DTN #3 – Java Coding Guidelines

Bare bones Java coding guidelines
Updated: Jan 2008

I Stole This

I have stolen most of this from the creators of Java, Sun Microsystems. These guidelines are a tiny subset of Sun's. See?

java.sun.com/docs/codeconv

Rationale

Why am I bothering with this? Coding guidelines have a number of benefits:

- Improves your efficiency with a consistent and disciplined approach
- Easier to share your code with others
- Better to understand your "old" code in the future
- You will have guidelines in the "real world"... few people code in a vacuum

Number One

You can easily prevent the number one mistake of new coders:

> **Think, then code.**
> - Koffman & Wolfgang "Data Structures and Design Using Java"

Take a couple of minutes to sketch out what you want to do *before* you sit down to do it. If you can't do it on paper, then you won't be able to do it on your computer. You can dramatically improve your efficiency and lower your frustration level, thinking before coding.

Give it a try!

Just Ten

Just ten of these… not set in stone, but only break them with good reason:

1. Use .java suffix for Java files. *Rationale: many java tools expect this.*

2. One public class or interface per file. *Rationale: Java convention.*

3. Use Javadoc style comments for each class, and all variables and methods defined in your class. *Rationale: Explain the public parts of your class. Javadoc is a standard. You can then generate web pages documenting your API, just like Sun does for Java. Use at least the following tags:* `@author` *for classes,* `@param`*,* `@return` *for methods. See* java.sun.com/j2se/javadoc

4. Make class variables private or protected. *Rationale: This is a common object-oriented paradigm. Access to class variables is often provided by accessor (set) and mutator (get) methods.*

5. Use camel notation for methods and variables. *Rationale: Java convention. Camel notation starts with lower case and begins new words with upper case, for example:* `professorPayRaise`*.*

6. Capitalize class, interface and package names. *Rationale: Java convention. Example:* `MonsterTruck`.

7. Use UPPER CASE for constants, separating words with an underscore. *Rationale: Java convention. Example:* `MAX_HEAD_ROOM`.

8. Use consistent style for braces and indentation. *Rationale: It makes your code more readable. The NetBeans default is fine.*

9. Use curly braces for all if-then statements or loops. *Rationale: Prevent difficult bugs by surrounding blocks even if they only include a single statement.*

10. Use inline comments to explain difficult sections of your code. *Rationale: This makes your code easier to understand. Assume that your reader understands Java and CS… don't explain Java in your comments. Comments should appear before the code they explain.*

Some practical tips for you:

- Use good descriptive names for everything: files, classes, methods, variables, etc. If you abbreviate, be consistent throughout.
- Learn the Java tools available to you: the debugger in NetBeans, Javadoc, JUnit (for testing). Ask your instructor or a buddy about them.
- I generally leave a file in my project folder called `README.txt` for comments and status.

Organized, well-designed code is more efficient to create, easier to understand, and beautiful!

thanks… yow, bill