Dave's CSE 143 AP Style Guide

Rough guidelines to help you build good coding and commenting style.[1]

Header

Always include a header comment at the top of your assignment. The header should include your first and last name, the date that you turned the assignment in, your section id (e.g., CSE 143 AP), your TA's name (e.g., David Cohen), and the assignment name and number (e.g., Assignment 1, Rectangle-Rama).

Variable and Method Names

When naming variables or methods, be descriptive and succinct. The names need to be understandable.

- Don't call it "x" when you mean "index" or "prime" or "result."
- Conversely, you do not want a variable called "theNewArrayThatIAmCopyingDataInto."
- Avoid declaring many variables with similar names, such as "temp1" "temp2" "temp3" etc.
- In some situations (e.g., a for loop index), a single-letter variable such as "i" is appropriate.

Fields

- Store data in a field only if it is part of your object's state. Do several methods need access to it?
- If it can be a local variable, it should be a local variable.
- If a value should not change while your code runs, declare a class constant (give it a name!).
- Mark your fields private.
- Initialize your fields in a constructor.

Boolean Zen

If you have a Boolean variable, treat it like the true or false that it is.

```
DON'T:      if (bool == true)              DO:   if (bool)

DON'T:      return bool == false;          DO:   return !bool;

DON'T:      if (bool)                      DO:   return bool;
                 return true;
            else
                 return false;
```

Commenting

When you write comments for classes or methods, your audience is the client who will be using the class. Comments are not for the TA (who hopefully understands the code) or for you (who wrote it). Imagine a random person you don't know who wants to use your code but doesn't care how it works — that's who you write for. A good way to get the hang of writing comments is to check out the Java API documentation and see how they write. Whenever we use their methods, we're their client, but we don't know how their code works.

Bad example:

```
      // This is the method that prints the values in list using
      // a for loop.  Should throw exception if the list is null
      public static void print(ArrayList list) { ...
```

What's wrong with this?

[1] Adapted from Michael Schmitz's student style primer and Alyssa "The Saint of Code" Harding's commenting guidelines.

- *Language*. We already know that it's a method, so just say what it does instead of "this method does." Also, don't say the method "should" do something — if you programmed it correctly, it "will" do it.
- *Implementation details*. Maybe later you learn how to use an iterator and decide to change how the method works. But since you've documented that the code uses a for loop, you can't go back and change it. It is therefore better to say *what* the method does rather than *how* it does it.
- *Parameters*. Tell the user what parameters your method takes; what does a client need to provide in order for your method to work properly?
- *Output or return value*. The user probably wants to know how the results look — does it print each value on separate lines or print it in reverse order? That makes a big difference.
- *Exceptions*. The user also wants to know how to avoid breaking the method, so it's good to tell them precisely what will cause an exception. In case they do break the method, they also want to know what type of exception it is in order to handle the error in their program.

Good example:

```
// Prints the contents of an ArrayList as comma-separated values.
// Throws an IllegalArgumentException if the list provided is null.
public static void print(ArrayList list) { ...
```

"But I like commenting how my code works!" you say. That's good! You're a smart programmer. But the best place to put these comments is right inside the code. Then it's in your personal domain and helps anyone actually reading the code to know what it's doing:

```
// for loop to go through the ArrayList and print each value
for (int i = 0; i < list.size(); i++ ) { ...
```

"But what if we used a specific data structure for a reason? Shouldn't the client know?"

We often see comments such as "Uses a SortedMap to maintain sorted order." This is easy to fix: say "Maintains sorted order." It allows you to change your code later.

"The write up / spec gives all the information about the method. Why can't I just copy what it says?"

The write up does give all the information — too much information for the comments, in fact. Use just the the appropriate information the client needs in your comments.

"Do I need to use the pre/post format?"

No, but it is a good way to think about your comments. Think of "pre" as what needs to be true about the parameters or the object for the method to work correctly (e.g., "List must be in sorted order."). Think of "post" as the results the client cares about. Particular commenting style (JavaDoc, pre/post, freehand) doesn't matter, as long as your comments give the client everything needed to clearly understand how to use your methods and your class.

Other Tips

- Eliminate redundancy in your code.
- Keep formatting, comments, etc. consistent throughout your program. Indent!
- Limit your code and comments to 80 or 100 characters per line; break long lines.
- Properly label data fields and methods as public or private.
- Remove any debugging code from your program before you turn it in.
- Be careful to follow every bit of the specification. Straying from it will likely hurt your score. Always check to make sure that any output **exactly** matches the specs, right down to the last whitespace character.
- Double-check everything before you turn in your assignment!