Comments

Nothing can be quite so helpful as a well-placed comment. Nothing can clutter up a module more than frivolous dogmatic comments. Nothing can be quite so damaging as an old crufty comment that propagates lies and misinformation.

The proper use of comments is to compensate for our failure to express ourself in code.
So when you find yourself in a position where you need to write a comment, think it through and see whether there isn't some way to turn the tables and express yourself in code.

Truth can only be found in one place: the code. Only the code can truly tell you what it does. It is the only source of truly accurate information. Therefore, though comments are sometimes necessary, we will expend significant energy to minimize them.

Explain Yourself in Code

There are certainly times when code makes a poor vehicle for explanation. Unfortunately, many programmers have taken this to mean that code is seldom, if ever, a good means for explanation. This is patently false.

Good Comments

Legal Comments

Sometimes our corporate coding standards force us to write certain comments for legal reasons. For example, copyright and authorship statements are necessary and reasonable things to put into a comment at the start of each source file.

Informative Comments

It is sometimes useful to provide basic information with a comment. For example, consider this comment that explains the return value of an abstract method:

```
// Returns an instance of the Responder being tested.
protected abstract Responder responderInstance();
```

A comment like this can sometimes be useful, but it is better to use the name of the function to convey the information where possible. For example, in this case the comment could be made redundant by renaming the function: responderBeingTested.

Explanation of Intent

Sometimes a comment goes beyond just useful information about the implementation and provides the intent behind a decision. In the following case we see an interesting decision documented by a comment.

Clarification

Sometimes it is just helpful to translate the meaning of some obscure argument or return value into something that's readable. In general it is better to find a way to make that argument or return value clear in its own right; but when its part of the standard library, or in code that you cannot alter, then a helpful clarifying comment can be useful.

before writing comments like this, take care that there is no better way, and then take even more care that they are accurate.

Warning of Consequences

Sometimes it is useful to warn other programmers about certain consequences.
You might complain that there are better ways to solve this problem.

TODO Comments

TODOs are jobs that the programmer thinks should be done, but for some reason can't do at the moment. It might be a reminder to delete a deprecated feature or a plea for someone else to look at a problem. It might be a request for someone else to think of a better name or a reminder to make a change that is dependent on a planned event. Whatever else a TODO might be, it is not an excuse to leave bad code in the system.

Amplification

A comment may be used to amplify the importance of something that may otherwise seem inconsequential.

Bad Comments

Mumbling

Plopping in a comment just because you feel you should or because the process requires it, is a hack. If you decide to write a comment, then spend the time necessary to make sure it is the best comment you can write.
Any comment that forces you to look in another module for the meaning of that comment has failed to communicate to you and is not worth the bits it consumes.

Redundant Comments

Indeed, it is less precise than the code and entices the reader to accept that lack of precision in lieu of true understanding. It is rather like a gladhanding used-car salesman assuring you that you don't need to look under the hood.

Misleading Comments

Sometimes, with all the best intentions, a programmer makes a statement in his comments that isn't precise enough to be accurate. Consider for another moment the badly redundant

Mandated Comments

It is just plain silly to have a rule that says that every function must have a javadoc, or every variable must have a comment. Comments like this just clutter up the code, propagate lies, and lend to general confusion and disorganization.

Journal Comments

Sometimes people add a comment to the start of a module every time they edit it. These comments accumulate as a kind of journal, or log, of every change that has ever been made.
these long journals are just more clutter to obfuscate the module. They should be completely removed.

Noise Comments
Sometimes you see comments that are nothing but noise. They restate the obvious and provide no new information.
/** The day of the month. */
private int dayOfMonth;

Rather than venting in a worthless and noisy comment, the programmer should have recognized that his frustration could be resolved by improving the structure of his code.

Scary Noise
Javadocs can also be noisy. What purpose do the following Javadocs (from a well-known open-source library) serve? Answer: nothing. They are just redundant noisy comments written out of some misplaced desire to provide documentation.

Don’t Use a Comment When You Can Use a Function or a Variable

Position Markers
Sometimes programmers like to mark a particular position in a source file. For example,
// Actions ////////////////////////////////

A banner is startling and obvious if you don’t see banners very often. So use them very sparingly, and only when the benefit is significant. If you overuse banners, they’ll fall into the background noise and be ignored.

Closing Brace Comments
Sometimes programmers will put special comments on closing braces. Although this might make sense for long functions with deeply nested structures, it serves only to clutter the kind of small and encapsulated functions that we prefer. So if you find yourself wanting to mark your closing braces, try to shorten your functions instead.

Attributions and Bylines
Source code control systems are very good at remembering who added what, when. There is no need to pollute the code with little bylines.

Commented-Out Code
Few practices are as odious as commenting-out code. Don’t do this!
Others who see that commented-out code won’t have the courage to delete it. They’ll think it is there for a reason and is too important to delete.

HTML Comments
Nonlocal Information
If you must write a comment, then make sure it describes the code it appears near. Don’t offer systemwide information in the context of a local comment.

Too Much Information
Don’t put interesting historical discussions or irrelevant descriptions of details into your comments.

Inobvious Connection
The connection between a comment and the code it describes should be obvious.

Function Headers
Short functions don’t need much description. A well-chosen name for a small function that does one thing is usually better than a comment header.

Javadocs in Nonpublic Code
As useful as javadocs are for public APIs, they are anathema to code that is not intended for public consumption.