Program Documentation Guidelines (Java)

Terry Sergeant

Background

Students are aware that programming style and documentation requirements in programming classes can be a source of conflict, especially when a "perfectly working" program receives a less-than-perfect score because of aesthetic concerns. What many students do *not* realize is that programming style and documentation also happens to be a source of conflict among instructors themselves!

It would seem that each person has a slightly different concept of how programs should be written and documented. In this document I plan to outline my expectations for you in this course. After all, *my* opinions on the subject are the only ones that matter!

NOTE: Sun's Java compiler typcially comes bundled with a variety of external utilities. One such utility is `javadoc` which creates HTML documentation directly from comments in your source code. Although the sample program provided makes use of "doc comments", we won't discuss them here.

The "Really Big" Principle

When writing a program you should keep in mind that you are writing for two audiences:

1. the computer / compiler
2. other programmers[1] who may be reading or modifying your code in the future

It is usually fairly easy to tell (at least in small programs like you are assigned in school) whether or not the computer is pleased with your efforts. If the program won't compile or if it produces the wrong output then you have failed to communicate effectively with the first audience. The "people interface" happens to be a bit less well-defined. Although there are several difficulties here, perhaps the biggest is that there is no immediate feedback. If somehow, magically, a program wouldn't compile until the documentation and programming style were acceptable, the quality of documentation would improve dramatically for many students.

What this means is that when writing a program you need to keep both audiences in mind. The compiler will help you make sure you have satisfied the computer and *you* will have to do your best to satisfy the other programmers who read your code.

This document focuses on the audience composed of other people. Before I give you some specific guidelines, let me first give you the "really big" principle that should guide you in your efforts to build beautiful programs. The "really big" principle is this:

Principle 1 (The "Really Big" Principle) *The goal of documentation is to communicate.*

That is, if you fail to communicate with your audience, then your documentation failed. It is easy for students (and employees) to get caught up in making the documentation "fit the requirements" to the neglect of producing documentation that "works!" In your zeal to comply with the "really big" principle, be sure you don't lose sight of its corollary:

Principle 2 (Corollary to "Really Big" Principle) *If you didn't communicate to the instructor then you didn't communicate.*

[1] keep in mind that the "other programmer" may very well be you if you plan to use a program for any length of time at all.

Some Guidelines

Allow me to provide some guidelines that I find helpful in writing documentation that communicates. These guidelines are hopefully illustrated by an attached printout of a Java program that I wrote for another class.

- *Make liberal use of the program header* to provide useful information such as programmer identity, modification dates, version numbers, related files, description of the program, strategies for representing the information and solving the problem, input format, output format, etc.
- *Use vertical spaces and / or horizontal lines* to clearly delineate among distinct pieces of the program.
- *Provide a description of every function/method.* If a function/method needs to be large, treat its header like a program header and provide information as it is useful.
- *Never underestimate the importance of implicit documentation* (such as using descriptive identifiers and visual cues).
- *Avoid stating the obvious.* Keep in mind that we are assuming the person looking at your code is a programmer, so you don't have to explain every little line of code.
- *Explain (at a high level) complicated statements or sections.*
- *Explain what each variable is used for.*
- *Indent your program consistently and intuitively.* Experts suggest that each level of indentation should be at least four spaces. Personally, I have been in the habit of indenting only two spaces for quite a while, which is probably not the best habit because visually matching levels of indentations beyond a few lines can be tricky.
- *Show a little bit of faith in optimizing compilers.* If you have a choice between writing several obvious lines of code and cramming a bunch of functionality into a super-duper-mega-monster-way-cool statement, you should go for the former. A decent compiler will produce very similar code in both cases and the person reading your code will probably invoke fewer curses on your family tree.