<Library Management System>

**Java Coding Standard**

1. **Purpose**

This document describes how the LMS project code must be written with the Java language coding standards presented in the Java Language Specification.

1. **File Names**

This section lists commonly used file suffixes and names.

* 1. **File Suffixes**

JavaSoft uses the following file suffixes:

|  |  |
| --- | --- |
| File Type | Suffix |
| Java source | .java |
|  |  |

* 1. **Common File Names**

Frequently used file names include:

**File Name: Use:**

README Summary of the employees and targets of the project

1. **File Organization**

This section gives information about the import statement and class interface.

* 1. **Java Source File:**

Java source of the LMS project files have the following ordering:

* **Import statement**

The first non-comment line of most Java source files is import statements. For example:

import java.awt.peer.CanvasPeer;

* **Class and Interface Declarations**

The following table describes the parts of a class or interface declaration, in the order that they should appear.

1. **Indentation**

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 4 spaces.

* 1. **Line Length**

Lines longer than 80 characters are avoided.

* 1. **Wrapping Lines**

When an expression will not fit on a single line, break it according to these general principles:

* Break after a comma,
* Break before an operator,
* Prefer higher-level breaks to lower-level breaks,
* Align the new line with the beginning of the expression at the same level on the previous line.
* If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.

if.jpgFor example from the LMS project:

1.png

2.png

1. **Comments**

**5.1. Implementation Comment Formats**

Programs can have four styles of implementation comments: block, single-line and trailing.

* **Block:**

Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments should be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe. Block commend is used like the following:

**/\***

**\* Description**

**\*/**

* **Single-Line:**

If a comment can’t be written in a single line, it should follow the block comment format. A single-line comment should be preceded by a blank line. Here’s an example:

**/\* Single line description \*/**

* 1. **Trailing:**

Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting. Here’s an example of a trailing comment in Java code:

**if (a == 2) {**

**return TRUE; /\* special case \*/   
} else {**

**return isprime(a); /\* works only for odd a \*/  
 }**

1. **Declarations**
   1. **Number Per Line**

One declaration per line is recommended since it encourages commenting. In other words,

int level; // indentation level

int size; // size of table

is preferred over

int level, size;

In absolutely no case should variables and functions be declared on the same line. Example:

long dbaddr, getDbaddr(); // WRONG!

Do not put different types on the same line. Example:

int foo, fooarray[]; //WRONG!

**Note**: The examples above use one space between the type and the identifier. Another acceptable alternative is to use tabs, e.g.:

int level; // indentation level

int size; // size of table

Object currentEntry; // currently selected table entry

* 1. **Placements:**

Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces “{” and “}”.) Don’t wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.

void MyMethod() {

int int1; // beginning of method block

if (condition) {

int int2; // beginning of "if" block

...

}

}

* 1. **Class and Interface Declarations**

When coding Java classes and interfaces, the following formatting rules should be followed:

* No space between a method name and the parenthesis “(“ starting its parameter list.
* Open brace “{” appears at the end of the same line as the declaration statement.
* Closing brace “}” starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the “}” should appear immediately after the “{“.
* Methods are separated by a blank line.

1. **Statements**

**7.1. return Statements**

A return statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

**return;**

**return myDisk.size();**

**7.2. if, if-else, if-else-if-else Statements**

The if-else class of statements should have the following form:

* **if (condition) {**

**statements;**

**}**

* **if (condition) {**

**statements;   
} else if (condition) {**

**statements;   
} else if (condition) {**

**statements;   
}**

* **if (condition) {**

**statements;   
} else {**

**statements;   
}**

**7.3. while Statements**

A while statement should have the following form:

**while (condition) {**

**statements;   
}**

1. **White Space**

**8.1. Blank Lines**

Blank lines improve readability by setting off sections of code that are logically related.

Two blank lines should always be used in the following circumstances:

* Between sections of a source file
* Between class and interface definitions

One blank line should always be used in the following circumstances:

* Between methods
* Between the local variables in a method and its first statement
* Before a block

**8.2. Blank Spaces**

Blank spaces should be used in the following circumstances:

* A keyword followed by a parenthesis should be separated by a space. Example:

**while** (i < librarians.size()) {

**…**

**}**

* Casts should be followed by a blank.
* The expressions in a for statement should be separated by blank spaces.

1. **Naming Conventions**

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier—for example, whether it’s a constant, package, or class—which can be helpful in understanding the code.

**Identifier Type Description**

Class 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

**Identifier Type Description**

Methods Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

Variables Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. 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.

1. **Programming Practices**

**10.1. Providing Access to Instance and Class Variables**

Don’t make any instance or class variable public without good reason. Often, instance variables don’t need to be explicitly set or gotten—often that happens as a side effect of method calls.

One example of appropriate public instance variables is the case where the class is essentially a data structure, with no behavior. In other words, if you would have used a struct instead of a class (if Java supported struct), then it’s appropriate to make the class’s instance variables public.

**10.2. Referring to Class Variables and Methods**

Avoid using an object to access a class (static) variable or method. Use a class name instead. For example:

Login.*record*.getBooks()

Record *record* = **new** Record();

*record*.getLibrarians()

**10.3. Constants**

Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a while loop as counter values.

**10.4. Miscellaneous Practices**

* **Parentheses:**

It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems. Even if the operator precedence seems clear to you, it might not be to others—you shouldn’t assume that other programmers know precedence as well as you do.

if (a == b && c == d) // AVOID!

if ((a == b) && (c == d)) // RIGHT

* **Returning Values:**

Try to make the structure of your program match the intent. Example:

**if (booleanExpression) {**

**return TRUE;   
} else {**

**return FALSE;   
}**

should instead be written as

**return booleanExpression;**

Similarly,

**if (condition) {**

**return x;   
}**

**return y;**

should be written as

**return (condition ? x : y);**