Java: Coding Guidelines v2 - Summary

Motivation. Using good programming style, common idioms, and best practices is essential to produce reliable software efficiently. Good software is quicker to build, cheaper to maintain, and more fun (or at least less difficult) to work with. But no set of rules can cover all situations - you must also have guiding principles. A good start would be to use the following.

- Readability - Make your program as readable as possible (good names, spacing, ...).
- Simplicity - Don't add unnecessary complication and duplication. KISS.
- Convention - Use standard conventions and good practices as much as possible.

---

1. Comments, indentation, spacing, braces, ...
    - Comments
        - Header comments at front of each file with purpose, author, date ...
        - "Paragraph" comments at beginning of each group of code.
        - Document "tricks" -- anything unobvious.
        - Avoid useless comments. Don't comment code that is already clear.
        - Using javadoc is essential for larger projects.
    - Indentation
        - Use either K&R or Allman indentation style. Don't use others.
        - Indent size 4 spaces
        - Blanks, not tabs
        - Use your IDE's indentation tool.
        - Continued statements should be indented two indentation levels.
    - Spaces and blank lines - Use them to make the source more readable
        - No space between method name and the left parenthesis. `f(x)`, not `f (x)`.
        - One space between keywords (if, while, for, switch, ...) and the left parenthesis.
        - Spaces around assignment and many other operators.
        - Space after comma. `f(a, b)`.
        - Put blank lines between elements - methods, inner classes, ...
2. Variable (local, instance, class) Declarations
    - One per line. Declare more than one only when there are very closely related (eg, x and y).
    - Add // comment if meaning of variable isn't completely clear or if has special range, values, etc.
    - Don't reuse variable name for more than one purpose. Variables are cheap.
    - Local variables (declared in method)
        - Declare at first use is preferred to declaring at front.
        - Give variable the least required scope (without adding extra blocks).
        - Declare within for loop header if possible.
        - Don't hesitate to create extra local variable if it improved readability.
    - Instance variables (fields)
        - Declare them private.
    - Static (class) variables
        - Use *static final* for named constants.

        - Static (class) variables are rare.
3. Braces
    - Use them, even for single statements.
    - K&R or Allman style
4. Naming conventions
    - Names must be meaningful if possible.
    - Conventional names can be used: i, j, iter.
    - Case rules should be followed. class - start upper, vars - start lower, const - all upper).
    - Instance variables (fields) may start with prefix (eg, "_", "m", ...).
    - Use plural names of mass nouns for arrays and data structures.
    - Avoid confusing name duplicates (method same as field, differing only by case, ...).
    - Don't use a $ in an identifier name.
5. Error Handling
    - "Happy Trails" programming allowed for most student programs.
    - Recover from user input errors.
    - Crash on programming errors is acceptable. Debugging output is useful.
    - Catching exceptions
        - Don't catch an exception if you can't handle it.
        - Don't use exceptions to handle normal flow (eg, ArrayIndexOutOfBoundsException).
        - Don't use exceptions to hide program errors; fix them.
        - Don't silently ignore exceptions, except in those few cases where you can't do anything anyway.
    - Throwing exceptions
        - Don't throw `Exception` - make it more specific.
        - Put informative message in exception constructor.
        - If you're supplying a class for someone, throw meaningful exceptions.
6. Visibility
    - General rule: make data private, methods public.
    - Protected - Use only when designing for inheritance.
7. Miscellaneous
    - DRY - Don't Repeat Yourself. Repeated code should be replaced with loop/method/...
    - No magic numbers - use named constants.
    - Avoid premature optimization. Don't worry about optimization unless there's a problem.
    - KISS. Keep It Simple, Stupid. Always prefer a simple to a complicated solution.
    - Methods
        - Don't assign to parameters; it makes code more difficult to read.
        - Consider declaring parameters *final* to assure the reader they aren't changed.
        - Use multiple returns only if it makes the code clearer, otherwise return only at the end.
        - Methods should fit on one page/screen. If they're bigger, consider splitting them.
        - Avoid recursion unless it's a situation where it's much better (eg, traversing trees).
    - Loops
        - Use *for* instead of *while* if it groups everything in one statement.
        - *For* loop clauses should be *coherent* -- all related.

        - Use the *enhanced for* (*foreach*) when possible.
        - Don't change a *for* loop iteration variable in the body of the loop.
        - Use *break* to exit early or to terminate an "infinite" loop.
        - The *continue* statement is rarely used.
    - Switch
        - Switch statements should include a *default* clause.
        - Make *default* the last clause in a *switch*.
    - If
        - Do not write an empty true or else clause.
        - Writing positive logical expressions is more readable then negative. Change if easy.
    - Data structures
        - Use the generic forms of Java Collections data structures.
        - Use newer rather than older versions: HashMap<...> instead of HashTable, ArrayList<...> rather than Vector.
    - Methods
        - A method should be small and focused on one task.
        - Split a method into several methods if it operates on different levels of data.
        - Split a method if it becomes too large (longer than one screen is a common guideline).
        - Split a method which does different things.
        - Smaller methods are easier to understand, use, and debug.
        - It's OK to have utility methods that are only called from one place.

Not covered here. There's lots not covered in this list: Object-Oriented Design and Programming guidelines, including *Patterns*, GUI design guidelines, and development methodologies.

References

Code Complete 2nd Edition by Steve McConnell, Microsoft Press 2004, may be the single best book that addresses coding guidelines. Several languages are included in the examples, but the thoughtful commentary is very useful. An excellent book for the intermediate programmer.

Sun's guidelines. Perhaps the most common coding guidelines are Sun's *Code Conventions for the Java Programming Language*, which can be read online of downloaded in several forms from java.sun.com/docs/codeconv/. The guideline in the list above have a lot in common with the Sun Guidelines, although they differ on a couple of issues (eg, naming instance variables).