Java Style Guide

Dr Caffeine

This document defines the style convention the students must follow in submitting their programs. This document is a modified version of the document originally written by LCDR Chris Eagle.

1.0 General

The purpose of the style guide is not to restrict your programming, but rather to establish a consistent format for your programs. Consistency leads to easier comprehension and maintenance. The style guide presented in this document is based from the Java coding convention. The guide is available from

http://java.sun.com/docs/codeconv/html/CodeConventionsTOC.doc.html

2.0 Visual Layout

You should format your program source code making use of page boundaries and blank lines to separate code groups, functions and groups of logically related blocks. As a general guideline, no block should be longer than one page. It is generally better to have some blank space at the end of a page than to start a code block and run it over onto the next page. Use a lot of blank spaces.

Output from a program should be both informative and formatted to improve the readability. A person reading the output should be able to determine whether the program works reasonably, without knowing the internal details of the program.

A printed page must be 8.5 x 11 inches. Margins must be set between ½ and 1 inch all around. You must use a fixed-width font (Courier, or Courier New

works well) with a font size between 10 and 12 points.

3.0 Indentation and Whitespace

Indentation is used to highlight the block structure of the code and should be used consistently. The proper use of whitespace and indentation makes the structure of the program evident from the layout of the code. You must use proper indentation. The recommended indentation is **four spaces**. You may vary it from 2 to 4 spaces. Use spaces rather than tabs, because printers and monitors usually have different tab spacing. Change one of the options of the IDE that replaces the tabs with a designated number of spaces. It is critical to replace tabs with spaces so you can print the source code nicely.

Set you IDE to insert spaces for tabs.

Aligning the declarations make them more readable. Use

```
MainWindow      mainWindow;
FileInputStream fileInput;
File            inputFile;
String          stmt1, stmt2;
```

instead of

```
MainWindowmainWindow;
FileInputStreamfileInput;
File inputFile;
Stringstmt1, stmt2;
```

Vertical whitespace is the use of blank lines to help set off blocks of code and increases readability of the program. Code with no vertical whitespace is difficult to follow.

No line of code should wrap beyond the right margin.

If a line of code becomes too long, break it up into pieces with carriage returns. Remember the compiler largely ignores extra whitespace in your programs (but be careful with strings).

The only thing that may follow an opening brace (‘{‘) on a line of code is a

comment. The block of code enclosed by a set of braces must be indented one level (four spaces) inside the enclosing braces. A closing brace must always be on a line by itself and must be at the same level of indentation as the line containing the corresponding opening brace.

Lines of code may be indented more than four spaces when the readability of the code is improved.

3.1 COMMENTS

Comments should be indented consistently with the block of code they describe. Comments can be divided into two general categories, strategic and tactical.

Strategic comments are used to give the reader a general overview of what is going on. These comments appear at the beginning of files and preceding class, method, and class variable declarations. Strategic comments tend to be written in sentences with an emphasis on explaining the big picture of what the block of code is designed to do.

Tactical comments are designed to explain tricky areas of code, what parameters do, and hints about the control flow of the program. These comments should be intermixed in the code when needed. They should be shorter and in bullet format, and may be in inline format

If you use sensible and informative naming, block structure, indentation, and straightforward programming, your code will be mostly self documenting. This is what you should strive for. **Keep comments simple.** Do not explain what is obvious from the code itself.

Instead of commenting constants in the code, define symbolic constants. Do

```
final int MAX_WINDOW_WIDTH = 450;
...
width = Math.min(userInput, MAX_WINDOW_WIDTH);
```

instead of

```
width = Math.min(userInput, 450);
                        //450 is max window width
```

3.2.1 File Headers. Each file must have a descriptive comment header block at the beginning as follows:

```
//---------------------------------------------------------
// Filename:    _______.java <name of this file>
// Date:        ___________
// Course:      ___________
// Project #    ___________
// Compiler:    ___________  <JDK 1.4.0 etc>
//---------------------------------------------------------
```

3.2.2 Header Comments -- classes. Each class declaration must have an appropriate "comment header" giving a brief description using the following format:

```
/**
 *  A single sentence describing the purpose of the class.
 *  Any additional info describing the class
 *
 *  @author <your name>
 */
```

The javadoc documentation tool that can automatically generate html documentation pages for your source code recognizes this style of comment. Sun's Java API documentation was created using the javadoc tool.

