Coding Standards

Lecture 3b

Agenda

- What is a Coding Standard?
- Why Do We Need a Coding Standard?
- Examples of Coding Standards
- A Review of The Java Coding Standard

What is a Coding Standard?

Coding Standards

- What is a standard?
  - A standard is a technical document designed to be used as a rule, guideline or definition. It is a **consensus-built**, repeatable way of doing something.
    - European Committee for Standardisation (CEN)
  - Standards are created by **bringing together all interested parties** such as manufacturers, consumers and regulators of a particular material, product, process or service. All **parties benefit from standardization** through increased product safety and quality as well as lower transaction costs and prices.

Coding Standards

- What is a coding standard?
  - A coding standard (aka coding convention) is a guideline that defines a programming style.
  - A coding standard ensures that all developers writing the code, in a specific language, write according to the guidelines specified. This makes the code easy to understand and provides consistency in the code.
  - The consistency of applying a coding standard positively impacts the quality of the system.

Why Do We Need a Coding Standard?

Why Do We Need a Coding Standard?

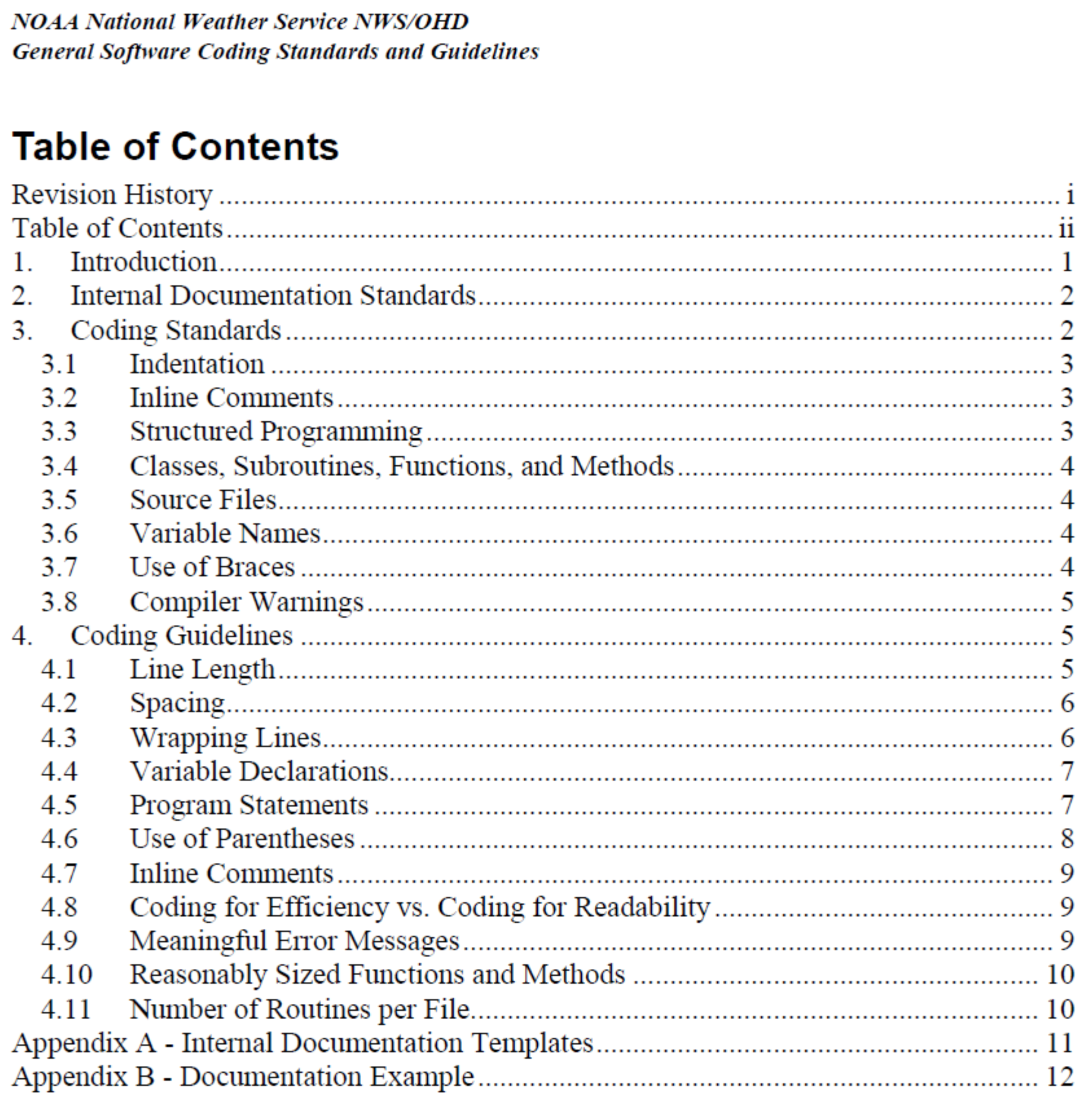
- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code standards improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.
- Reduce the possibility of introducing security vulnerabilities or performance pitfalls in a piece of code.

