CS 062 Lab Style Guide

Like English prose, computer programs can be presented clearly and elegantly. Part of the course is to develop your stylistic judgment and teach you how to write clean, crisp, and correct code. The guidelines here, although by no means complete or authoritative, give you a place to start.

A program is not merely a set of instructions to the computer. It is also a means of communication among people. In real life—as opposed to Computer Science courses—programs are written by *many* people over months or even years. It is critically important that others can easily understand your work. The overall format, comments, and the choice of names all contribute to the clarity of your work and deserve close attention.

1 General Structure

- Be consistent. You may experiment with different formats at different times, but do not do it in a single file.
- Pay attention to style from the time you begin to write, and use it as an aid in structuring your code. Contrary to your feelings as time runs short, it is *never* expedient to write sloppy code and "fix it" later.
- Review your work before you submit it. Be certain that it is *both* correct and well-presented.

2 Blank Lines and Indentation

- Use blank lines to separate logically separate parts of a program. For example, methods should be separated by a blank line. Also, long blocks of code should be broken into logical "paragraphs" with blank lines. On the other hand, do not insert blank lines where they would separate closely related parts of your code.
- Use indentation consistently. Eclipse's format function (click on Source, then Format) or type `CMD+a` to select all and `CMD+i` to indent (where CMD is the apple or command key).
- Give each declaration and statement its own line. Or, equivalently, start a new line after a semicolon.
- Don't make lines too long (< 80 characters is a good rule) as they may get truncated or wrapped when printing – making the code unreadable in either case. You can break a statement over multiple lines and often make it more readable.

3 Comments

- Each part of your program should be introduced by a comment that indicates its purpose, semantics, and use. Early in the course, we will explore the JavaDoc system that promotes this kind of documentation.
  - A class (or program) is normally in a separate file, unless it is an "inner class." It should begin with a comment that states who wrote the code, when, and why. This is the place to advertise any special features of your code. Be sure to use `@author` and `@version` javadoc tags.

  - Each method should be preceded with a comment that tells what the method does, what each parameter signifies, and what value (if any) is returned. Use `@param`, `@return`, and `@throws` as appropriate.

    Pre and post-conditions should be included in the comments for a method. The @return tag will normally include the postcondition in cases where the method returns a value. Preconditions for public methods should be checked at the beginning of a method, with an appropriate exception thrown if the precondition failed. Preconditions for private methods can be checked using an `assert` statement as their failure indicates an error by the programmer of the class, making it hard to recover from the error dynamically. Post-conditions are typically also checked via an assert statement, whether the method is public or private. If evaluating the assert statement is too expensive (e.g., essentially involves re-executing the method) then the assert statement may be omitted.
  - Each variable, or block of closely related variables, should have a comment indicating the purpose of the variable.

- Avoid over-commenting. If there are sufficient general comments, as discussed above, then line-by-line comments will often be unnecessary. In particular, avoid useless comments like the one below.

  ```
  n = n + 1 // add one to n
  ```

- Comments should go above the code or, in the case of very short comments, to the right. Always put a blank line before a comment. Indent comments by the same amount as the code.
- As you are developing and testing a program, it is often useful to "comment out" blocks of code. But the code you submit should have no such comments.
- Comments on variables and methods should be entered when the code is first typed in. This will make it much easier for you and the instructor or TA to figure out what your program is actually doing versus what you think it may be doing.

4 Variables and Names

- Instance variables should be used sparingly, and they should never be `public`. When necessary, provide methods to allow the user of a class to access or modify internal data.
- Use descriptive names. If you are working on a program that simulates a bank account, `balance` is a good name (if the quantity is actually a balance!), while `b` or `boris` are not. For the most part, one-character names are appropriate *only* when they are used locally within a very small block of code.
- The convention in Java is to use all lowercase for variable and method names. Class names begin with an uppercase letter, and constants (indicated by `static final`) are all uppercase.

  When a name is a "compound word," the second and subsequent words begin with an uppercase letter. Examples: `ArrayList`, `binarySearch`, `hashCode`.