Java Official Practices and Suggested Guidelines for Coding – Will Stuckey

This list contains some important Java conventions that all Java programmers should follow; the others are suggestions and tips learned over the past year or so. Anything in bold is a **Sun/Oracle convention**.

* A tab character will be defined as equal to four spaces.
* Do not use spaces for tabs, use the tab character.
* Drop all braces down to the next line, so they line up.
* **Tab every statement one tab past the braces.** (depends on the style convention used, but this is generally acceptable practice)
* **Do not use braces if their use is not required**. (e.g. one statement following a loop, or conditional statement)
* Tab all line continuation two tabs or 8 spaces.
* **Only have one (or less) statement per line.**
* **A key word followed by parenthesis should have a space.** (e.g. if (condition) not if(condition))
* **Casts should also be followed by a space.**
* If more than 3 parameters are used, drop down each parameter after the first leaving commas at the end and indenting two tabs. (eight spaces)
* If more than 3 logical operation are used, drop down each operation after the first, keeping the operators at the BEGINNING of the line and indenting two tabs. (eight spaces)
* If an exception must be caught but not handled DO NOT drop down the braces after the catch. (e.g. catch (Exception e) {})
* **Avoid catching the super class Exception**
* Do not use global import (e.g. import java.util.\*), many of these packages have hundreds of classes. Efficiency is key.
* Use import statements, do not link to the class in the code.
* There is no formal convention for organizing imports, make up your own and stick to it.
* **Packages are ENRITELY lowercase.** (e.g. mypackageisawesome)
* **Remember, packages are not low-level organizational tools.** (there should be at least 1-3 dozen classes to warrant the creation of a sub-package. Many packages contain hundreds of classes.)
* **Javadoc every method in every class, even if it is private or local protected**
* **The class header should be as follows**:

/\*

\* @author name, name, etc.

\* @date mm/dd/yy

\* <p>Class description</p>

\*

\* Notes.

\*/

* If methods or classes are overloaded, elucidate in the javadoc as to the differences and as to why it was necessary.
* If a File is taken as a parameter, explicate in the javadoc if the file is verified by the class, or should be done so prior by the user.
* use //TODO for personal to-do notes.
* **Classes are uppercased with each successive word having the first letter capitalized as well.** (e.g. MyClassName) This also applies to internal classes, abstract classes, and interfaces. Runner classes may be left lowercase.
* **Methods are lowercase with each successive word having the first letter capitalized as well.** (e.g. myMethodName)
* **Variables are lowercase with each successive word having the first letter capitalized as well.** (e.g. int myInteger)
* **Declare variables of the same type on the same line.**
* **Do not instantiate variables of the same type on the same line.**
* **Do not imbed assignments in an attempt to improve performance or save space.** This is the job or the JVM and compiler respectively.
* **Follow C++/ANSI naming conventions for constants, all caps separated by underscore**. (e.g. private static final int MY\_INTEGER\_IS\_A\_CONSTANT = 2)
* Always include access modifiers for fields, classes, and constructors, although not always required by the compiler/JVM (public, protected, private)
* **Methods/Fields modifiers are to be listed in the following order.** <access level> <static> <synchronized> <return type> <name>(parameters)
* **If a parameter's sole purpose is to initialize a field, make both the parameter and filed have exactly the same name and use the keyword this for initialization.**
* group fields as follows:

+ group by final

+ group by static

+ list groups in order of complexity (e.g primitives first [boolean, then int, then double, etc.], existing classes second, user classes third, JNI classes last)

* Static classes are to be declared with static fields, static methods, and an explicit, null, private, constructor.
* Flush streams as frequently as possible.
* If an object is taken as a parameter, when possible, instantiate that object in the reference call. Add a comment if necessary. This is from a memory management/GC relief stand point
* Do not call System.gc();
* Any code that has the potential to damage system elements SHOULD BE CLEARLY MARKED, and the primary call be commented and subsequently commented out until use, for safety purposes.
* If you intend to use code that may have a security flaw outside of the realm of personal use, include the security concern in the javadoc.
* Double check memory management on thread/runnable termination.
* Avoid using ArrayList<> with threads/runnables, use vector instead.
* Thread activity should be handled by a public, or private encapsulated, boolean flag named running.
* If a JNI, OS specific hook is used, specify the OS compatibility clearly and thoroughly in the javadoc.
* Do not save code that will not parse/compile. Finish the statement or comment it out so if the code OS called it will at least run although the result will not be as intended.
* Consider having a flag that enables or disables printing comments. This can be helpful if other people are using your code in their application. If allows them to enable your explicit debug if necessary, or disable it to keep their console clean.
* Recycle as much code as possible.
* Do not implement code that has been implemented before; find a library that supports your needs. Ensure the license allows you to use/publish it as well. Apache Commons has tons of this type of stuff.
* Use a runner when possible to keep potentially exportable code outside of the main method/main class.
* If you intend to distribute/publish source code, include a distribution license. The GNUv3 is a common, established freeware license.
* If you work in a team, all members must consent by law to change the distribution license, regardless of the will of the project leader.
* Use GetHub or SourceForge to distribute and share source. Both are free.
* Use dropbox for backups and team work.
* Use the ANSI system for software version. Understand the differences between pre-alpha, alpha, beta, and full release.
* Test on multiple platforms.