Common Aspects of a Coding Standard

- Naming Conventions
- File Naming and Organization
- Formatting and Indentation
- Comments and Documentation
- Classes, Functions and Interfaces
- Exception Handling / Logging
- Pointer and Reference Usage
- Testing

Examples of Coding Standards

NOAA National Weather Service NWS/OHD

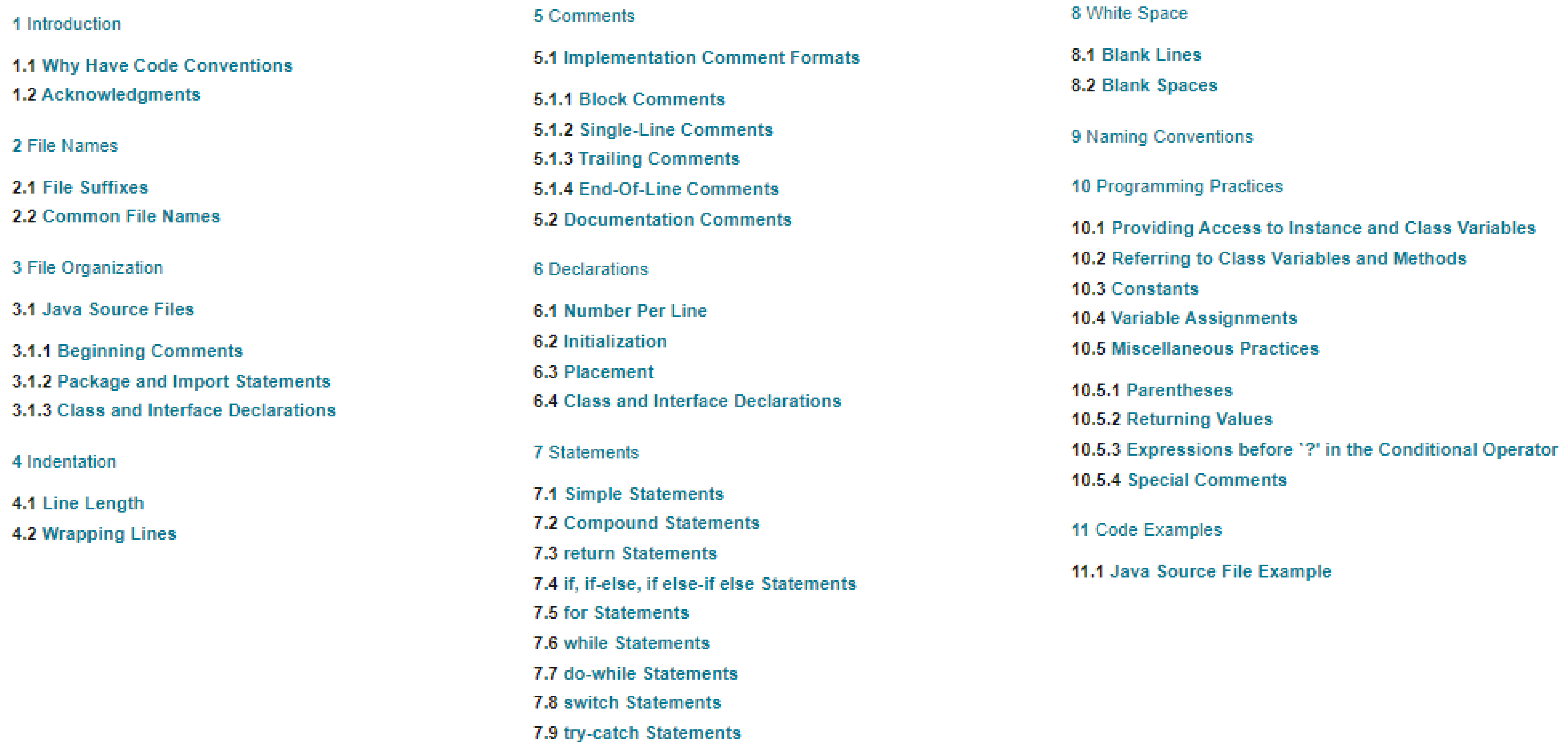


NOAA National Weather Service NWS/OHD
General Software Coding Standards and Guidelines