3.2.3 Header Comments -- methods. Each method declaration must have an appropriate "comment header" giving a brief description using the following format:

```
/**
 *  A single sentence describing the purpose of the method.
 *  Any additional info describing the method
 *
 *  @param     <param-name> <param purpose>.
 *  @return    <describe the value returned>.
 *  @exception <any exception thrown by this method>
 */
```

```
EXAMPLE:

/**
 *  Return the square root of the specified value.
 *
 *  @param     value  the value to take the square root of.
 *  @return    the square root of the number.
 *  @exception ArithmeticException
 */

public float squareRoot(float value)
                        throws ArithmeticException {
   …
}
```

3.2.4 Header Comments – data members. Each class declaration must have an appropriate "comment header" giving a brief description using the following format:

```
/**
 *  A single sentence describing the purpose of the variable.
 */
```

If the comment fits in a single line, do this:

```
/** Date of employment */
private Date beginDate;
```

4.0 Control Structures

The use of braces is required (for style purposes, not syntax purposes) for all control structures, including those with a single statement in the body. Avoid the overuse of `continue` and `break` in loops. Always use a `default` case in `switch` statements. Use a `break` statement to end all cases (including `default`) in a `switch` statement. The use of `// end if, // end while`, and `// end switch` comments after the corresponding ending right brace is optional.

Indentation and formats for control structures are as follows (opening brace on the same line):

The if statement:

```
//left brace on same line
if (boolean expression) {
    //statement(s);

} else if (boolean expression) {
    //statement(s);

} else {
    //statement(s);
}
```

The for statement:

```
//left brace on same line
for (expression; expression; expression) {
    //statement(s);
}
```

The do/while statement:

```
//left brace on same line
do {
    //statement(s);

} while (boolean expression);
```

The while statement:

```
//left brace on same line
while (boolean expression) {
    //statement(s);

}
```

The switch statement:

```
//left brace on same line
switch (expression) {

    case <constant expression>:
            //statement(s);
            break;

    case <constant expression>:
            //statement(s);
            break;

    case <constant expression>:
            //statement(s);
            break;

    default:
            //all switch statement should
            //have default
            //statement(s);
            break;
}
```

5.0 Naming

Variables must have **meaningful names/identifiers**. This must be balanced against using names which are too long. Fifteen or so characters approaches the “too long” limit.

Single letter variables or constants should not be used. An exception to this rule is when it is common practice to identify some thing with a single letter. An example of this is the coordinate system (x, y, and z). A second exception occurs in the use of loop counter variables where it is common practice to use variables like i and j as counters in for loops

Only class names should begin with a capital letter, all other identifiers (package names, method names, and variable names) should begin with a lower case letter. An identifier consisting of multiple names shall have each name distinguished by making the first letter of each name part upper case (e.g. redCarColor). Package names should be all lower case.

Examples:

```
package  redcarcolor;
class    RedCarColor { ... }
int      redCarColor;
```

Names of constant (i.e. `final`) data members should be in all capital letters.

Do not use the lower case letter `L', or the letter `O' in names unless they are part of normal words. This is to avoid confusion with the numbers 1 and 0. For example, is "Cel1" C- e - l – ONE or C – e – l – l ?

Avoid names that differ only in case, look similar, or differ only slightly. For example, InputData, InData and DataInput will certainly be confusing if used in the same program.

Names of methods should reflect what they do (printArray), or what they return (getAge). Boolean variable names should sound like Yes/No things – "isEmpty" and "isFinished" are possible examples.

6.0 Classes

Classes are declared in the following order:

- public data members
- protected data members
- private data members

- constructors

- public methods
- protected methods
- private methods

- nested class definitions