Coding Rules

### **2.1 File name**

The source file name consists of the case-sensitive name of the top-level class it contains (of which there is [exactly one](http://google.github.io/styleguide/javaguide.html#s3.4.1-one-top-level-class)), plus the .java extension.

## **Source file structure**

A source file consists of, **in order**:

1. License or copyright information, if present
2. Package statement
3. Import statements
4. Exactly one top-level class

**Exactly one blank line** separates each section that is present.

### **3.1 License or copyright information, if present**

If license or copyright information belongs in a file, it belongs here.

### **3.2 Package statement**

The package statement is **not line-wrapped**. The column limit (Section 4.4, [Column limit: 100](http://google.github.io/styleguide/javaguide.html#s4.4-column-limit)) does not apply to package statements.

### **3.3 Import statements**

#### **3.3.1 No wildcard imports**

**Wildcard imports**, static or otherwise, **are not used**.

#### **3.3.2 No line-wrapping**

Import statements are **not line-wrapped**. The column limit (Section 4.4, [Column limit: 100](http://google.github.io/styleguide/javaguide.html#s4.4-column-limit)) does not apply to import statements.

#### **3.4.1 Exactly one top-level class declaration**

##### **3.4.2.1 Overloads: never split**

When a class has multiple constructors, or multiple methods with the same name, these appear sequentially, with no other code in between (not even private members).

#### **4.1.1 Braces are used where optional**

Braces are used with if, else, for, do and while statements, even when the body is empty or contains only a single statement.

#### **4.1.2 Nonempty blocks: K & R style**

Braces follow the Kernighan and Ritchie style ("[Egyptian brackets](http://www.codinghorror.com/blog/2012/07/new-programming-jargon.html)") for *nonempty* blocks and block-like constructs:

* No line break before the opening brace.
* Line break after the opening brace.
* Line break before the closing brace.
* Line break after the closing brace, *only if* that brace terminates a statement or terminates the body of a method, constructor, or *named* class. For example, there is *no* line break after the brace if it is followed by else or a comma.

#### **4.1.3 Empty blocks: may be concise**

An empty block or block-like construct may be in K & R style (as described in [Section 4.1.2](http://google.github.io/styleguide/javaguide.html#s4.1.2-blocks-k-r-style)). Alternatively, it may be closed immediately after it is opened, with no characters or line break in between ({}), **unless** it is part of a *multi-block statement* (one that directly contains multiple blocks: if/else or try/catch/finally).

### **4.3 One statement per line**

Each statement is followed by a line break.

### **4.4 Column limit: 100**

#### **4.6.1 Vertical Whitespace**

A single blank line always appears:

1. *Between* consecutive members or initializers of a class: fields, constructors, methods, nested classes, static initializers, and instance initializers.
   * **Exception:** A blank line between two consecutive fields (having no other code between them) is optional. Such blank lines are used as needed to create *logical groupings* of fields.
   * **Exception:** Blank lines between enum constants are covered in [Section 4.8.1](http://google.github.io/styleguide/javaguide.html#s4.8.1-enum-classes).
2. As required by other sections of this document (such as Section 3, [Source file structure](http://google.github.io/styleguide/javaguide.html#s3-source-file-structure), and Section 3.3, [Import statements](http://google.github.io/styleguide/javaguide.html#s3.3-import-statements)).

A single blank line may also appear anywhere it improves readability, for example between statements to organize the code into logical subsections. A blank line before the first member or initializer, or after the last member or initializer of the class, is neither encouraged nor discouraged.

*Multiple* consecutive blank lines are permitted, but never required (or encouraged).

##### **4.8.2.1 One variable per declaration**

Every variable declaration (field or local) declares only one variable: declarations such as int a, b; are not used.

**Exception:** Multiple variable declarations are acceptable in the header of a for loop.

##### **4.8.4.2 Fall-through: commented**

Within a switch block, each statement group either terminates abruptly (with a break, continue, return or thrown exception), or is marked with a comment to indicate that execution will or *might* continue into the next statement group. Any comment that communicates the idea of fall-through is sufficient (typically // fall through). This special comment is not required in the last statement group of the switch block.

Notice that no comment is needed after case 1:, only at the end of the statement group.

##### **4.8.4.3 The** default **case is present**

Each switch statement includes a default statement group, even if it contains no code.

