Computing and Algorithms III: Style Guide

CS-203 **Summer 2015**

Programming assignments are graded not only on the correctness of the algorithm used, but also upon the style used in the code. A "good" program is not only one which is functionally correct, but one which is easy to read (and thus to modify). A *significant* portion of the total program grade for a given assignment is dependent upon good style.

Good style starts from the moment one begins to write code. It is much easier to take the time to write using good style from the beginning than to try and clean up the style later, especially if "later" is five minutes before the program deadline. Plus, writing with good style can help in understanding and debugging one's own programs.

All programs submitted for grading should follow these guidelines.

1. *Identification:* Begin every file with a comment block which gives your name, login id, class, assignment number, and a brief description of the file. For example:

```
/*************************************************************/
/* Chris Student                                             */
/* Login ID: stud9999                                        */
/* CS-203, Summer 2014                                       */
/* Programming Assignment 42                                 */
/* Book class:  information about a single library book      */
/*************************************************************/
```

2. *File Separation:* Place each class definition in a separate file.

3. *Method Headers:* Each method should begin with a comment block, which explicitly describes the purpose of the method, each parameter passed to the method (along with a comment regarding its purpose), and the return value (if any). For example:

```
/*************************************************************/
/* Method: search                                            */
/* Purpose: Search all the records in the current database   */
/*          to find a particular book                        */
/* Parameters:                                               */
/*   String target:             title of book to find        */
/* Returns: Boolean:            did we find it?              */
/*************************************************************/
```

   Headers for short methods with similar functionality (*e.g.* accessor and mutator methods) may be combined.

4. *White Space:* Use vertical and horizontal white space appropriately. Indent each block of a control structure, and separate the major components of your code with one or more blank lines. Use spacing consistently; a good choice is three to five spaces per level of horizontal spacing and one to two lines for vertical spacing.

5. *Method Length:* Keep your methods relatively short; about a screen to a page in length (roughly 25-50 lines). No subroutine should make more than 7-9 decisions; anything longer should be broken into smaller methods. Do not write methods longer than 75 lines without a good reason.

6. *Line Length:* Restrict the length of lines in your source code to fit upon a printed page (usually, 80 characters). Break method calls into pieces if necessary. (For example, insert a line break between parameters in a method call, or break a long string constant into pieces using the "+" operator and place the pieces on separate lines.)

7. *Meaningful Names:* Choose meaningful names for classes, methods, variables, and parameters. In particular, one-letter variable names (and most two letter names) are *very* bad style and are *expressly forbidden.*

8. *Capitalization of Names:* Begin all class names with an upper case letter and continue with a lower case letter. Begin all method, variable, and parameter names with a lower case letter. The use of upper case letters to join two words (*e.g.* "`LinkedList`", "`getLength`") is encouraged.

9. *Constants:* Use a `final` constant value to define all fixed values which have an easily stated purpose (*e.g.* maximum size of user input, user-selected menu options, elements in a `case` statement). If a number or a character constant has a meaning independent of its particular numerical value, give it a name which reflects that meaning.

10. *Variable Comments:* Comment *every* constant, variable, and member declaration with a description of its purpose.

11. *Inline Comments:* Use comments to describe the tasks performed by every section of code. It is not necessary to comment every line of code, but a subroutine should *never* have no comments at all. (*Note*: comments indicating the end of a block are not sufficient to fulfill this requirement.)

12. *Deprecated Code:* Do not use deprecated code. If the Java compiler complains that you are using deprecated code, re-compile with the "-deprecation" flag to identify the offending code; then replace the offending code with non-deprecated code. (Looking up the offending call in the Java API will yield a pointer to the replacement call.)

13. *Information Hiding:* In general, all variables in non-main classes should be hidden. Use `get` and, where necessary, `set` methods to access variables of other classes.

14. *Modular Design:* Use good design principles when deciding where to place methods within classes. In particular, place methods closest to the data needed to perform that method.

For an example of what *not* to do, check out *How to Write Unmaintainable Code* at:
`http://mindprod.com/jgloss/unmain.html`