Table of Contents

The Java Coding Standard

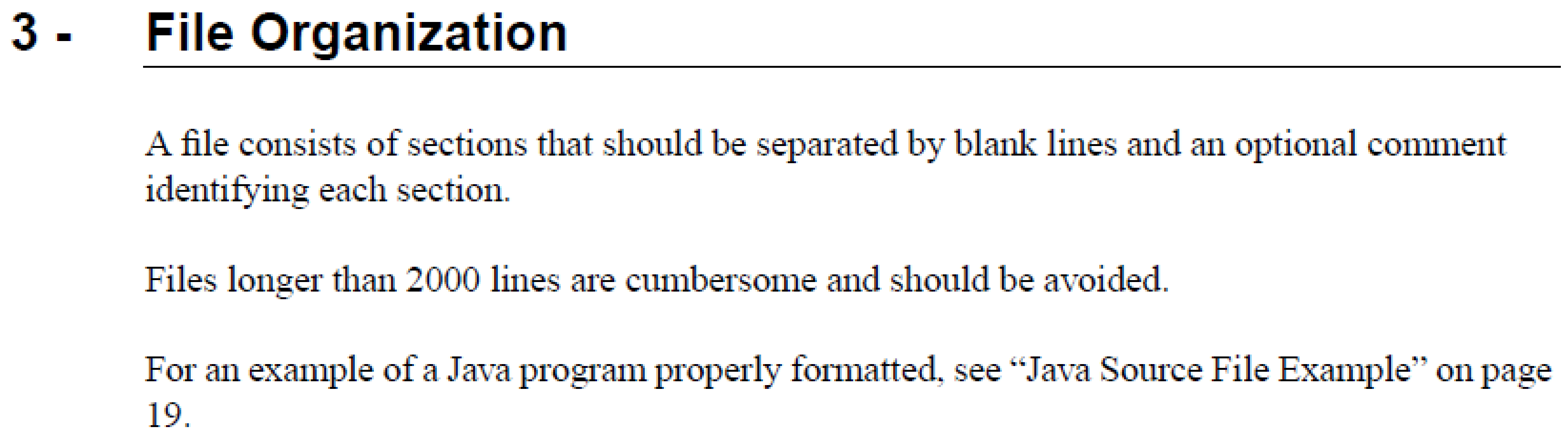
- The structure of the document



A Review of The Java Coding Standard

The Java Coding Standard

- Let us check some sections…



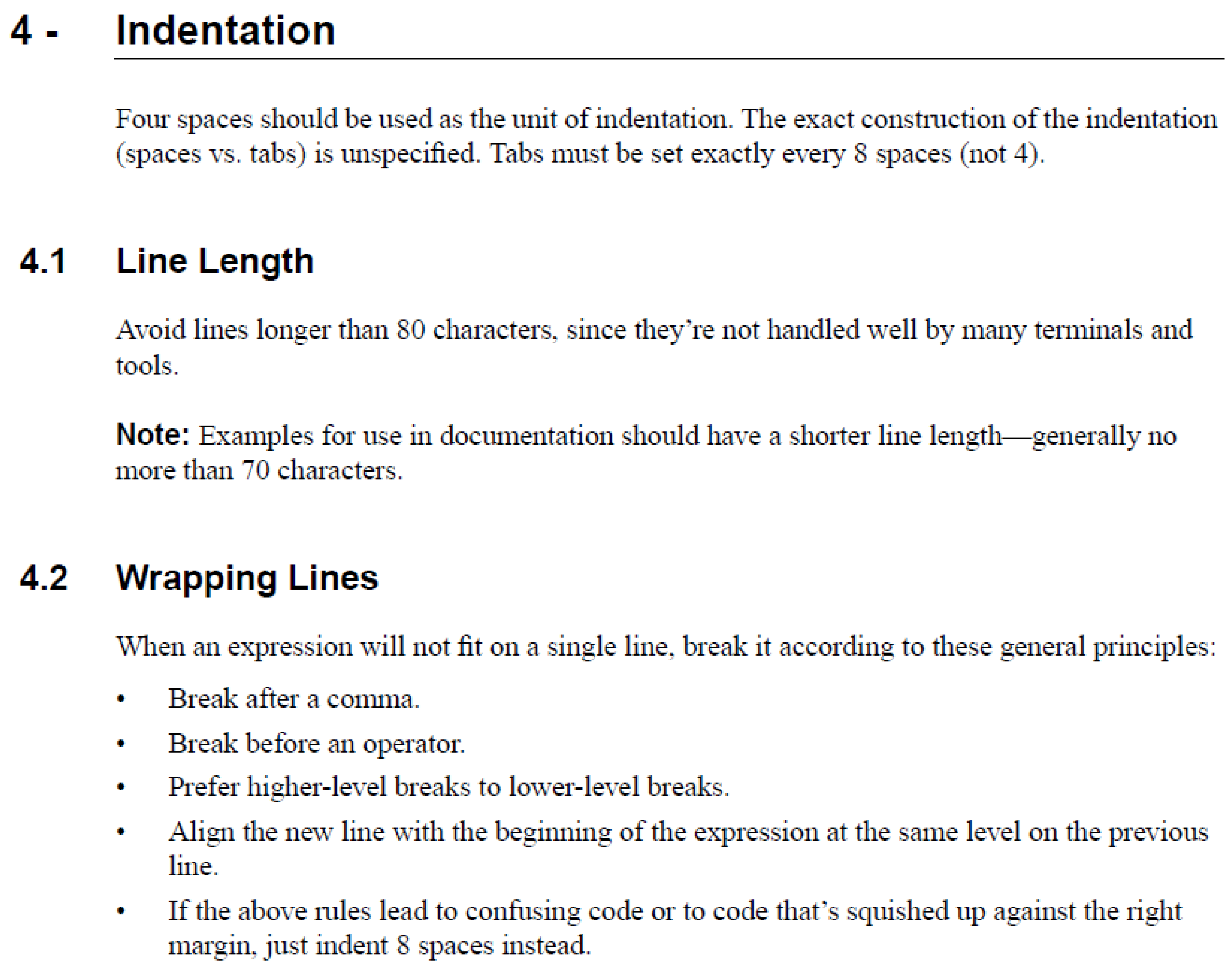
3 - File Organization

A file consists of sections that should be separated by blank lines and an optional comment identifying each section.

Files longer than 2000 lines are cumbersome and should be avoided.

For an example of a Java program properly formatted, see “Java Source File Example” on page 19.

The Java Coding Standard



4 - Indentation

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces (not 4).