**Exception:** A switch statement for an enum type *may* omit the default statement group, *if* it includes explicit cases covering *all* possible values of that type. This enables IDEs or other static analysis tools to issue a warning if any cases were missed.

### **5.1 Rules common to all identifiers**

Identifiers use only ASCII letters and digits, and, in a small number of cases noted below, underscores. Thus each valid identifier name is matched by the regular expression \w+ .

In Google Style, special prefixes or suffixes are **not** used. For example, these names are not Google Style: name\_, mName, s\_name and kName.

### **5.2 Rules by identifier type**

#### **5.2.1 Package names**

Package names are all lowercase, with consecutive words simply concatenated together (no underscores). For example, com.example.deepspace, not com.example.deepSpace orcom.example.deep\_space.

#### **5.2.2 Class names**

Class names are written in [UpperCamelCase](http://google.github.io/styleguide/javaguide.html#s5.3-camel-case).

Class names are typically nouns or noun phrases. For example, Character or ImmutableList. Interface names may also be nouns or noun phrases (for example, List), but may sometimes be adjectives or adjective phrases instead (for example, Readable).

There are no specific rules or even well-established conventions for naming annotation types.

*Test* classes are named starting with the name of the class they are testing, and ending with Test. For example, HashTest or HashIntegrationTest.

#### **5.2.4 Constant names**

Constant names use CONSTANT\_CASE: all uppercase letters, with each word separated from the next by a single underscore. But what *is* a constant, exactly?

Constants are static final fields whose contents are deeply immutable and whose methods have no detectable side effects. This includes primitives, Strings, immutable types, and immutable collections of immutable types. If any of the instance's observable state can change, it is not a constant. Merely *intending* to never mutate the object is not enough. Examples:

These names are typically nouns or noun phrases.

#### **5.2.5 Non-constant field names**

Non-constant field names (static or otherwise) are written in [lowerCamelCase](http://google.github.io/styleguide/javaguide.html#s5.3-camel-case).

These names are typically nouns or noun phrases. For example, computedValues or index.

#### 5.2.6 Parameter names

Parameter names are written in [lowerCamelCase](http://google.github.io/styleguide/javaguide.html#s5.3-camel-case).

One-character parameter names in public methods should be avoided.

#### **5.2.7 Local variable names**

Local variable names are written in [lowerCamelCase](http://google.github.io/styleguide/javaguide.html#s5.3-camel-case).

Even when final and immutable, local variables are not considered to be constants, and should not be styled as constants.

#### **5.2.8 Type variable names**

Each type variable is named in one of two styles:

* A single capital letter, optionally followed by a single numeral (such as E, T, X, T2)
* A name in the form used for classes (see Section 5.2.2, [Class names](http://google.github.io/styleguide/javaguide.html#s5.2.2-class-names)), followed by the capital letter T (examples: RequestT, FooBarT).

### **6.2 Caught exceptions: not ignored**

Except as noted below, it is very rarely correct to do nothing in response to a caught exception. (Typical responses are to log it, or if it is considered "impossible", rethrow it as anAssertionError.)

When it truly is appropriate to take no action whatsoever in a catch block, the reason this is justified is explained in a comment.

## **7 Javadoc**

### **7.1 Formatting**

#### **7.1.1 General form**

The *basic* formatting of Javadoc blocks is as seen in this example:

/\*\*  
 \* Multiple lines of Javadoc text are written here,  
 \* wrapped normally...  
 \*/  
public int method(String p1) { ... }

... or in this single-line example:

/\*\* An especially short bit of Javadoc. \*/

The basic form is always acceptable. The single-line form may be substituted when the entirety of the Javadoc block (including comment markers) can fit on a single line. Note that this only applies when there are no block tags such as @return.

#### **7.1.3 Block tags**

Any of the standard "block tags" that are used appear in the order @param, @return, @throws, @deprecated, and these four types never appear with an empty description. When a block tag doesn't fit on a single line, continuation lines are indented four (or more) spaces from the position of the @.

### **7.3 Where Javadoc is used**

At the *minimum*, Javadoc is present for every public class, and every public or protected member of such a class, with a few exceptions noted below.

Additional Javadoc content may also be present, as explained in Section 7.3.4, [Non-required Javadoc](http://google.github.io/styleguide/javaguide.html#s7.3.4-javadoc-non-required).