Coding Conventions

**1.0** **Source Files**

1.1 Source file naming

1.2 Source file organization

**2.0** **Naming Conventions**

2.1 Package naming

2.2 Class and Interface naming

2.3 Variable naming

2.4 Method naming

**3.0** **White Space Usage**

3.1 Blank lines

3.2 Blank spaces

3.3 Indentation

**4.0** **Comments**

4.1 Single-line comments

4.2 Block comments

**5.0** **Classes**

5.1 Class declaration

5.2 Class body organization

5.3 Method declaration

5.4 Encapsulation

5.5 Member documentation

**6.0** **Interfaces**

**7.0** **Braces**

7.1 Default Rule

7.2 Conditionals/Loops

7.3 Rule Exceptions

**8.0** **References**

8.1 <http://chukin.ru/book/java-style.pdf>

**1.0 Source files**

Java files should only contain a single public class or interface declaration. Nested classes are allowed (See 5.1 for more info). Files should be kept below 1500 lines of code, any more is a code smell and an indicator that the class should be broken up.

**1.1 Source file naming**

Source files should be of the form:

ClassName.java or InterfaceName.java

As source files contain a single class or interface, the name of the source file should be the exact name of the public class/interface it contains.

**1.2 Source file organization**

Components of a source file are laid out in the following order

1) Author comment block

2) Package declaration

3) Import declarations

4) Class/interface declaration

**2.0 Naming Conventions**

**2.1 Package naming**

Package names should be all lowercase and contain no underscores. Unique prefixes are constructed using the components of its internet domain in reverse. For an example:

com.company.product

**2.2 Classes and Interface Naming**

All names should use camel case with the first letter capitalized. Do not use underscores in class or interface names. Class names should ideally be a noun or contain a noun. Interfaces should (when possible) be an adjective to match java API conventions. Examples are Comparable, Observable, etc.

Positive examples: Car, AttackCommand, WebServer

Negative examples: CommandAttack, Web\_Server

**2.3 Variable Naming**

Variables start with a lowercase letter with every new word capitalized. Do not use underscores unless you are declaring a constant. Constants are all uppercase, with underscores used to separate successive words. For an example,

final int BUFFER\_SIZE // <- Good

final int bufferSize// <- Bad

Single character names are acceptable only in instances where they are used to iterate or are temporary variables. In these cases, i,j,k,m,n,x, and y are acceptable integer names. For every other datatype just use the first letter of the data type (eg: String s = “”)

**2.3 Method Naming**

Method names are similar to variable names. Start with a lowercase letter, then capitalize the first letter of every new word without using underscores. Method names should be functional and describe the action they represent. showStatus() is a good functional method name, where mouseButton() sounds more like a class name.

**3.0 White Space Usage**

**3.1 Blank lines**

Blank lines should be used: after copyright block comments, package declarations, import sections, between class and method declarations. Also between the last field declaration and first method declarations in a class.

**3.2 Blank spaces**

Blank spaces are used:

Between keyword and the opening parenthesis (keywords: catch, for, if, switch, while). A space is needed after keyword that takes an argument, between two adjacent keywords, between a keyword or closing parenthesis, and opening braces. There should be a space after a comma in a list. Example:

if (obj instanceof Button) {

**3.3 Indentation**

Line indentation is always 4 spaces.

**3.4 Continuation lines**

If a line gets too long and needs to be broken down unto continuation lines, the following lines should be indented additional 4 spaces. Example:

foo(longExpression1, longExpression2, longExpression3,

longExpression4);

**4.0 Comments**

In general excessive comments are a bad sign. Software should be self-commenting for the most part. That is to say, the names of classes, methods, and variables should in of themselves describe the functionality they represent. When possible avoid them, but if a particular section needs additional explanation feel free to use them as needed.

**4.1 Single line comments**

Single line comments are denoted with a //. There are two acceptable locations for single line comments. If the comment is a short phrase and the commented area is an operation in a method, the comment can be on the same line positioned after the statement like so:

(some complex or opaque operation) //comment describing intent

(an additional operation) //comment describing intent

As shown in the example, this type of single line comment should line up to form a column. If the commented item is a method, or a class field, or the comment itself is too long to put on the same line as the item it is acceptable to place it on it’s own line above the item.

//comment describing the operation in more detail

(some complex operation)

**4.2 Block Comments**

In general block comments are discouraged. Unless being used to create a javaDoc or in the author block at the top of files, they should be avoided. This is mostly because their primary use to comment out source code, but since they do not nest inside each other, using them for descriptions and for standard use prevents the source code from being block commented out.

**5.0 Classes**

**5.1 Class declaration**

Classes are declared in the following format

public abstract class MyClass extends Superclass implements

Interface {

}

In the case that the class declaration takes excessive space, the class can be declared like so:

public abstract class MyClass

extends Superclass

implements Interface {

}

**5.2 Class body organization**

Class body components are organized in the following way

1. Static variable declaration
2. Instance variable declaration
3. Static initializers (EG: getInstance() in the singleton model)
4. Static inner class declarations
5. Instance initializer
6. Instance constructor
7. Instance inner classes
8. Instance methods
9. Instance getters/setters

**5.3 Method declaration**

Methods use the following format. Items in brackets are optional.

[Modifiers] DataType methodName(DataType parameter1, DataType

parameter2)[throws Exceptions]{

}

If the method signature proves long, it is also acceptable to use the following format:

[Modifiers] DataType methodName(

DataType parameter1,

DataType parameter2,

DataType parameter3)

[throws Exceptions]{

}

**5.4 Encapsulation**

All members should be aggressively encapsulated to the point where the only publicly visible members are part of the documented interface of the class. This is vital to ensuring software is loosely coupled.

**5.5 Member documentation**

Comments should be minimal, as excessive comments is a sign that some sort of refactoring may be needed. However, in cases where it cannot be avoided, avoid using block comments. Refer to 4.1 for single line comment information. If more than one line is needed, use the following format.

//This is the first line of the comment and notice how it

//carries onto the next line. This comment explains the method

public void someComplexOperation(){

}

**6.0 Interfaces**

Interface declaration is mostly the same as class declaration. The main difference is that there is no need for visibility modifiers as Interfaces can only have public modifiers. Interface bodies must be organized in the following way:

1. Interface constant field declarations
2. Interface method declaration

**7.0 Braces**

**7.1 Default rule**

By default, opening braces should be on the same line as the method or class declaration. Unless preceded by 7.2 or 7.3 this rule should be followed.

**7.2 Conditionals/Loops**

If a loop or conditional covers multiple lines it must use braces, but any instance where the brace can be left out it should be. If the statement is short enough, it is even acceptable to put conditional and statement on the same line. The following are all acceptable.

if(a > b) a = b;

int i;

if(someLongVariable > someOtherVariable && (i = bar) == foo)

performOperation();

**7.3 Rule Exceptions**

In the case where a function is a getter/setter, is so routine as to be clearly understood regardless of formatting, and is composed of a single operation it is acceptable to collapse the method onto one line.

int getMyNumber(){return this.myNum;}

This rule is left primarily to the discretion of the programmer, but this rule applies to most getters/setters.

**8.0 References**

This document was created using the Java Coding Style Guide by Achut Reddy as inspiration and a starting point. A pdf of his work can be viewed [here](http://chukin.ru/book/java-style.pdf).