4.1 Line Length

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

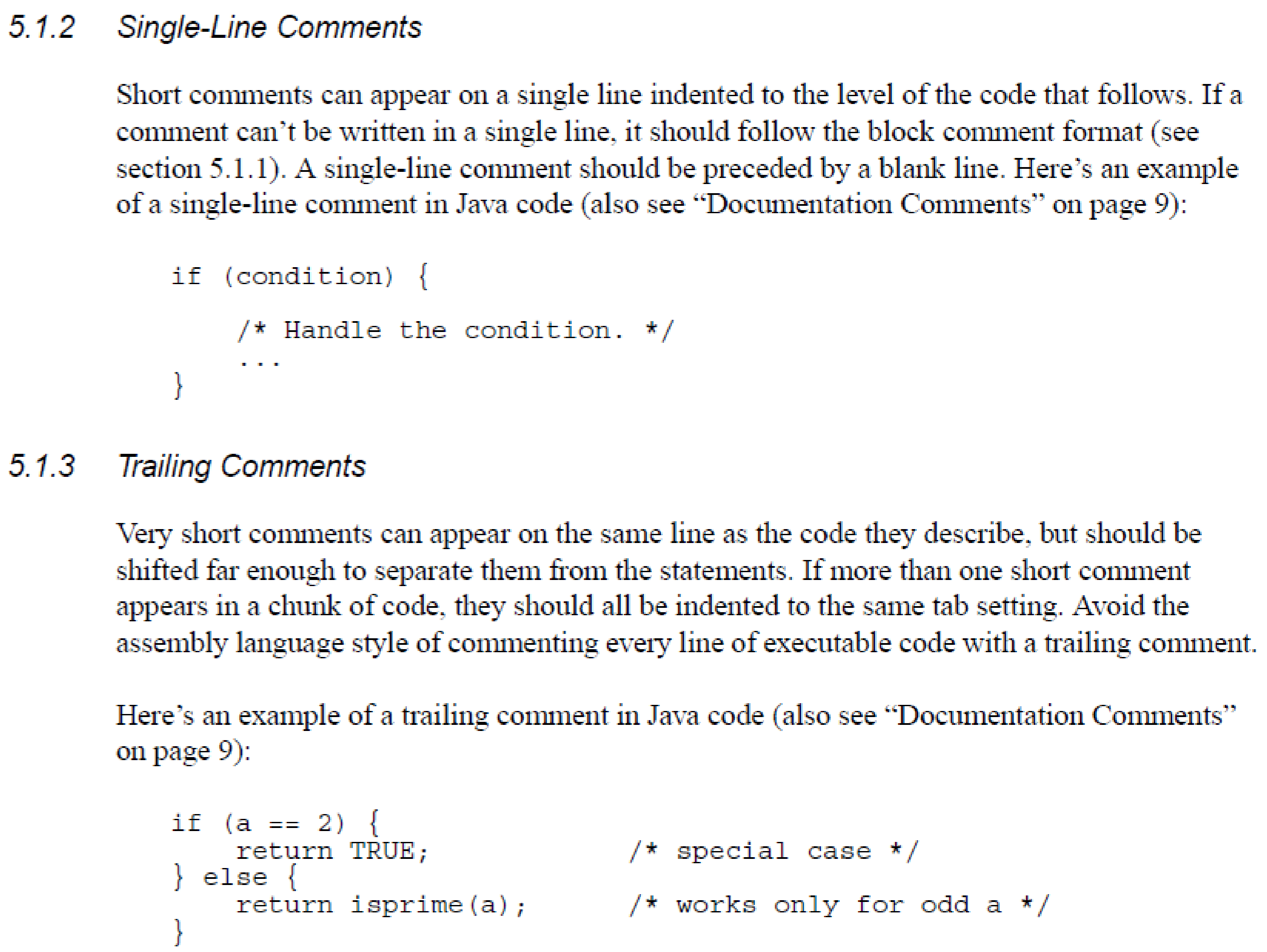
Note: Examples for use in documentation should have a shorter line length—generally no more than 70 characters.

4.2 Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

The Java Coding Standard



5.1.2 *Single-Line Comments*

Short comments can appear on a single line indented to the level of the code that follows. If a comment can't be written in a single line, it should follow the block comment format (see section 5.1.1). A single-line comment should be preceded by a blank line. Here's an example of a single-line comment in Java code (also see "Documentation Comments" on page 9):

```
if (condition) {

    /* Handle the condition. */
    ...
}
```

