iCarnegie Java Code Conventions Quick Reference

http://java.sun.com/docs/codeconv/

Indentation

1. Use four spaces as the unit of indentation.
2. Use tabs or spaces to indent, not a mixture of the two. (Exception: A mixture can be used when wrapping lines).
3. Do not indent top-level classes and interfaces.
4. Indent variables, methods, and named inner classes one level.
5. Indent the body of a method one level.

Braces for methods, classes, and interfaces

1. Put opening brace on same line as declaration.
2. Put closing brace on new line and indent it to the level of the matching declaration.

```
class Example {

    private void doTask() {

        statements;
    }
}
```

Miscellaneous

1. Avoid lines longer than 80 characters.
2. One statement per line.
3. One declaration per line.
4. Initialize variables when they are declared except when the initial value is unknown.
5. If a control structure—like an if-statement or a for-loop—contains a single statement, the single statement should be enclosed in braces.
6. Use the class name, not a reference, to access static methods and variables.
7. Use parenthesis to clarify the order of evaluation in complex expressions.
8. Avoid coding literal constants directly. Use a well-named symbolic constant instead. (Exception: 0, 1, and –1 are acceptable.)

Implementation comments

1. *Do not add comments that state the obvious*.
2. A blank line should precede a comment.
3. Minimize the need for comments by making the code self-documenting with appropriate name choices and an explicit logical structure.
4. Comments should provide additional information that is not readily apparent in the code itself. Comments that present an overview of a code block can be useful.

```
// single-line comment
/* single-line comment */
/*
 * block comment
 */
statement;    // trailing comment
```

Javadoc comments:

1. Use to document classes, interfaces, methods, and variables (with class-scope).
2. Should describe the entity being documented from an *implementation-free* perspective.

```
/**
 * Javadoc comment
 */
/** Javadoc comment */
```

Wrapping lines

When a statement will not fit on a single line:

1. Break after a comma
2. Break *before* a binary operator
3. Prefer high-level breaks to low-level breaks
4. Align new line with beginning of expression (or argument list) on previous line:

```
a = b * (c + d – e)
    + (f / g);
x = getValue(a + b + c,
             d + e + f);
```

5. If these rules lead to confusing code or code that's jammed up against the right margin,

Naming conventions

1. Names should be words or word phrases. Keep names short but descriptive. Avoid abbreviations.
2. Classes and interfaces: Use nouns, in mixed case with first letter of each word capitalized. Examples: TextField and MouseListener
3. Methods: Use verbs, in mixed case with first letter lowercase and first letter of each internal word capitalized. Example: setBackground
4. Variables: Use nouns, in mixed case with first letter lowercase and first letter of each internal word capitalized. Example: fontSize
5. Constants: All uppercase with words separated by underscores. Example: EXIT_ON_CLOSE

Blank lines

Use one blank line:

1. Before a comment
2. Between methods
3. After a method header
4. After a block of local variable declarations
5. Between logical sections of code so that logically-related statements are grouped

Spaces

Use a space:

1. Between a keyword and a left parenthesis
2. After commas in argument and parameter lists
3. To separate a binary operator from its operands (see exception below)
4. To separate a ternary operator from its operands.
5. Between initialization, expression, and update parts of a for-loop
6. After a cast

Do *not* use a space:

1. Between the dot operator (.) and its operands
2. Between a unary operator and its operand
3. Between a method name and a left parenthesis

indent 8 spaces (2 tabs) instead.

Return statements

1. Do not enclose the return value in parentheses unless they make the return value more obvious in some way.
2. Make the structure of your code match its intent:

Replace this if-else statement:

```
if (booleanExpression) {
    return true;
} else {
    return false;
}
```

with a return statement:

```
return booleanExpression;
```

Replace this code fragment:

```
if (condition) {
    return x;
}
return y;
```

with a return statement:

```
return (condition ? x : y);
```

Ternary statements

The following formats are acceptable:

```
a = condition ? b : c;
a = condition ? b
                : c;
a = condition
    ? b
    : c;
```

1. Parentheses around condition are optional.
2. Use parentheses when the condition is a binary expression:

```
absoluteValue = (x >= 0) ? x : -x;
```

3. Avoid nested ternary statements.
4. Use conditional operator, not if-else statement, when assigning a value to a variable:

```
a = condition ? b : c;
```

while statements

Use the following format:

```
while (condition) {
    statements;
}
```

for statements

Use the following format:

```
for (initialization; condition; update) {
    statements;
}
```

Declare the loop control variable inside for-loop:

```
for (int i = 0; i < size; ++i) {
    statements;
}
```

do-while statements

Use the following format:

```
do {
    statements;
} while (condition);
```

if-else statements

Use the following formats:

```
if (condition) {
    statements;
}
if (condition) {
    statements;
} else {
    statements;
}
if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

switch statements

Use the following format:

```
switch (condition) {
case ABC:
   statements;
   /* falls through */

case DEF:
   statements;
   break;

default:
   statements;
   break;
}
```

1. Always include default case.
2. Use the comment line */* falls through */* when the case label does not have a break statement.

try-catch blocks

Use the following format:

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

Line wrapping for if-statements

Use 8 space rule (2 tabs) when wrapping an if-statement so body is easier to see:

```
if ((a && b)
        || (c && d)
        || (e && f)) {
    statements;
}
```