumbruch.

- Zeilenumbruch

Unter Umständen können durch Zeilenumbrüche Strukturen besser erkennbar gemacht werden. Falls dies der Fall ist, kann ein Zeilenumbruch benutzt werden, wenn:

- ein Separator wie Punkt, Komma, Und, Strich liegt vor
- eine Klammer geöffnet wird

- Einrückung
  Wir benutzen die automatische Eclipse-Formatierung „Eclipse [Built-in]“. Falls die Formatierung mit den hier genannten Regeln kollidiert, ist die automatische Formatierung vorzuziehen. Für Android Studio sollte eine möglichst äquivalente Formatierung benutzt werden.
- Jede Kontrollstruktur verwendet geschweifte Klammern. Dies gilt speziell für leere Blöcke.
- Geschweifte Klammern und Zeilenumbrüche für Blockstrukturen

```
return () -> {
  while (condition()) {
    method();
  }
};

return new MyClass() {
  @Override public void method() {
    if (condition()) {
      try {
        something();
      } catch (ProblemException e) {
        recover();
      }
    } else if (otherCondition()) {
      somethingElse();
    } else {
      lastThing();
    }
  }
};
```

1) Kein Zeilenumbruch vor einer öffnenden Klammer.

2) Zeilenumbruch nach einer öffnenden Klammer.

3) Zeilenumbruch vor einer schließenden Klammer.

4) Zeilenumbruch nach einer schließenden Klammer, falls danach die Kontrollstruktur abgeschlossen ist.

4. Variablen und Arrays

- Variablen
  Jede Variablendeklaration deklariert genau eine Variable. Also sind Mehrfachdeklaration untersagt, außer in eine for-Schleife.

Deklaration für **lokale** Variablen finden unmittelbar vor ihrer ersten Nutzung statt und werden i.d.R. initialisiert.

- Arrays
  Array Initialisierungen können blockweise geschehen. Die eckigen Klammern sind ausschließlich hinter dem Namen der Variable. Außerdem gilt bei der Deklaration nicht der C-Style. Die eckigen Klammern stehen **immer am Datentypen**: `String[] args`.

5. Switch-Statements

- Zeilenumbrüche
  Nach jedem Switch-Label ist ein Zeilenumbruch.

- Case ohne Terminierung (break, continue, return)
  Es muss deutlich gemacht werden, dass nach diesem Fall noch weitere Cases folgen können, z.B. durch den Kommentar “fall through”.

- Default-Case
  Jedes Switch-Statement hat einen Default-Case. Falls **alle** möglichen Eingaben mit einem Fall behandelt werden, kann der Default-Case ausgelassen werden.

6. Annotations

Jede Annotation ist über der eigentlichen Methode/Variable aufgelistet und ist nur durch einen Zeilenumbruch von den anderen getrennt.

7. Modifizierer

- Folgende Präzedenz:
  public protected private abstract default static final transient volatile synchronized native strictfp

8. Suffixe

Es wird immer der **Großbuchstabe** benutzt, um einen Suffix, wie z.B. *long x = 1L;* anzugeben.

9. Namengebung

In diesem Abschnitt wird oft von camelCase gesprochen, was in der **Google Java Style Guideline** definiert ist.

- Generelle Regeln
  Wir benutzen ausschließlich ASCII Zeichen um Namen für Sachverhalte zu deklarieren. Die Namensgebung findet ausschließlich in Englisch statt. Generell ist jeder Name, sei es für eine Variable oder Klasse, nach dem „*Keep it Simple, stupid*“(Kiss-Prinzip) zu gestalten. Des Weiteren soll es direkt möglich sein, anhand des Namens, den Sinn der Variable/Klasse etc. zu verstehen. Variablen, die aus einzelnen Buchstaben bestehen werden in der Regel nicht verwendet, außer es handelt sich um Zählervariablen, wie in Schleifen, oder um temporäre Variablen. Von einem Unterstrich innerhalb eines Namens ist abzusehen, außer bei der Erstellung von GUIs. Dort soll, um Kontrollelemente zu benennen, sich an folgende Form gehalten werden: [Kontrollelement camelcase]_[fx:ID im Scenbuilder camelcase], z.B.:

button_changeView. Falls Abkürzungen verwendet werden, sollten diese für jeden anderen Entwickler ebenfalls schnell verständlich sein.

- Pakete
  Pakete haben keine Unterstriche und jeder Paketname wird **komplett** klein geschrieben.
- Klassen/Interfaces
  UpperCamelCase - jedes Wort im Namen wird mit einem Großbuchstaben begonnen. Falls ein bestimmtes Designpattern benutzt wurde, so steht der Komponentennamen des Patterns noch vor dem Klassennamen, z.B.: *CompositeTitel.*
- Methoden
  lowerCamelCase - jedes Wort außer das erste im Namen beginnt mit einem Großbuchstaben. Methodennamen sind i.d.R Verben, die auch mit Nomen zusammengesetzt werden können.
- Konstanten
  Jede Konstante hat einen Modifizierer *static final* und ist **vollständig** in Großbuchstaben zu schreiben.
- Nicht-Konstanten (Variablen)
  lowerCamelCase - i.d.R Nomen oder gemischt mit Verben
- Parameter
  lowerCamelCase - Parameter haben immer mehr als einen Buchstaben
- Typvariablen
  Entweder **ein** Großbuchstabe oder einen Namen, worauf ein Großbuchstabe folgt.

