**Java Development Standards within Naturalis**

Document management

**Name** [name of document]

**Version number** [version number of document]

**Version date** [version date document]

**Status** [status document: concept or final]

**Reviewed by** [names of reviewers]

**Administrator [**document administrator name]

**Entrance documents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| versie | titel | datum | auteur | status |
|  |  |  |  |  |

**Table of Contents**

**Introduction** [**1**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.qa020zruiaxj)

**Application of style guides** [**1**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.920e6mbuwvv6)

**Integrated Development Environment (IDE)** [**2**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.e9kovib1s5xi)

[**Java Compiler settings**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.5in5c64gs4iv)[**2**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.5in5c64gs4iv)

**Naming conventions** [**2**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.wdtnfw4gzdou)

[**Formatting**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.4r6ezie8tk5b)[**3**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.4r6ezie8tk5b)

**Testing** [**3**](https://docs.google.com/document/d/1ORr6vh3FGORpKIoLGhMnYyMYb32Rl9tNuTuic4q8vzA/edit#heading=h.j7zt69ibyxxx)

**Introduction**

This document describes the coding practices for Java development within Naturalis. In general, we follow the conventions and regulations as laid down in the Twitter Java Style Guide and the Google Java Style Guide.

This document is only a supplement to these two style guides. If the style guides have opposing views, we will make a choice in this document.

**Application of the style guides**

At present, our Java projects do not fully follow these style guides (for example regarding the use and placement of accolades). Adjust your code gradually, at the moment you encounter a "violation", to the rules of the style guides.

**Integrated Development Environment (IDE)**

**Java software development within Naturalis takes place by means of the Eclipse IDE.** However, we must bear in mind that any external consultants may use other IDEs. Eclipse-specific metadata files (such as .project) should therefore not be stored in the git repository; add them to .gitignore.

**Java Compiler settings**

Compiler settings are located in the file eclipse-prefs.epf, in the same folder as this document. Import eclipse-prefs.epf into Eclipse: File -> Import -> General -> Preferences

**Given the compiler settings in eclipse-prefs.epf, self-created Java code can never lead to warnings within Eclipse**. If Eclipse does give warnings on your code, the warning must either be solved or suppressed using @SuppressWarnings.

Examples of situations where the suppression of warnings may be legitimate:

1. If you have an unused variable or instance field, and you do not want to delete it for now, then mark it with @SuppressWarnings ("unused")
2. If you have an instance method that you could make static, but you have good reasons not to do so (eg with methods in JUnit test classes), mark the method with @SuppressWarnings (static method).
3. Especially with generics warnings are sometimes not avoided. Suppress that with annotations like @SuppressWarnings ("unchecked") and @SuppressWarnings ("rawtypes")

By always resolving warnings or explicitly suppressing them with @SuppressWarnings, you and your colleagues know that there is no "suspicious code" that you have overlooked or that you have not paid attention to.

An important exception to the "no warning" rule is generated code (such as JAXB classes), because you usually have no control over how the code is generated. This type of classes, and the package in which they are located, may therefore have a warning icon within Eclipse. Isolate these classes in a separate package.

**Naming conventions**

Use common Java naming conventions. See for example here in the Google Java Style Guide or the Twitter Java Style Guide.

**Some additions / emphases:**

1. Use nouns for classes and interfaces. For example: Vehicle, Bicycle, BrahmsDataImporter. Use short phrases in the authoritative or questioning ways for methods, for example startMotor, isMotorRunning, import. Think carefully about class names because they provide good class names self-documenting and self-disciplining code.
2. Preferably use camel case for abbreviations in identifiers. For example: CsvPrinter instead of CSVPrinter. Or the other way around. This is not a hard & fast rule. More a guideline that saves you some naming ideas.
3. The names generated by Eclipse derive from the method being tested; they are long and ugly. Moreover, you often want to write several test methods for 1 source method. Proposal: start one with test, followed by the name of the source method, followed by a counter, for example: testGroupByScientificName\_01 (). Use comments to explain what is being tested (instead of processing this in the name of the test method).
4. Use fully qualified names for system variables. For example: nl.naturalis.foo.bar.batchSize. This prevents clashes with libraries that happen to "pick up" a batchSize system variable.

**Formatting**

The formatting rules for Naturalis Java can be applied automatically within Eclipse with the above Eclipse preferences file: eclipse-prefs.epf, in the same folder as this document. Import eclipse-prefs.epf into Eclipse: File -> Import -> General -> Preferences.

Form your code before you push it to the central git repository (CTRL-SHIFT-F in Eclipse) and assume that another

Google Translate for Business:Translator ToolkitWebsite Translator

**Testing**

Although the Twitter and Google style guides say nothing about testing (after all, they are style guides), all code of unit tests and, if applicable, integration tests must be provided. For legacy code we use the principle of defect-driven development (make a unit test for code with a bug in it). For new code, we use the test-driven development method. So do not wait until you have almost finished writing unit tests.