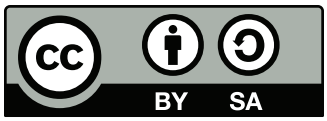
William Lyon Mackenzie C.I.
Java Programming Style Guide

2017



Contents

1 The Basics

1.1 Organized Code

1.1.1 Indenting

Indent your code with tabs if possible. Most IDEs and editors use tabs as the default. If your IDE doesn't use tabs, try and enable them in your IDE settings. Some older IDEs such as Dr. Java do not have this option. Because of this, no marks will be deducted for using spaces instead of tabs. Just make sure that your indentation is consistent.

There is no rule for how many characters wide indentation should be. However, most programmers use either four or eight character wide indentation. Virtually all IDEs (including Dr. Java, which uses two spaces as its default) allow this to be changed. It does not matter how wide your indentation is, as long as it is something sensible (at least two characters wide and no more than eight characters wide). The most important thing is to **be consistent**.

1.1.2 Whitespace

Generally, some whitespace is good, but too much whitespace is bad.

You should have an extra line:

- After field declarations.
- Before a new method.
- Between sections of code (put a line between your local variable declarations and the processing for example), although this is more of a guideline than a rule. Use your common sense!

Spaces have more complicated rules than new lines, but there are not many rules to them. For example: have a space after commas and before and after braces (unless the brace is the last character on the line). For more details on spacing, see the exemplar below.

1.1.3 Comments

Contrary to popular belief, you do **not** need to comment every other line of your code.

If you are using unfamiliar libraries, it may help to use more comments, but generally, lots of comments are not needed. There is nothing wrong with having extra comments, but they are rarely necessary. If you think that your code does not make any sense without comments, you should think if it can be written in a simpler way. If it cannot, then write some comments, but you will find that you can usually make your code a lot more KISS.

Again, this is a guideline, not a rule, but if you feel that you need comments **beyond** the Javadoc comments, then your code might not be KISS.

2 Javadoc

Proper Javadoc style consists of Javadoc comments for all fields, classes, and methods.

Javadoc comments are designated with two asterisks after the slash. They go on top of the method, field, or class declaration that they are explaining.

```
/** This is a Javadoc comment for a field. */
private int commentMe;

/**
 * This is a Javadoc comment for a method.
 *
 * @param num The number to square.
 * @return The square of the number.
 */
private int square(int num) {
    return num * num;
}

/**
 * This is a Javadoc comment for a class.
 *
 * <h2>Course Info:</h2>
 * ICS4U0 with Krasteva, V.
 *
 * @version 17.02.24
 * @author Vincent Macri
 */
public class MyClass {
    // Code
}
```

This allows the Javadoc generator to generate a webpage that explains everything easily. Modern IDEs will also pick up on your Javadoc comments, and remind you what the method, field, or class does when you hover your mouse over it.

There is one exception to the have a Javadoc comment for every method rule. If a method is overriding another method, and your changes are small enough that the Javadoc in the parent class still makes sense, you do not need a Javadoc comment. You should, however, use the @Override notation, to explicitly state that the method is being overridden. We do this because it does not make sense to write a comment for something that is already explained well in the parent class.

```
@Override
public void overriddenMethod() {
    // Trivial code
}
```

3 Exemplar

```java
/***************************************
 * Copyright Notice (Optional)
 ***************************************/

/**
 * An example FizzBuzz program.
 * <a href="https://en.wikipedia.org/wiki/Fizz_buzz">Wikipedia on FizzBuzz.</a>
 * <p>
 * This program demonstrates proper syntax and Javadoc style.
 *
 * <h2>Course Info:</h2>
 * Course Code with Teacher Name
 *
 * @version Version of the program.
 * @author Your name
 */
public class FizzBuzzExample {

    /** How high to count to. */
    private int counterEnd;

    /**
     * {@link FizzBuzzExample} constructor.
     * Sets the values of {@link #counterStart} and {@link #counterEnd}.
     *
     * @param endNumber How high to count to.
     */
    public FizzBuzzExample(int endNumber) {
        this.counterEnd = endNumber;
    }

    /**
     * Runs the FizzBuzz program.
     *
     * @throws Exception Demonstration of using throws.
     */
    private void fizzBuzz() throws Exception {
        for (int i = 1; i <= counterEnd; i++) {
            if (i % 3 == 0 && i % 5 == 0) {
                System.out.println("FizzBuzz");
            } else if (i % 3 == 0) {
                System.out.println("Fizz");
            } else if (i % 5 == 0) {
                System.out.println("Buzz");
            } else {
                System.out.println(i);
            }
        }
    }

    /**
     * The entry point to the program.
     *
     * @param args The arguments passed from the command line.
     */
    public static void main(String[] args) {
        FizzBuzzExample e = new FizzBuzzExample(30);
        try {
            e.fizzBuzz();
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }

}
```