Rule 1: Strive for consistency, clarity, and readability everywhere in your code.

Comments and javadoc

All classes, instance variables, and methods, whether public or private, must have a javadoc comment. You must include the javadoc tags @param, @returns (for non-void methods), @author (for classes), and @throws (for checked exceptions). Do not use javadoc-style comments for anything else.

Method comments should explain briefly what the method does, but not how it works. The first sentence (which plays a special role in the html form of the javadoc) should provide a concise summary. The description for most methods should begin with a short declarative verb phrase in the third person, e.g.

- Returns the length of this list. (good)
- Return the length of this list. (so-so)
- This method will compute the length of this list
  by counting the number of elements and then return it. (really bad)

Methods that override a superclass or interface method normally do not need to be separately javadoc’ed, provided that the supertype itself is documented (and that you use the @Override annotation).

However, if you override or implement a method, and the behavior of the method differs from the description in the supertype documentation, then you must provide a javadoc comment. It is not necessary to repeat the @param or @throws tags. See the section “Additional remarks on javadoc” at the end of this document for more detail.

Internal (non-javadoc) comments

Internal comments should be provided where likely to be helpful to the reader. Internal comments must use the // style comments, not the Javadoc(**) style or C (/*) style.

All comments, except for very brief inline comments, should come before the code they describe and should be indented to the same level. Every non-inline comment should be preceded by a blank line.

Use meaningful names for variables and methods. Single-letter variables such as i, j, and k are acceptable as integer loop counters. Variable and method names should begin with a lower case letter. Class and enum names should begin with an upper case letter. Use camelCase for multi-word names and avoid underscores. Constants (static final variables and enum constants) should be all caps.
**Formatting**

Any reasonable convention is acceptable for formatting code blocks, provided that you are consistent in your use of braces and indentation. However, all method bodies and block statements must be indented relative to the surrounding code, regardless of where you put the braces.

It is recommended that all if/else/while/do blocks include braces, even where not strictly required for compilation. However, Rule 1 takes precedence.

Do not put multiple statements on a line; likewise do not put multiple variable declarations on a line.

Include a blank space around all binary operators, after a comma, and after each semicolon in a for-statement.

Lines longer than 80 characters should be broken onto multiple lines. Continuation lines should normally be indented one additional level.

When adding code to an existing or partially implemented class, you should normally follow the format of the existing code and avoid reformatting code that you did not write.

**Additional remarks on javadoc (optional)**

The javadoc tool is what generates the nicely formatted html pages that you see when you browse the Java API. To generate the html documentation for your own Eclipse projects, just right-click on the project and select Export --> Java --> Javadoc.

The general behavior of the javadoc tool is something like this: Whenever you override a method, if you don't javadoc it, the javadoc tool will copy the entire javadoc from the supertype(s). (Comments in a @throws tag are copied only if the current method actually declares that it throws the exception.)

If you put some text in the “main” part of the javadoc comment (i.e., not in a @param or @throws tag), that text will replace the main text from the supertype, but the @param and applicable @throws tags still will be copied.

If you want to keep the main text from the supertype documentation, but add additional remarks, use the tag {@inheritDoc}, for example:

```java
/**
 * {@inheritDoc}
 * This is some additional text.
 */
```

(Note the curly braces are part of the tag.) The @param and applicable @throws tags will still be
copied over from the supertype documentation.

Note there are two catches in getting all this to actually work:

1) The source code for the supertype must be available on your build path. So if you want it to work for an API interface like java.util.List, you'll need to install the JDK source code. (Since you are not required to turn in your generated documentation, it is not necessary for you to do this unless you want to see the final generated html.)

2) If you have an @Override annotation, it must come AFTER the javadoc comment. This behavior does not seem to be documented anywhere.

References

Some references you might find interesting. Note we do not follow nor enforce every recommendation given in these documents:

http://java.sun.com/j2se/javadoc/writingdoccomments/