CS 106 / CS 206 - Data Structures
Style and Design Guide

Now that you are no longer a new programmer, you are expected to pay attention to your code organization. Your programming should adhere to the standards explained here. While some of the rules are due to convention, many more are pearls of wisdom distilled from solid software-engineering principles and all of them are essential for making your code readable and/or easy to maintain and modify/reuse. This document explains guiding principles. For more specific formatting rules please refer to the `formatting guidelines`.

Intentional Coding

1. Every line of code should have a deliberate effect on your program. If you are not sure why a line is there, it shouldn't be. If your program stops working because you removed that line of code that you don't understand, you need to understand it before you put it back.

2. Do not copy-and-paste code, only to change a few details. Instead, declare a method or other abstraction and parameterize it.

3. Use `public static final` constants to avoid so-called magic numbers - numerical values that are scattered all over your program and difficult to understand or modify.

4. Do not debug by trying all combinations. If you (or the debugger) pinpointed a line that doesn't work, it is important to spend the time to understand why. The most valuable gain of programming proficiency happens right here. A bug is not truly fixed unless you

   (a) understood why it happend in the first place and,

   (b) understood why changing the code the way you did fixed it

Scoping and Encapsulation

1. Declare variables in the smallest scope possible. That is, prefer local variables over instance variables.

2. Never use `public` (non-`final`) instance variables. Use `private` and accessor methods (getters).

Class Design

In Java, instance variables store the data. Thus it is very important to carefullly consider how to organize your classes and instance variables so that

1. Classes are independent. There must be clear delinearization of responsibiities between different classes so that the work does not overlap. If methods in two different classes are doing the same thing, or data is stored in more than one place, you have bad design.

2. Classes have good communication. Your classes need to work together, thus there must be ways for them to get what they need from each other.

Data Structure Design

Often, there is more than one way to store and represent data.

1. Do not store the same thing more than once, or in more than one place.
2. Choose the smallest storage so that the above is true and accomplishes your algorithmic goals.