TU/e

Project Fingerpaint

CaCSaS

# Conventions about Coding Style and Software


*Authors:*
Tessa Belder
Lasse Blaauwbroek
Thom Castermans
Roel van Happen
Benjamin van der Hoeven
Femke Jansen
Hugo Snel

*Junior Management:*
Simon Burg
Areti Paziourou
Luc de Smet

*Senior Management:*
Mark van den Brand
Lou Somers

*Advisor:*
Ion Barosan

*Customer:*
Patrick Anderson


April 25, 2013

**Abstract**

This document describes conventions about coding style, as well as the software we used in the Fingerpaint application . Note that in general, we followed the ESA coding style recommendations [1] in general and the Oracle Java coding conventions [2] in particular. Also, we used a couple of frameworks/libraries. These are described in this document, as well as their use and the reason(s) we have decided to work with those libraries. Finally, some rules about committing to the various repositories are included in this document.

# Contents

# Document Status Sheet

## Document Status Overview

### General

| | |
|---|---|
| Document title: | Conventions about Coding Style and Software |
| Identification: | CaCSaS-0.1 |
| Author: | Group Fingerpaint |
| Document status: | Draft |

### Document History

| *Version* | *Date* | *Author* | *Reason of change* |
|---|---|---|---|
| 0.1 | 24-Apr-2013 | Thom Castermans | Initial version. |

## Document Change Records Since Previous Issue

### General

| | |
|---|---|
| Date: | 24-Apr-2013 |
| Document title: | Conventions about Coding Style and Software |
| Identification: | CaCSaS-0.1 |

### Changes

| *Page* | *Paragraph* | *Reason to change* |
|---|---|---|
| - | - | Initial version. |

# Chapter 1

# Coding Style

In our project, we will use a coding style that is based on and resembles closely the ESA coding standard [1] and the Oracle Java coding conventions [2]. We will list all relevant conventions we made in this chapter. To keep this document concise, we will only list conventions that we find important. Other things that are left unspecified are done as described in the Java coding style. Note that the structure of this chapter is identical to the structure used in the referenced ESA document.

## 1.1 Readability Standards

In this section, general conventions that ensure the code is *readable* in general are described. This means that if a programmer scans through the code, he or she can understand the general idea of that part of the program. To achieve this, we will:

- Use a consistent writing style, namely British-English spelling where applicable.
- Use a consistent naming scheme, namely the one described in the Oracle Java coding conventions. For clarity, we quote those conventions here.
  - "Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words - avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML)."
  - "Interface names should be capitalized like class names."
  - "Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized." We want to clarify here that a method name should not *be* a verb, but *start with* a verb. For example, `getBackground` is a correct method name. Furthermore, names of getters should start with `get` and names of setters likewise with `set`.
  - Like methods, variables should be in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. However, constants should be in upper case.

    "Variable names should be short yet meaningful. The choice of a variable name should be mnemonic - that is, designed to indicate to the casual observer the intent

of its use. One-character variable names should be avoided except for temporary throwaway variables. Common names for temporary variables are `i`, `j`, `k`, `m`, and `n` for integers; `c`, `d`, and `e` for characters."

- Write all reserved words (`class`, `public`, ...) in lower case.
- Not use very long identifiers in general. Preferably, all identifiers should have a length of at most 15 characters. Class names can however get quite long, so for class names a name of at most 30 characters is acceptable.
- Be concise and prevent unused code from appearing in the program as good as we can.
- Write no more than one statement per line (except for writing `for` and `while` loops).
- Write lines of at most 80 characters wide.
- Use no tabs for indentation, or at least a tab size of a fixed size, namely 4 spaces.
- Group related constructs and separate different groups of related statements by blank lines.
- Vertically align comments and identifiers where applicable. In particular, some form of alignment in JavaDoc will be preferred:

```
/**
 * Concise description of a method in one line.
 * Some more explanation. This goes into details.
 *
 * @param param1Name Description of this parameter, that is
 *                   pretty long and aligned with the previous
 *                   line of description.
 * @param foo This parameter also has a length description that
 *            is aligned with the rest of the description, but
 *            NOT with the previous description.
 * @return Description of what is returned that - what a coincidence
 *         - is also pretty length and aligned with only its own
 *         description, not others.
 */
public static DataType solveProblem(param1Name, foo) { ... }
```

## 1.2 Naming Conventions

We want the code to be easily readable and thus, want to have easy-to-understand names for classes, methods and variables. Therefore, we will:

- Document each class, method and global variable with JavaDoc. Also, we will document local variables that are not temporary variables when needed. Variable names should be self-explanatory as much as possible.

- Use only the Latin alphabet as far as applicable, that is, refrain from using special characters and/or Chinese characters for example. Digits in identifiers can be used, but it is not recommended to do so normally.
- Make sure that identifiers indicate the purpose of a method/variable, not the (return)type.

## 1.3 Comments

As described in the ESA coding standards, we want comments to be explanatory, not a restatement of the code. Also, when an identifier is self-explanatory, no further comments are needed there. Thus, something like the following is undesirable:

```
int childCount; // count of number of children
```

In general, comments should be as close to the code as possible. JavaDoc can be written in the lines directly preceding the line with the identifier that is documented, while explanatory comments for local variables can be on the preceding line or even on the same line after the identifier that is documented.

# Chapter 2

# Software Used

Bluppetie.

# Chapter 3

# Committing Conventions

Committing is cool.

# Chapter 4

# Reviewing Conventions

Internally, we will do reviews of documents

# Chapter 5

# References

[1] ESA [3], "ESA coding standard." http://web.cecs.pdx.edu/~mperkows/CLASS_VHDL_99/545_14-ESA-coding-standards.pdf, August 2001. [Online; accessed 24-April-2013].

[2] Oracle, "Oracle java coding standard." http://www.oracle.com/technetwork/java/codeconventions-150003.pdf, September 1997. [Online; accessed 24-April-2013].

[3] ESA, *ESA Software Engineering Standards*. March 1995.