CS 162 Style Guidelines

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

1. Use a **consistent code-layout format**. The style used in the textbook is acceptable.
2. Always **use braces around loop bodies and `if`/`else` actions**, even though these actions may only consist of a single statement.
3. You are expected to **indent your code** and **use white space** to display your program structure. Each line should be a decent length. Try not to have your code wrap on a terminal because it lowers readability.
4. Numerical constants should not be coded directly. Define the constant value in one place.
5. Use **informative identifier names** (for functions, variables, classes, etc.). For example, use named constants (e.g., `DAYS_PER_YEAR`, not 365).
6. 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`) or use an underscore to separate (e.g., `draw_and_label_X_axis`); constants are all upper case (e.g., `SECONDS_PER_MINUTE`). Classes start with an upper case letter (e.g., `GradSchool`). Avoid names that differ only in case, like `foo` and `FOO`. Similarly, avoid `foobar` and `foo_bar`. The potential for confusion is considerable. Similarly, avoid names that look like each other.
7. Don't use **global variables**. There are *very* few situations where they are necessary.
8. **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`.
9. Your **main program module** should be accompanied by a comment describing **what** the program does and **how** to run it.
10. Each function 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**. We're less concerned with the exact format of the comment than the contents.
11. Use **in-line comments** 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.
12. 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.
13. Programs should include a program **header** at the very top of the file looks like following:

```
/*********************************************************************
** Program name:
** Author:
** Date:
** Description:
*********************************************************************/
```

Adapted from guidelines written by Mike Clancy.