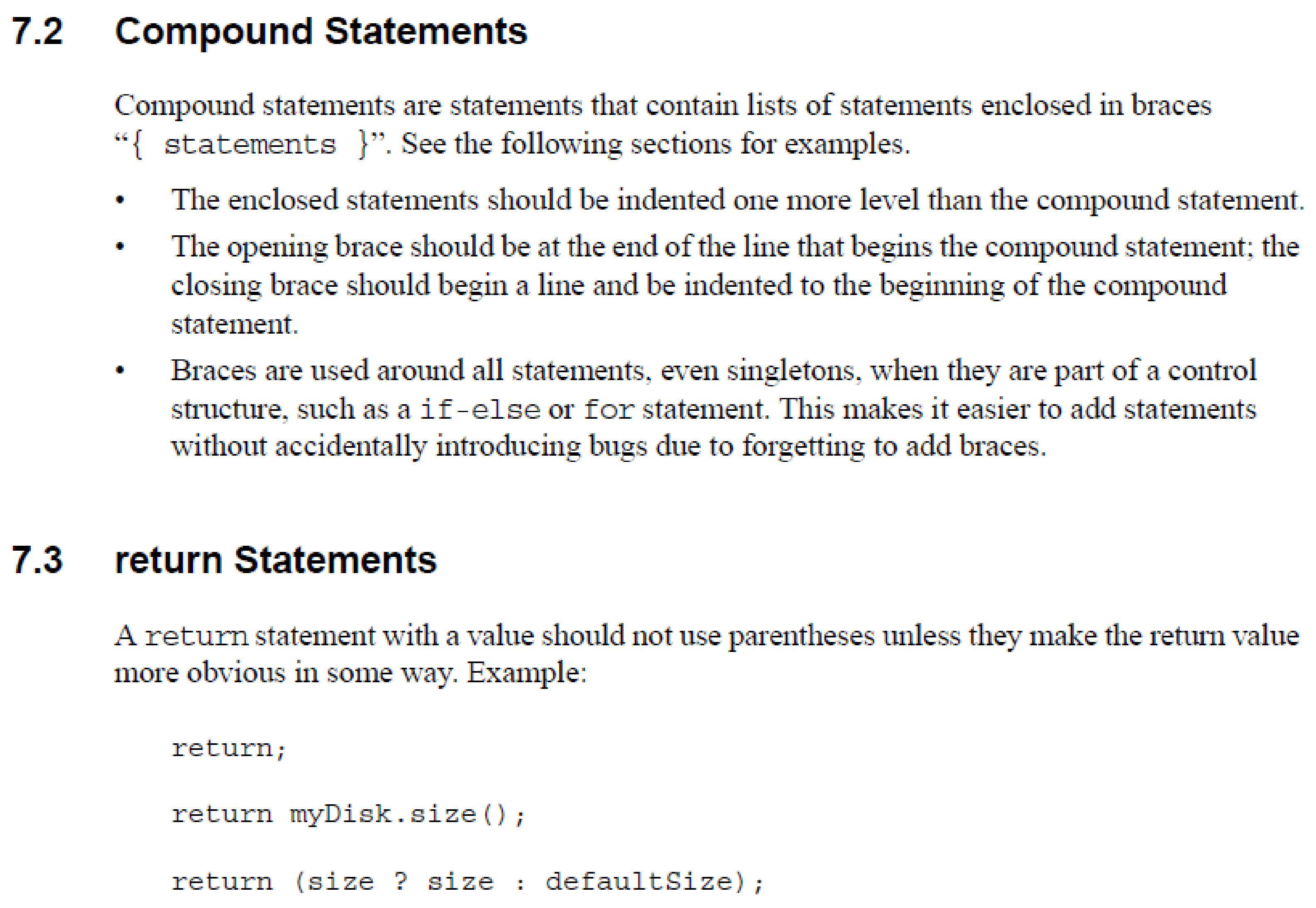
5.1.3 *Trailing Comments*

Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting. Avoid the assembly language style of commenting every line of executable code with a trailing comment.

Here's an example of a trailing comment in Java code (also see "Documentation Comments" on page 9):

```
if (a == 2) {
    return TRUE;             /* special case */
} else {
    return isprime(a);       /* works only for odd a */
}
```

The Java Coding Standard



7.2 Compound Statements

Compound statements are statements that contain lists of statements enclosed in braces "`{ statements }`". See the following sections for examples.

- The enclosed statements should be indented one more level than the compound statement.
- The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.
- Braces are used around all statements, even singletons, when they are part of a control structure, such as a `if-else` or `for` statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

7.3 return Statements

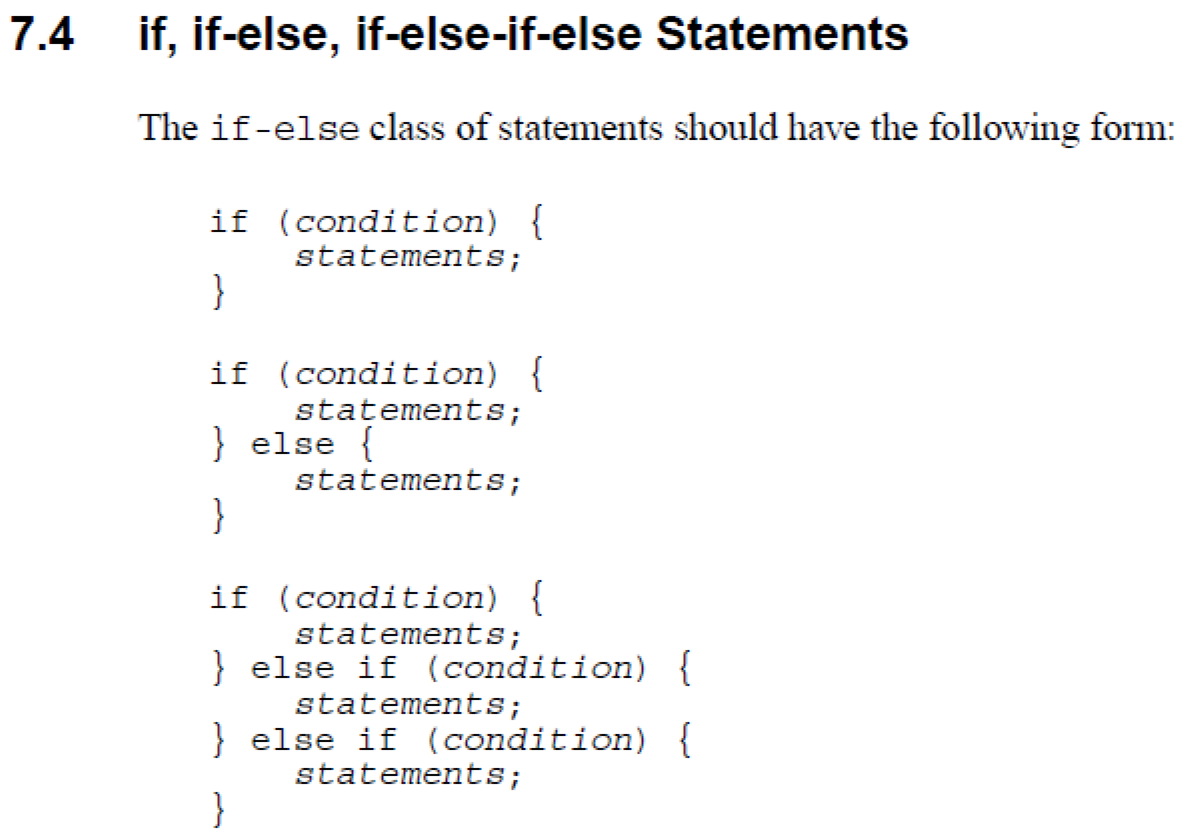
A `return` statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

```
return;

return myDisk.size();

return (size ? size : defaultSize);
```

