CSE143 Commenting Guidelines
Alyssa Harding
Winter 07

When you write comments for classes or methods, your audience is the client who will be using the class, not for the TA who (hopefully) understands the code or for you who wrote it. So 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 the ArrayList
// using a for loop.  Should throw exception if the list is null
public static print(ArrayList list) { ...
```

What's wrong with this?

- **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 tell *what* the method does rather than *how* it does it.
- **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 list as comma-separated values.
// Throws an IllegalArgumentException if list is null.
public static print(ArrayList list) { ...
```

"But I like commenting how my code works!" you say. That's good! It means 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++ ) { ...
```

More objections:

"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 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. It is a good idea to pull the key points for your comments from the write up, but only the appropriate information the client needs.

"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. Think of "post" as the results the client cares about.

Quick hints:

- DO NOT comment in all caps.
- DO NOT write a book for your comments. This page, for example, is much too long for comments.
- DO comment any special cases the client may care about, ie: "List must be in sorted order."
- DO include a header comment with your name, section, TA, and brief description of the program.
- DO limit comments to 80 characters per line. Longer lines make your print outs look horrible, plus you have to scroll your screen sideways to read them.