Javadocs and CodePro: What Makes the Code Mightier than the Sword

By: Danny Cook

If you find yourself sitting in front of your computer screen reading this document, chances are likely that you've done a fair share of coding in your lifetime. By now, you have most likely developed your own unique coding style, and it is most likely pretty different from the code your friends and colleagues are writing. Of course, this is very natural in the coding world; just like no two authors write the same novel, no two coders are going to compile the same code line for line, character for character. That is what lends us our own uniqueness and signature, and what separates each and every one of us from the pack.

Unfortunately, that "uniqueness" might not always be beneficial. While your own style may suit your needs, chances are most likely very slim that you have been required to work on a class or method in tandem with one other person. Differences of opinion and style can lead to coding confusion and wasted time as you try to dissect each other's code...overall, it can be a real mess. And, if you think that problem is bad, try multiplying it by three!

In CS 2340, you are going to be *required* to work in a group of at least three other students, in order to successfully complete a project. Unless you want to get a horrible score back on your report card at the end of the term, you are *all* going to have to contribute to the coding...and this can lead to some style conflicts, if you aren't careful. Your group members will be reading, analyzing, and working with your code, so you need to make sure you design it as effectively as possible to improve your group's overall efficiency. You also need to know how to make sure your group members know what your code actually *does*, without having to pick up the phone and call you every time they fire up their IDE of choice.

"How am I going to do all of this," you ask? Simple. You will have two very powerful tools at your disposal this semester...and it's up to you to know how to use them to their fullest.

Javadoc: Best Friend, or Worst Enemy?

If you're just coming out of CS 1332, or have done any type of coding in Java in the past, I'm sure the word "Javadoc" rings a bell in that decisive mind of yours. Unfortunately, for some people, it is not always a melodious bell; many of my friends detest Javadoc. They think it is worthless, and a tedious waste of time for coders.

Let me assure you now that Javadoc can be your single best friend in this course. Throughout my group's project this year, I got lost several times because my team members did not properly document their classes and methods...this cost us all a great deal of time. On the other hand, they were able to take one look at my code and

know *exactly* what it was supposed to do, because I laid out my Javadocs well ahead of time. In order to code your best, you need to make sure that others understand what you are doing. Here are two important things to remember:

1. Use the Proper Format When You Javadoc

If your Javadocs are going to be effective, they need to be able to convey all of the necessary information about what you want your function to do...not just bits and pieces.

For example, take a look at an example of this method Javadoc that one of my group members wrote:

```
/**
 * Creates appointment
 */
```

The only information we've gathered here is that this method apparently creates an appointment of some sort. We don't know what goes in, what comes out, or how the method is supposed to be implemented. Without looking at the code itself, digging deep and analyzing, could you tell what was supposed to be going on?

In order to create an effective documentation of a method, you should include the following things:

- A detailed, but concise, description of the method's implementation
- Any and all parameters, with a description of each one (each marked with a @param tag)
- The method's return, if anything other than void (marked with a @return tag). Make sure to not just provide a variable name...describe what the return should be under default conditions.

Let's look at what our Javadoc for this method *should* look like:

```
/**
 *The makeAppointment method takes in the Date, Time, Requested
 *Doctor, and Reason for the new appointment. Using these fields,
 *the method creates a new Appointment object, and returns it to
 *the user.
 *
 * @param date The date of the appointment, as a String
 * @param time The time of the appointment, as a String
 * @param doc The name of the doctor requested for the
 * appointment, as a String
 * @param reason The reason for the appointment, as a String
 * @return The new Appointment object, if created successfully.
 */
```

As you can see, we have provided a description, and all of our proper parameter and return notations. Just from reading the Javadoc, you can tell *exactly* what the function is supposed to do.

Remember, when you Javadoc, don't just focus on the methods; your *class* needs a header, as well. When you write this, make sure to include a detailed description of what your class does, no matter how small its functionality (remember, these documents all are generated as part of the Java API...without this information, your API documents will be very bare!). In addition, make sure to include proper @author and @version tags. NEVER WRITE YOUR NAME AT THE TOP OF A CLASS AND CALL IT DONE.

2. Don't Wait Until The Last Minute To Javadoc!

Because a lot of coders think Javadocing is tedious work, many wait until the very last minute to complete all of it. This is a very bad habit to develop...at the end of our project, we had over 20 classes, each with several methods. My other group members spent three hours each having to go through and add in their Javadocs at 4 in the morning on the day the project was due...this was not a fun experience. I got mine done well in advance, so I was able to spend my time working on other parts of our code.

My best advice to you here is to write your documentation for a method or class as your write the header for it in the code. Don't say "I'll do it when I finish this method," because, chances are, you won't. If you stay on top of your Javadocs as you code, you will end up saving yourself a lot of headache and heartache in the long run.

CodePro: Overcoming Style Differences

Now that you have made your code descriptive enough to allow your group members to know what you were thinking, it's time to refine it even further. To help even more in this course, you will be provided with an Eclipse IDE tool known as CodePro: a software device that will go through and "audit" your code, allowing you to see critical errors in your coding conventions, and style errors that will allow you to make your code even more readable.

Like Javadocs, make sure you audit each class as you code, not at the last minute. It's always better to fix your code in small pieces, because it is overall less work, and you will learn the conventions as you progress.

Unlike Javadocs, however, there is no completely set format that you absolutely have to follow when you use CodePro. Sometimes, it will generate errors that are irrelevant to your code, and are unnecessary and impossible to fix without making your code extremely complicated and difficult to read. In these cases, use your better judgment, and make sure to consult with your TA if you are unsure about any aspect of your coding.

If you follow these guidelines, you will save yourself, and your group members, a great deal of time and energy as your project comes down to the wire. Adopt these techniques early, and use them often, and you will be that much closer to creating a project that blows the rest of your class out of the water!

Good luck!