The Java Coding Standard



7.4 if, if-else, if-else-if-else Statements

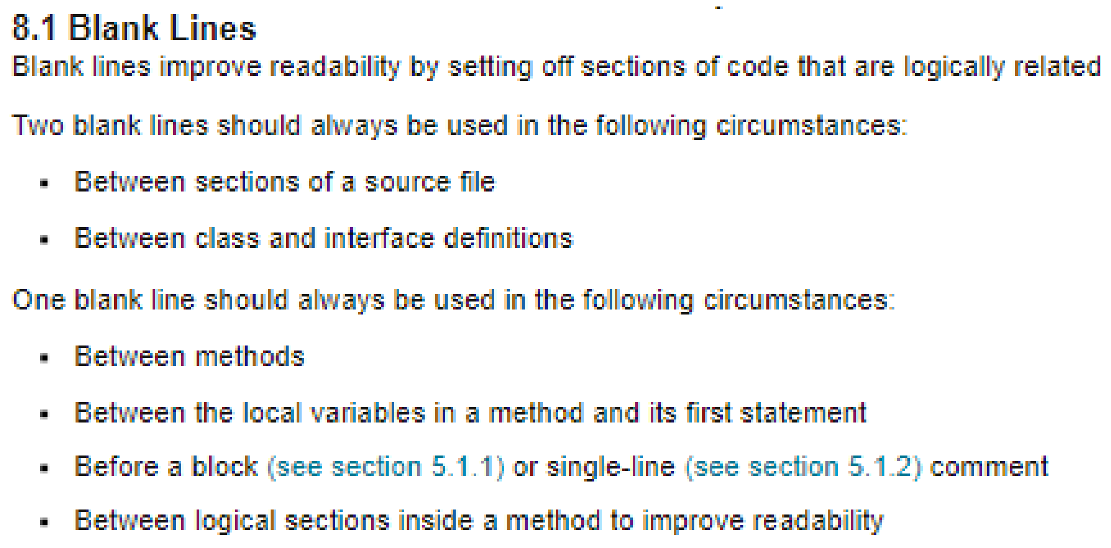
The `if-else` class of statements should have the following form:

```
if (condition) {
    statements;
}

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

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

The Java Coding Standard



8.1 Blank Lines

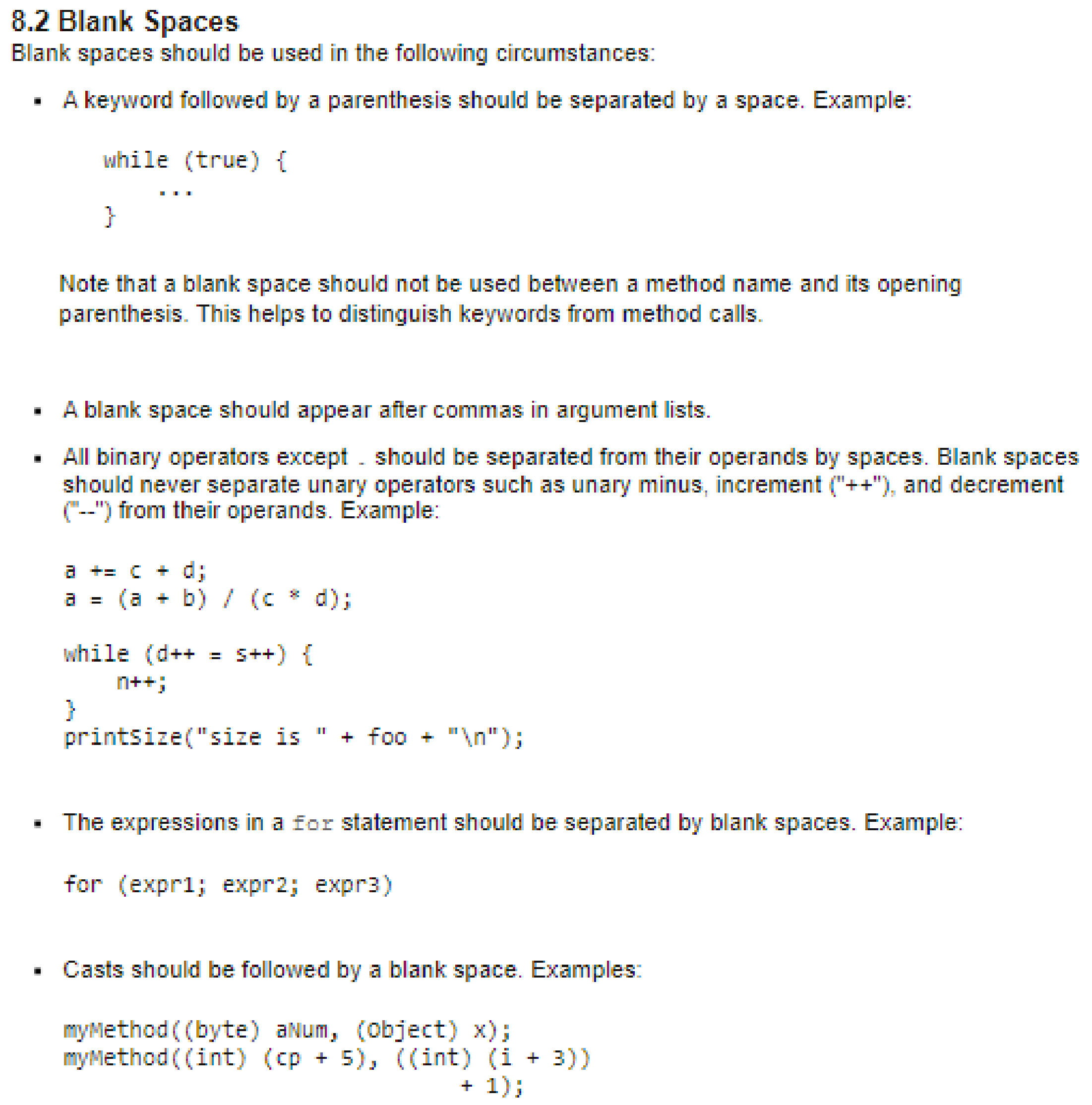
Blank lines improve readability by setting off sections of code that are logically related

Two blank lines should always be used in the following circumstances:

- Between sections of a source file
- Between class and interface definitions

One blank line should always be used in the following circumstances:

- Between methods
- Between the local variables in a method and its first statement
- Before a block (see section 5.1.1) or single-line (see section 5.1.2) comment
- Between logical sections inside a method to improve readability



8.2 Blank Spaces

Blank spaces should be used in the following circumstances:

- A keyword followed by a parenthesis should be separated by a space. Example:

