

Good Programming Style

Joseph Bonneau, CS 106 A, Autumn 2005

October 12, 2005

1 The Importance of Style

Many novice programmers think that style is a complete waste of time, that any working code is as good as any other. In practice, it has been found that 80% of software costs go into maintenance of existing code, and virtually no modern software is written by only one developer throughout its lifetime. Proper style significantly reduces maintenance costs and increases the lifetime and functionality of software. Most software disasters, whether in CS 106 or the real world, are rooted in poor style. Whatever your feelings are on the matter, the Stanford CS Department considers style to be very important, especially in the 106 level classes. To that end, all of your CS 106 programs will be graded on style.

2 General Principles

The general ideas of what makes good style fall into two categories:

Readability

Good code is written to be easily understood by colleagues. It is properly and consistently formatted and uses clear, meaningful names for functions and variables. Concise and accurate comments describe a natural decomposition of the software's functionality into simple and specific functions. Any tricky sections are clearly noted. It should be easy to see why the program will work, and reason that it should work in all conceivable cases.

Maintainability

Code should be written so that it is straightforward for another programmer to fix bugs or make changes to its functionality later. Functions should be

general and assume as little as possible about preconditions. All important values should be marked as constants which can be easily changed. Code should be robust to handle any possible input and produce a reasonable result without crashing. Clear messages should be output for input which is not allowed.

3 Comments

Comments are the first step toward making computer programs human-readable. Comments should explain clearly everything about a program which is not obvious to a peer programmer. Thoroughness is good, but superfluous comments detract from style as well. The volume of comments written is meaningless, quality is all that counts.

Block comments are written using the `/* comments */` style. They should go at the top of every source file, and generally include your name, (and for CS 106, your section leader's name), the date your code was written, and an overall description of the purpose of that program. Block comments should also precede most functions with a description of the function's purpose, these can be omitted for very short, obvious functions only.

Inline comments are written as `//comments`, they should go near important lines of code within functions, or with variables when they are initialized. A complete commenting example:

```
/*
 * My Name
 * TA: Joseph Bonneau
 *
 * This program does ...
 */
public static class MyProgram extends GraphicsProgram
{
    /*
     * This method does the following ...
     */
    public static void run()
    {
        int x; //x is an integer representing...

        //It is crucial that we multiply x by 7 here.
        x = x * 7;
    }
}
```

```
    }  
}
```

4 Naming

Names given to classes, variables, and functions should be unambiguous and descriptive. Other guidelines:

- Capitalization is used to separate multi-word names: `StoneMasonKarel`.
- The first letter of a class name is always capitalized: `GraphicsProgram`
- The first letter of a function or variable name never is: `setFilled()`.
- Wherever possible, function names should start with a verb: `makeAlternatingRow()` instead of `alternatingRow()`
- The names `x` and `y` should only be used to describe coordinates.
- The names `i`, `j`, and `k` should only be used as variables in `for` loops.
- Other one-letter names should be avoided: `area = base * height` instead of `a = b * h`.
- Names of constants are capitalized, with underscores to separate words: `BRICK_COLOR`.
- Use abbreviations with caution. `max` instead of `maximum` is fine, but `xprd` instead of `crossProduct` is a problem.

5 Indentation

Indentation is used to clearly mark control flow in a program. Within any bracketed block, all code is indented in one tab. This includes the class body itself. Each additional `for`, `while`, `if`, or `switch` structure introduces a new block which is indented, even if brackets are omitted for one line statements. For `if` statements, any corresponding `else` statements should line up. A complete indenting example: `/*`

```
public static class MyProgram extends GraphicsProgram {  
  
    public static void run()  
    {
```

```

int number;

if(number > 0)
{
    if(number < 5)
        number = 7;
}
else
{
    while(number < 6)
    {
        number--;
    }
}
}

```

6 "Magic" Numbers

Raw numbers should almost never appear in your code. Any significant number should be declared as a constant which is easily changeable. This improves readability greatly and ensures that it will be changed equally in all places it is referred. The only common exceptions to this are the numbers -1, 0, 1 which are often used in programming logic. Other numbers can be used directly when they are part of a well known formula only, ie `circumference = 2 * Math.PI * radius`.

7 White Space

White space is meaningless to compilers, but should be used consistently to improve readability. Typically three blank lines are left in between functions. Individual blank lines are used within functions to separate key sections. Use of spaces varies as well, but inserting one space usually makes expressions more readable: `next = 7 * (prev - 1);` is clearer than `next=7*(prev-1);`

8 Function Usage

Functions should be short and accomplish a clear, specific task. As much as possible they should be considered “black boxes” which do not depend on anything except their parameters, and can handle any possible input gracefully. A common rule of thumb is the “Ten Line Rule,” usually functions longer than ten lines are trying to do too much and should be simplified.

Another important aspect of functions is that any repeated segments of code should be made into a separate function. This will shorten your programs and improve readability.

9 Output

A final, overlooked aspect of style is how your programs output results and information to users. Part of writing professional-looking programs is providing clear instructions and results to the users of your programs. This means proper English with no spelling or punctuation errors. This also means clearly explaining error conditions. Always assume that you are writing programs to be used by somebody with no understanding of computer programming whatsoever.

10 The Bottom Line

There is far more to correct style than is mentioned here or in any other guide. For an exhaustive list of standard practices, you can visit <http://java.sun.com/docs/codeconv/>. Still, there is no such thing as perfect style.

For CS 106, you are expected to write code that could be easily read and maintained by another programmer at your skill level, or certainly by the TA’s who will grade your programs. If you are unsure what is the best way to do something, ask. The important thing for your grade is that you make a consistent effort to comply with the principles of good programming style.