Java Coding Standards and Guidelines

Naming Conventions

- Use multiword identifiers. In mixed case names, capitalize the first letters of the contained words in multiword identifiers to enhance readability. For example: maxElement, currentFile, and so on. In single case names, separate the words in a multiword name with underscores. For example: BUFFER_SIZE.
- Use mixed case for class names and begin them with an uppercase letter.
- Use mixed case for package, method and variable names and begin them with a lowercase letter.
- Use only uppercase characters for constants (final variables).
- Use names that describe the roles of packages, classes, variables, methods, and constants—this generally means that names should be more than two or three characters long. For example, use lowerBound rather than lb. The exception is integer loop control variables used to index arrays, which are traditionally i, j, and k, and caught exception objects, which are traditionally e.
- Use the prefix is for boolean variables and methods, setX for operations that assign a value to an attribute X, getX for methods that retrieve the value of an attribute X, and toX for converter methods that return objects of type X.

Formatting

- Use standard indentation conventions for block structured languages.
- Indent 2 to 4 spaces at a time. An easy way to arrange this is to tab for indents, and set the tab to be 2, 3, or 4 spaces.
- Place at most one variable declaration, constant definition, or statement on a single line of code. Use the rest of the line for comments when necessary.
- Use vertical white space to separate code into segments that do parts of a whole task carried out in a block.
- Use horizontal white space to reflect precedence in expressions.

Types

- Avoid float in favor of double. This helps avoid underflow, overflow, and precision problems.
- Do not use char variables to store numbers.
- Avoid the byte and short types.
- Prefer void method return types—this forestalls long sequences of object method invocations that are hard to read.

- Use the easier-to-read Java array type declaration syntax rather than the C/C++ syntax. For example, use byte[] buffer = new byte[256]; rather than the older byte buffer[] = new byte[256] declaration.

Expressions

- Use parentheses liberally.
- Avoid expressions with side-effects, in particular, do not make assignments in boolean expressions.
- Simplify complex boolean expressions by factoring, and by using DeMorgan's Laws to drive negations inward.
- Use <= and < instead of >= and >.
- Use equal instead of == in object comparisons; in particular, do not use == to compare Strings.
- Make loop termination expressions as weak as possible.
- Avoid casts, but if they are used, embed them in a conditional so that a failed cast is handled correctly: if (x instanceOf C) then y = (C)x; else handleProblem().

Control Structures

- Avoid the continue statement, but if necessary, use a labeled continue statement.
- Except in switch statements, always use a labeled break statement.
- Use comments to mark missing loop parameters and missing switch break statements. Use either comments or an empty pair of brackets to mark null loop bodies.
- Always have a default case in switch statements.
- Make infinite loops with the construct while (true).

Constants, Attributes, and Variables

- Do not rely on automatic initializations—initialize all variables explicitly.
- Initialize all instance attributes in constructors, not at the point of declaration. Avoid instance initializers.
- Initialize all static attributes at the point of initialization, or if necessary, with static initializers, to ensure that they are all initialized independently of the creation of class instances.
- Declare local variables only when their initial values can be assigned (typically at the top of a block, but not always).
- Don't reuse local variables for some other task when they are no longer needed—declare and initialize a new one instead, thus improving readability.

- Declare all for loop control variables in the initialization portion of the loop header, for example: for (int i = 0; i < 10; i++), restricting variable scope.
- Assign null to reference variables that are no longer being used, especially arrays. The unused objects can then be garbage-collected.

Methods

- Avoid overloading on parameter type alone, that is, when overloading methods, create methods with different numbers of parameters, not just different parameter types. This will make code easier to read.
- Provide get and set methods for attributes only when they are really needed—many internal bookkeeping attributes should not be changed from outside the class.
- Make parameters constant (final) if possible, but not methods.

Classes

- Place each class (even non-public classes) in a separate program file.
- Order the contents of a class file in a standard way. For example, one could arrange to have all static members first (static finals, then static public, protected, package, and private attributes, then static public, protected, package, and private methods), then all non-static members (non-static finals, followed by public, protected, package, and private attributes, then public, protected, package, and private methods).
- Create a main method for every class that contains unit test or class demo code.
- Place the main method for an application in a standalone class (and a separate file) rather than in one of its principle component classes.
- Only declare constructor and finalize methods that do something.
- If a non-final class has any explicit constructors at all, then if possible it should have a constructor with no parameters. Otherwise subclassing may cause constructor chaining errors.
- Add the call super.finalize() to the end of any finalize method. This assures finalizer chaining, which is not done automatically in Java.
- Avoid static attributes and methods, except static final constants.
- Avoid naming an attribute in a subclass after an attribute in a superclass (attribute shadowing)—this is very confusing.
- To promote information hiding, prefer private to package attributes, package to protected attributes, and protected to public attributes; in particular, never declare a non-final attribute public.
- Avoid declaring a class final—you may need to subclass it later.
- Prefer interfaces over abstract classes, for maximum flexibility in inheritance.

- Try to flatten and simplify inheritance hierarchies.

Packages

- Form a package for each self-contained project or group of related classes, and create and use directories in accord with the Java conventions for mapping between package names and directories.
- Avoid the anonymous (unnamed) package—place all classes in a named package.
- Minimize wildcard importing (e.g., import aPackage.*) and make sure all imported classes are actually used.

Comments

- Use C-style comments (/* or /** up to */) for block or banner comments, and C++-style comments (// to the end of the line) for shorter in-line comments. Also use the C-style comments for hiding debugging and testing code.
- Start each class with a block comment stating the class name, purpose, author, date of creation, class invariant (optional), and notes explaining any special features of the class. Javadoc tags should be used where appropriate. For example:

```
/**
 *  class description--should include the purpose, provenance,
 *  class invariant, and notes on origin and use.
 *
 *   @author author-name
 *  @version version-number-and-date
 */
```

- Precede each method definition with a banner comment describing the method, including its origin, purpose, parameters, return value, thrown exceptions, post-conditions, and notes explaining any side effects, explanations of algorithms, and so on. Javadoc tags should be used where appropriate. For example:

```
/**
 * method description--should include post-conditions and
 * notes on origin and use of this method.
 *
 *   @param param-name description
 * @returns description of what is returned
 *  @throws exception-class-name description
 */
```

- In long sequences of code, break the code into cohesive blocks and precede each one with a summary of the processing carried out in the block.
- Note for every parameter whether it is an *in*, *out*, or *in/out* parameter.
- Follow each variables declaration with a comment explaining the purpose of the variables (its role should be recorded in its name).

- In long sequences of code, break the code into cohesive blocks and precede each one with a summary of the processing carried out in the block.
- Document unexpected side-effects in code segment comments (like header comments), at the point of declaration of the affected object and at the points where the side-effect occurs.

Error Checking and Exception Handling

- Always try to handle exceptions as close to the spot they are thrown as possible—in other words, avoid exception propagation.
- Check (and optionally modify) method or exception parameters likely to be used improperly before doing any processing.

Target Metric Values

- Aim for method comment-to-code ratios of at least 0.8.
- Try to write method with no more than 60 NCSL.
- Try to write classes with no more than 500 NCSL.
- Do not exceed a block nesting level of 7.