```
    while (true) {
        ...
    }
```

  Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.

- A blank space should appear after commas in argument lists.
- All binary operators except . should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands. Example:

```
a += c + d;
a = (a + b) / (c * d);

while (d++ = s++) {
    n++;
}
printSize("size is " + foo + "\n");
```

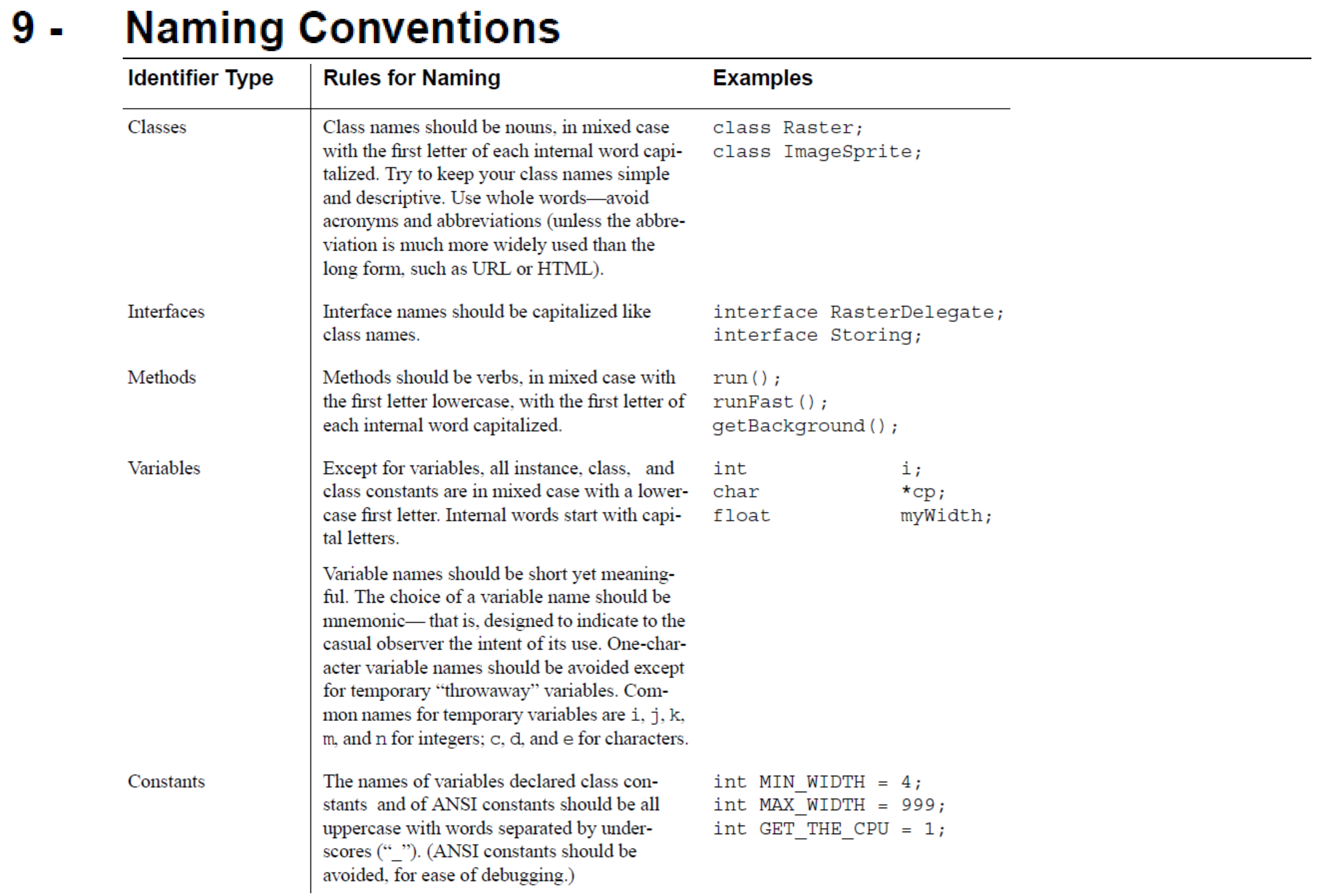
- The expressions in a `for` statement should be separated by blank spaces. Example:

```
for (expr1; expr2; expr3)
```

- Casts should be followed by a blank space. Examples:

