1 Introduction

This document serves as the definition of POTATOLK's coding standards in the Java Programming Language. A java source is described as being in Potatolk's Style only if it adheres to the rules herein.

Like other programming style guides, the issues covered span not only aesthetic issues of formatting, but other types of conventions or coding standards as well. However, this document focuses primarily on the **hard-and-fast rules** that we follow universally, and avoids giving *advice* that isn't clearly enforceable (whether by human or tool).

1.1 Terminology notes

In this document, unless otherwise clarified:

1. The term *class* is used inclusively to mean an "ordinary" class, enum class, interface or annotation type (@interface).
2. The term *member* (of a class) is used inclusively to mean a nested class, field, method, *or constructor*; that is, all top-level contents of a class except initializers and comments.
3. The term *comment* always refers to *implementation* comments. We do not use the phrase "documentation comments", instead using the common term "Javadoc."

Other "terminology notes" will appear occasionally throughout the document.

2 Source file basics

2.1 File name

The source file name consists of the case-sensitive name of the top-level class it contains plus the .java extension.

2.2 File encoding: UTF-8

Source file are encoded in UTF-8.

2.3 Whitespace characters

Tab characters are used for indentation.

3 Source file structure

A source file consists of, in order:

1. License or copyright information, if present
2. Package statement
3. Import statements
4. Exactly one top-level class

Exactly one blank line separates each section that is present.

4 Formatting

4.1 Braces

4.1.1 Braces are used where optional

Braces are used with **if, else, for, do** and **while** statements, even when the body is empty or contains only a single statement.

4.3 One statement per line

Each statement is followed by a line break.

4.4 Variable declarations

4.4.1 One variable per declaration

Every variable declaration (field or local) declares only one variable: declarations such as int a, b; are not used.

Exception: Multiple variable declarations are acceptable in the header of a for loop.

4.4.2 Declared when needed

Local variables are not habitually declared at the start of their containing block or block-like construct. Instead, local variables are declared close to the point they are first used (within reason), to minimize their scope. Local variable declarations typically have initializers, or are initialized immediately after declaration.

4.5 Arrays

4.5.1 Array initializers: can be "block-like"

Any array initializer may *optionally* be formatted as if it were a "block-like construct." For example, the following are all valid (not an exhaustive list):

4.6 Annotations

Annotations applying to a class, method or constructor appear immediately after the documentation block, and each annotation is listed on a line of its own. These line breaks do not constitute line-wrapping, so the indentation level is not increased.

4.7 Comments

This section addresses *implementation comments*. Javadoc is addressed separately in Section 7, Javadoc.

Any line break may be preceded by arbitrary whitespace followed by an implementation comment. Such a comment renders the line non-blank.

5 Naming

5.1 Package names

Package name are all lowercase, with consecutive words simply concatenated together (no underscores). They are all based on the opcodes of the protocol specification. (https://2020-5ei-team6-trentin.readthedocs.io/en/latest/PCP-Min/)

5.2 Method names

Method names are written in lowerCamelCase.

Method names are typically verbs or verb phrases.

5.3 Parameter names

Parameter names are written in lowerCamelCase.

One-character parameter names in public methods should be avoided.

5.4 Local variable names

Local variable names are written in lowerCamelCase.

Even when final and immutable, local variables are not considered to be constants, and should not be styled as constants.

6 Programming Practices

6.1 @Override: always used

A method is marked with the @Override annotation whenever it is legal. This includes a class method overriding a superclass method, a class method implementing an interface method, and an interface method respecifying a superinterface method.

6.2 Caught exceptions: not ignored

Except as noted below, it is very rarely correct to do nothing in response to a caught exception. (Typical responses are to log it, or if it is considered "impossible", rethrow it as an AssertionError.)

When it truly is appropriate to take no action whatsoever in a catch block, the reason this is justified is explained in a comment.