10. Lambdas und anonyme innere Klassen

Beispiele zu Lambdas und Methoden sind hier (Abschnitt 4 und 5) zu finden.

- Lambdas
  Generell sollten Lambdas genutzt werden, da es der Lesbarkeit dient und Code minimiert. Es ist auch freigestellt, Anonyme innere Klassen zu benutzen.

11. Programmiergepflogenheiten

- @Override immer benutzen, falls Nutzung dafür vorgesehen, Methode/Klasse zu überschrieben.
- Statische Methodenaufrufe immer auf der Klasse der Methode.
- Exception
  In der Regel werden Exceptions **nicht** ignoriert. Falls es einen ersichtlichen Grund gibt, eine Exception nicht aufzufangen, wird dieser durch einen Kommentar beschrieben.
- Für Tests kann es unter Umständen möglich sein, gewollt Exception auszulösen. Diese werden dann *ExceptionXY expected* genannt.

12. Kommentare

Kommentare stehen immer über dem Sachverhalt, den sie beschreiben. Außerdem sollten Kommentare so geschrieben werden, dass sie die Einarbeitung in den Code vereinfachen. Falls ein Kommentar per „//“ gestartet wird, so sollte dieser Maximal 30 Zeichen enthalten.

Bei größeren Texten wird geraten „/* */“ zu benutzen. Die Kommentierung einer Klasse wird mit folgendem Kommentar geleistet: „/** */“.

13. JavaDoc

JavaDoc ist ein Dokumentationstool, das Informationen aus dem Quellcode extrahiert und als HTML-Dokumentationsdatei speichert. Es ist seit Version 2 ein Bestandteil des Java Development Kits und bietet neben den standardisierten Tags auch die alternativen Tags durch die Verwendung von Doclets. Dabei „parst“ dieses Tool die zu dokumentierenden Java-Quelltexte und sucht nach Kommentaren beginnend mit „/**“ und Tags in diesen Kommentaren beginnend mit „@“ oder „{@“.

- Tag: Wort beginnend mit „@“ oder „{@“ im Java Quelltext in einem Kommentar
- Taglet: Erweiterungsmöglichkeit des Standard-Wortschatzes
- Doclet: Erzeugt Ausgabe in Datei (Standardweise in HTML)

- Unter welchen Umständen sollte ein Kommentar für JavaDoc erstellt werden?
  Generell sollte jede Methode mit einem Kommentar für JavaDoc versehen sein. Dies gilt nicht für Getter/Setter und implementierte/geerbte Methoden von Interfaces/Klassen, die bereits einen entsprechenden Kommentar haben. Falls aber neue Funktionalität beim Erben entsteht, so sollte diese erwähnt werden. Außerdem sollte jedes Interface/Klasse/Enum eine kurze Beschreibung aufweisen, welche zwischen dem letzten Import und der Klassendeklaration steht.

Rot: Wird nicht verwendet
Grün: Verwendung gefordert
Gelb: Optionale Anwendung, verschönert Endergebnis

Tag und Parameter	Ausgabe	Verwendung in
@author name	Beschreibt den Autor.	Klasse, Interface
@version version	Erzeugt einen Versionseintrag. Maximal einmal pro Klasse.	Klasse, Interface
@since jdk-version	Seit wann die Funktionalität existiert.	Klasse, Interface, Instanzvariable, Methode
@see reference	Erzeugt einen Link auf ein anderes Element der Dokumentation.	Klasse, Interface, Instanzvariable, Methode
@serial	Beschreibt die serialisierten Daten eines Serializable-Objekts.	Klasse
@serialField	Beschreibt ein Feld eines Serializable-Objekts.	Klasse, Methode
@param name description	Parameterbeschreibung einer Methode.	Methode
@return description	Beschreibung des Rückgabewerts einer Methode.	Methode
@exception classname description	Beschreibung einer Exception, die von einer Methode geworfen werden kann.	Methode

@throws classname description		
@deprecated description	Beschreibt eine alte Methode, die nicht mehr verwendet werden sollte. Sollte ab Java 5.0 immer mit der @Deprecated-Annotation verwendet werden.	Methode
{@inheritDoc}	Kopiert die Beschreibung aus der überschriebenen Methode.	Überschriebene Methode
{@link reference}	Link zu einem anderen Symbol.	Klasse, Interface, Instanzvariable, Methode
{@linkPlain reference}	Der Link wird in Standardtext statt in Quelltextzeichen angezeigt.	Klasse, Interface, Instanzvariable, Methode
{@value}	Gibt den Wert eines konstanten Feldes zurück.	Statisches Feld
{@docRoot}	Gibt den Pfad zum Hauptverzeichnis wieder.	Package, Klassen, Felder, Methoden
{@code}	Formatiert Text buchstabengetreu mit dem Quelltextsatz (entsprechend <code>) und blockiert die Interpretierung von beinhalteten HTML oder Javadoc-Tags.	Klasse, Interface, Instanzvariable, Methode
{@literal}	Kennzeichnet buchstabengetreuen Text und unterdückt die Interpretierung von beinhalteten HTML oder Javadoc-Tags.	Klasse, Interface, Instanzvariable, Methode