```
myMethod((byte) aNum, (Object) x);
myMethod((int) (cp + 5), ((int) (i + 3))
                          + 1);
```

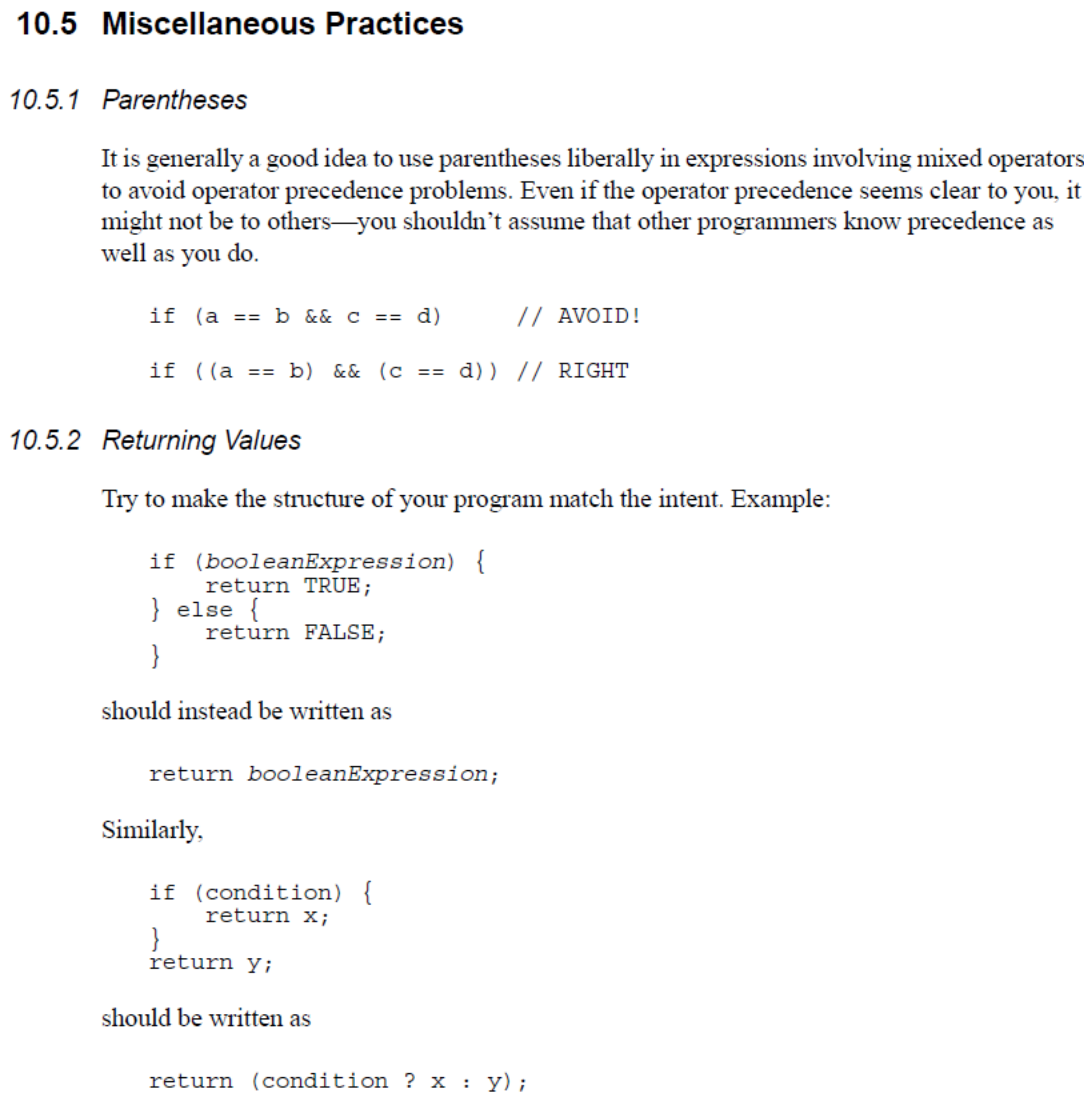
The Java Coding Standard



9 - Naming Conventions

Identifier Type	Rules for Naming	Examples
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words—avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	`class Raster;` `class ImageSprite;`
Interfaces	Interface names should be capitalized like class names.	`interface RasterDelegate;` `interface Storing;`
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	`run();` `runFast();` `getBackground();`
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic— that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	`int i;` `char *cp;` `float myWidth;`
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	`int MIN_WIDTH = 4;` `int MAX_WIDTH = 999;` `int GET_THE_CPU = 1;`

The Java Coding Standard



10.5 Miscellaneous Practices

10.5.1 Parentheses

It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems. Even if the operator precedence seems clear to you, it might not be to others—you shouldn't assume that other programmers know precedence as well as you do.

```
if (a == b && c == d)     // AVOID!

if ((a == b) && (c == d)) // RIGHT
```

10.5.2 Returning Values

Try to make the structure of your program match the intent. Example:

```
if (booleanExpression) {
    return TRUE;
} else {
    return FALSE;
}
```

should instead be written as

```
return booleanExpression;
```

Similarly,

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

should be written as

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

Next On

Take Away Points

- Coding standard.
  - What is it?
  - Why do we need it?
  - Common aspects.
- The way coding standards are documented.

Further Reading

- Oracle Technology Network, “Code Conventions for the Java Programming Language”
  - https://www.oracle.com/technetwork/java/codeconvtoc-136057.html
- National Weather Service, “General Software Development Standards and Guidelines Version 3.5”
  - https://www.nws.noaa.gov/oh/hrl/developers_docs/General_Software_Standards.pdf

Next Lecture

- Code Inspection