CS 455 Style Guidelines

(last modified 1/2021) [Bono]

Coding style, documentation, and good design are important issues in this class. A portion of your program grades will be based on style issues (i.e., not just correctness). The following list is not meant to be exhaustive; but it's a good starting point.

Appendix E in the textbook has a good set of style guidelines, so we recommend reading over those too for further advice about good use of indenting and whitespace and more about the motivation for particular guidelines. With a few exceptions those guidelines are the same as or more restrictive than what's listed below. We put * on items listed here that are *not* required by the texbook guidelines (besides the first one below, we're mostly giving more guidelines on the contents of comments).

1. * always use braces around loop bodies and `if`/`else` actions, even though these actions may only consist of a single statement. A way to help you remember both braces is to type both of them (not on the same line) *before* you type in the body.

2. use a consistent code-layout format. You are expected to indent your code and use white space to display your program structure. The style presented in lecture (and used in all lecture examples) or the style used in the text are both acceptable. Most IDE's include auto-indenting support that will help you with indentation too.

3. indent each level 3 spaces.

4. * each line should have at most one statement.

5. each variable definition should be for just one variable. E.g.,

   ```
   int dimes, nickels; // No
   int dimes;    // Yes
   int nickels;
   ```

6. use informative identifier names (for functions, variables, classes, etc.).

7. use named constants (e.g., `DAYS_PER_YEAR`, not 365). (The lack of this is called "using magic numbers".)

8. use a consistent style for names. Methods, functions, and variables start with a lower case letter and use upper case to delineate "words" within an identifier (e.g., `drawAndLabelXAxis`); constants are all upper case (e.g., `SECONDS_PER_MINUTE`). Classes start with an upper case letter (e.g., `GradSchool`).

9. each method or function should be no more than 30 lines of code. Whitespace, comments, assert statements, and lines with just a single brace won't be counted as part of the 30 lines.

10. all classes' data should be `private`.

11. don't use global variables (C++). there are *very* few situations where they are necessary. Similarly in Java, avoid `static` class variables (unless the variables are `static`

`final`, i.e., named constants).

12. * a generalization of the previous two points: limit the scope of data or methods to only where they are used as much as possible. A few examples: if a variable is used in only one method, it should be local, not a data member. If a method is only needed by other methods of the same class it should be `private`, not `public`.

13. * your main program file should be accompanied by a comment describing what the program does and how to run it.

14. * each function or method should have a comment at its start which describes what it does and describes it's arguments, including any restrictions on them (a.k.a., preconditions), and describes what it returns. For non-static methods this would include what the function tells us about the object or how it changes the state of the object. We're less concerned with the exact format of the comment than the contents. Some possible formats of such comments are the Javadoc format (used in the textbook), or writing them as preconditions and postconditions for the function.

15. * use in-line comments only on any code that isn't self-explanatory or is otherwise difficult to understand. One rule of thumb is that if it was difficult for you to write this code correctly, it probably merits explanation for those that may try to read or modify it. Often, breaking up your complex computations into multiple methods obviates the need for any inline comments.

16. * each class definition should be accompanied by a comment describing what abstraction the class represents, what it's for, how to use it, and with a summary of its functionality.

17. * each class implementation should be accompanied by a comment describing the internal data representation used, including the representation invariants.

Adapted from guidelines written by Mike Clancy.