Java Coding Guidelines - Part A

1
Introduction

1.1
Fundamental Principals

Java supports the development of reliable, reusable and
readable software. However, no programming language on its own can ensure that this is achieved. Programming has to be done as part of a well-disciplined process.

Clear, understandable and maintainable Java source code is the
primary goal of most of the guidelines provided here. These guidelines can be captured in the following three simple fundamental principles that are consistent with object oriented theory.

Single Point of Maintenance
Whenever possible, a design decision should be expressed at
only one point in the Java source, and most of its consequences should be derived programmatically from this point. Violations of this principle reduce maintainability and reliability, as well as understandability.

Write for Others, Not Yourself
Over its lifetime, source code is read more often than it is
written. Ideally, code should read like an English-language description of what is being done, with the added benefit that it executes.

Minimal Surprise
Reading code is a complex mental process that can be supported
by uniformity, also referred to here as the minimal-surprise principle. A uniform style is a major reason for a team of software developers to agree on programming standards, and it should not be perceived as some kind of punishment or as an obstacle to creativity and productivity.

Minimal Noise
To aid legibility, source code should not be cluttered with
visual “noise” such as bars, boxes and other text with low information content or information that does not contribute to the understanding of the purpose of the software. Comments should only be used to support the code, not paraphrase it.

1.2
Assumptions

The guidelines presented here make a small number of basic
assumptions:
.
The reader knows the Java language
The use of advanced Java features is encouraged
wherever beneficial, rather than discouraged on the ground that some programmers are unfamiliar with them.
.
The reader knows English.
Many of the naming conventions are based on English,
both vocabulary and syntax. Moreover, Java keywords are common English words, and mixing them with another language degrades legibility.

1.3
The Overriding Guideline

Use common sense.

When you cannot find a rule or guideline, when the rule

obviously does not apply, when everything else fails: use common sense, and check the fundamental principles in outlined previously. This guideline overrides all others.

2 Comments

Contrary to a widely held belief, good programs are not characterized by the number of comments, but by their quality.

Comments should be used to complement code, never to paraphrase it. Java by itself is a very legible programming language - even more so when supported by good naming conventions. Comments should supplement code by explaining what is not obvious; they should not duplicate the code syntax or semantics. Comments should help the reader to grasp the background concepts, the dependencies, and especially complex data encoding or algorithms. Comments should highlight deviations from coding or design standards, use of restricted features, and special "tricks." Comment frames, or forms, that appear systematically for each major code construct (such as methods and classes) have the benefit of uniformity and of reminding the programmer to document the code, but they often lead to a paraphrasing style. For each comment, the programmer should be able to answer well the question: "What value is added by this comment?”

A misleading or wrong comment is worse than no comment at all.
Comments are not checked by the compiler. Therefore, in accordance with the single-point-of-maintenance principle, design decisions should be expressed in code rather than in comments, even at the expense of a few more declarations.

2.1 Guidelines for the Use of Comments

Comments should be placed near the code they are associated with, with the same indentation, and attached to that code.

```
aFirstOne = aFirstOne + 1
// This comment
relates to the first one.

// This comment
relates to the second one.
aSecondOne = aSecondOne + 1
```

Use blank lines to separate related blocks of source code (comments and code) rather than heavy comment lines such as:

```
//*************************************************
```

Use empty comments, rather than empty lines, within a single comment block to separate paragraphs:

```
//Some explanation here that needs to be continued
in
//a
subsequent paragraph.
//
//The empty comment line above makes it clear that we
are
//dealing
with a
single comment block.
```

Although comments can be placed above or below the code construct(s) to which they are related, place comments such as a section title or a major piece of information that applies to several code constructs above the construct(s). Place comments that are remarks or additional information below the code construct to which they apply.

Group comments at the beginning of the code construct, using the whole width of the page. Avoid comments on the same line as a code construct. These comments often become misaligned.

Do not replicate information normally found elsewhere. Instead, provide a pointer to the information.

Use code wherever possible, rather than a comment. To achieve this, you can use better names, extra temporary variables, static expressions, and attributes, all of which do not affect the generated code (at least with a good compiler). You can also use small, predicate functions (which will be inlined by the compiler) and split the code into several parameterless procedures, whose names provide titles for several discrete sections of code. For example, replace:

```
if (message.indexOf(&lsquo;,&rsquo;) >= 0)
{
  // A separator was
found.
  do something
}
```

with:

```
separatorFound = message.indexOf(separatorChar)
>= 0
if (separatorFound) {
  do something
}
```

Take care with style, syntax, and spelling in comments.

Do not use accented letters or other non-English characters.

For methods, document the following where appropriate:

- The purpose of the method, but only if it is not obvious from the name;
- Which exceptions are raised and under which conditions;
- Pre- and post-conditions on parameters, if any;

- Additional data accessed, especially if it is modified; this includes especially if the method has side-effects;
- Any limitations or additional information needed to properly use the method.

Remove code that has been commented out except if it needs to be un-commented in the future.

Do not copy JavaDoc comments when implementing interfaces or overriding methods as this reduces maintainability. Use the @link tag to reference the original JavaDoc. For example:

```
/**
 * Implements {@link Object#toString()}.
 */
@Override
public String toString() {
   etc
}
```

Continued...