Coding Conventions und Metriken

Carsten Gips (HSBI)

Coding Conventions: Richtlinien für einheitliches Aussehen von Code

=> Ziel: Andere Programmierer sollen Code schnell lesen können

- **Namen, Schreibweisen**: UpperCamelCase vs. lowerCamelCase vs. UPPER_SNAKE_CASE
- **Kommentare** (Ort, Form, Inhalt): Javadoc an allen `public` und `protected` Elementen
- **Einrückungen und Spaces vs. Tabs**: 4 Spaces
- **Zeilenlängen**: 100 Zeichen
- **Leerzeilen**: Leerzeilen für Gliederung
- **Klammern**: Auf selber Zeile wie Code

Beispiele: Sun Code Conventions, Google Java Style, AOSP Java Code Style for Contributors

Beispiel nach Google Java Style/AOSP formatiert

```java
package wuppie.deeplearning.strategy;

/**
 * Demonstriert den Einsatz von AOSP/Google Java Style .................. Umbruch nach 100 Zeichen |
 */
public class MyWuppieStudi implements Comparable<MyWuppieStudi> {
    private static String lastName;
    private static MyWuppieStudi studi;

    private MyWuppieStudi() {}

    /** Erzeugt ein neues Exemplar der MyWuppieStudi-Spezies (max. 40 Zeilen) */
    public static MyWuppieStudi getMyWuppieStudi(String name) {
        if (studi == null) {
            studi = new MyWuppieStudi();
        }
        if (lastName == null) lastName = name;

        return studi;
    }

    @Override
    public int compareTo(MyWuppieStudi o) {
        return lastName.compareTo(lastName);
    }
}
```

Formatieren Sie Ihren Code (mit der IDE)

- IDE: Code-Style einstellen und zum Formatieren nutzen
- google-java-format: `java -jar google-java-format.jar --replace *.java`
- **Spotless** in Gradle:

```
plugins {
    id "java"
    id "com.diffplug.spotless" version "6.5.0"
}

spotless {
    java {
        // googleJavaFormat()
        googleJavaFormat().aosp()  // indent w/ 4 spaces
    }
}
```

`./gradlew spotlessCheck` (Teil von `./gradlew check`) und `./gradlew spotlessApply`

Demo: Konfiguration Formatter (IDE), Spotless/Gradle

Metriken: Kennzahlen für verschiedene Aspekte zum Code

- **NCSS** (*Non Commenting Source Statements*)
  - Zeilen pro Methode: 40; pro Klasse: 250; pro Datei: 300
    Annahme: Eine Anweisung je Zeile ...
- **Anzahl der Methoden** pro Klasse: 10
- **Parameter** pro Methode: 3
- **BEC** (*Boolean Expression Complexity*)
  Anzahl boolescher Ausdrücke in `if` etc.: 3
- **McCabe** (*Cyclomatic Complexity*)
  - Anzahl der möglichen Verzweigungen (Pfade) pro Methode + 1
  - 1-4 gut, 5-7 noch OK
- **DAC** (*Class Data Abstraction Coupling*)
  - Anzahl der genutzten (instantiierten) "Fremdklassen"
  - Werte kleiner 7 werden i.A. als normal betrachtet

Beispiel: Metriken an MyWuppieStudi#getMyWuppieStudi

=> Verweis auf LV Softwareengineering

Tool-Support: Checkstyle

- IDE: diverse Plugins: Eclipse-CS, CheckStyle-IDEA
- CLI: `java -jar checkstyle-10.2-all.jar -c google_checks.xml *.java`
- Plugin "**checkstyle**" in Gradle:

```
plugins {
    id "java"
    id "checkstyle"
}

checkstyle {
    configFile file('checkstyle.xml')
    toolVersion '10.2'
}
```

  - Aufruf: `./gradlew checkstyleMain` (Teil von `./gradlew check`)
  - Konfiguration: `<projectDir>/config/checkstyle/checkstyle.xml` (Default)
  - Report: `<projectDir>/build/reports/checkstyle/main.html`

Demo: IntelliJ, Checkstyle/Gradle

Checkstyle: Konfiguration

```
<module name="Checker">
    <module name="LineLength">
        <property name="max" value="100"/>
    </module>

    <module name="TreeWalker">
        <module name="AvoidStarImport"/>
        <module name="MethodCount">
            <property name="maxPublic" value="10"/>
            <property name="maxTotal" value="40"/>
        </module>
    </module>
</module>
```

Demo: Konfiguration mit Eclipse-CS, Hinweis auf Formatter

SpotBugs: Finde Anti-Pattern und potentielle Bugs (Linter)

- **SpotBugs** sucht nach über 400 potentiellen Bugs im Code
  - Anti-Pattern (schlechte Praxis, "dodgy" Code)
  - Sicherheitsprobleme
  - Korrektheit
- CLI: `java -jar spotbugs.jar options ...`
- IDE: IntelliJ SpotBugs plugin, SpotBugs Eclipse plugin
- Gradle: SpotBugs Gradle Plugin

```
plugins {
    id "java"
    id "com.github.spotbugs" version "5.0.6"
}
spotbugs {
    ignoreFailures = true
    showStackTraces = false
}
```

`./gradlew spotbugsMain` (in `./gradlew check`)

Demo: SpotBugs/Gradle

Konfiguration für das PM-Praktikum (Format, Metriken, Checkstyle, SpotBugs)

Formatierung

- Google Java Style/AOSP: **Spotless**

Checkstyle

- Minimal-Konfiguration für **Checkstyle** (Coding Conventions, Metriken)

Linter: SpotBugs

- Vermeiden von Anti-Pattern mit **SpotBugs**

Wrap-Up

- Code entsteht nicht zum Selbstzweck => Regeln nötig!
  - Coding Conventions
  - Formatieren mit **Spotless**
  - Prinzipien des objektorientierten Programmierens

- Metriken: Einhaltung von Regeln in Zahlen ausdrücken
- Prüfung manuell durch Code Reviews oder durch Tools wie **Checkstyle** oder **SpotBugs**
- Definition des "PM-Styles"

LICENSE