Coding Conventions [Part II]

Hardik Patel
Mindstix Labs

Comments

- Java programs can have two types of comments: **implementation comments** and **documentation comments.**

- **Implementation comments:**
  - Implementation comments are those found in C++, which are delimited by `/*...*/`, and `//`.
  - Implementation comments are means for commenting out code or for comments about the particular implementation.

Comments

- **Documentation comments:**
    - Documentation comments (known as ‘doc comments’) are Java-only, and are delimited by `/**...*/`. Doc comments can be extracted to HTML files using the javadoc tool.
    - Doc comments are meant to describe the specification of the code, from an implementation-free perspective to be read by developers who might not necessarily have the source code at hand.
- Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself.
- Comments should contain only information that is relevant to reading and understanding the program. For example, information about how the corresponding package is built or in what directory it resides should not be included as a comment.

Comments

- Discussion of nontrivial or nonobvious design decisions is appropriate, but avoid duplicating information that is present in (and clear from) the code.

- It is too easy for redundant comments to get out of date. I general, avoid any comments that are likely to get out of date as the code evolves.

- **Note:** The frequency of comments sometimes reflects poor quality of code. When you feel compelled to add a comment, consider rewriting the code to make it clearer.

- Comments should not be enclosed in large boxes drawn with asterisks or other characters.

- Comments should never include special characters such as form-feed and backspace.

Comments

- **Implementation Comment Formats**
    - Programs can have four styles of implementation comments: **block, single-line, trailing,** and **end-of-line.**
    - **Block Comments**
        - Block comments are used to provide descriptions of files, methods, data structures and algorithms.
        - Block comments may be used at the beginning of each file and before each method.
        - They can also be used in other places, such as within methods.
        - Block comments inside a function or method should be indented to the same level as the code they describe.

Comments

- A block comment should be preceded by a blank linnet set it apart from the rest of the code.

```
/*
 * Here is a block comment.
 */
```

- Block comments can start with `/*-`, which is recognised by **indent**(1) as the beginning of a block comment that should not be reformatted. Example:

```
/*-
 * Here is a block comment with some very special
 * formatting that I want indent(1) to ignore.
 *
 *     one
 *         two
 *             three
 */
```

Comments

- **Note:** If you don't use **indent**(1), you don't have to use `/*-` in your code or make any other concessions to the possibility that someone else might run **indent**(1) on your code.

- **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 single line, it should follow the block comment format.

  - A single line comment should be preceded by a blank line. Here's an example of a single-line comment in Java code:

```
if (condition) {

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

Comments

- **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. Here's an example of a trailing comment in Java code:

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

Comments

- **End-Of-Line Comments**
  - The `//` comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code. Examples of all three styles follow:

```
if (foo > 1) {
    // Do a double-flip.
    . . .
} else {
    return false;  // Explain why here.
}

//if (foo > 1) {
//
//    // Do a double-flip.
//    . . .
//} else {
//    return false;  // Explain why here.
//}
```

Comments

- **Documentation Comment Formats**
  - Doc comments describes Java classes, interfaces, constructors, methods, and fields.
  - Each doc comment is set inside the comment delimiters `/**...*/`, with one comment per class, interface, or member. This comment should appear just before the declaration:

```
/**
 * The Example class provides ...
 */
public class Example { ...
```

  - If you need to give information about a class, interface, variable, or method that isn't appropriate for documentation, use an implementation block comment or single-line comment immediately after the declaration.
  - Doc comments should not be positioned inside a method or constructor definition block, because Java associates documentation comments with the first declaration after the comment.

Declarations

- **Number Per Line**
  - One declaration per line is recommended since it encourages commenting. In other words,

    ```
    int level; // indentation level
    int size;  // size of table
    ```

    is preferred over

    ```
    int level, size;
    ```

  - Do not put different types on the same line. Example:

    ```
    int foo, fooarray[]; // WRONG!
    ```

  - **Note:** The examples above use one space between the type and the identifier. Another acceptable alternative is to use tabs. For example,

    ```
    int         level;       // indentation level
    int         size;        // size of table
    Object      currentEntry // currently selected table entry
    ```

Declarations

- **Initialisation**
  - Try to initialise local variables where they are declared. The only reason not to initialise a variable where it is declared is if the initial value depends on some computation occurring first.
- **Placement**
  - Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces ‘{‘ and ‘}’.)
  - Don’t wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.

```
void myMethod() {
    int int1 = 0;                 // beginning of method block

    if (condition) {
        int int2 = 0;             // beginning of ‘if’ block
        ...
    }
}
```

Declarations

- The one exception to the above rule is indexes of `for` loops, which in Java can be declared in the `for` statement:

```
for (int i = 0; i < maxLoops; i++) { ... }
```

- Avoid local declarations that hide declarations at higher levels. For example, do not declare the same variable name in inner block:

```
int count;
...
myMethod() {
    if (condition) {
        int count;          // AVOID!
        ...
    }
    ...
}
```

Declarations

- **Class and Interface Declarations**
  - When coding Java classes and interfaces, the following formatting rules should be followed:
    - No space between a method name and the parenthesis '(' starting its parameter list.
    - Open brace '{' appears at the end of the same line as the declaration statement.
    - Closing brace '}' starts a line by itself indented to match its corresponding opening statement, except when it is null statement the '}' should appear immediately after the '{'

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}
    ...
}
```

    - Methods are separated by a blank line.

To be covered Next…

- Statements
- White Space
- Naming Conventions
- Programming Practices
- Code Examples