How to use JavaDoc to document your programs

Mary Ellen Foster

August 22, 2000

1 What is JavaDoc?

You have probably used the web documentation for the Java API, available at

`http://java.sun.com/products/jdk/1.2/docs/api/overview-summary.html`

It explains everything you need to know in order to use the classes in the API libraries. This documentation is convenient to use, not only because it is online, but also because it is cross-linked. So, for example, if you are reading about the `java.io.BufferedReader` class and see that it extends `java.io.Reader`, you can click on a link and immediately read about `java.io.Reader`.

Setting up these fancy web pages by hand would be a lot of work. Fortunately, you can create them automatically using JavaDoc.

There are two steps to using JavaDoc. First, include special "documentation comments" in your program. (A documentation comment is one that starts with `/**` and ends with `*/`.) Then, run JavaDoc. It will extract these special comments and build cross-linked documentation, written in HTML so that it can be viewed on the web.

2 Writing comments for JavaDoc

Figure 1 gives an example program containing the most common kinds of JavaDoc documentation comments. The 148 web site contains the document that JavaDoc creates automatically from this example code. Notice that JavaDoc describes the constructors and methods first in summary, and then in detail.

Now look at the program. Find the comments that start with `/**`. These are the documentation comments. JavaDoc finds the first sentence of each documentation comment by looking for everything up to the first period followed by a space ("`. `"). This sentence will be used as a "summary sentence", so it should contain a brief but complete summary of the class, method, or member variable. You can include more sentences after the summary to give more details; they will be used in the detailed description of the item, but not in the summary.

JavaDoc tags

After the descriptive sentences, there may be several "tags", used to document certain standard components of your code, such as parameters of a method. JavaDoc uses the tags to decide how to format your web document.

Each tag starts with a `@` character. Below are the tags that you should use when documenting your own programs. There are more, but this is a good start:

- `@param`: You should use one `@param` tag for each parameter to a method. It looks like this:
  ```
  @param <parameter name> <description of parameter>
  ```

- `@return`: Any method that return a value (that is, any method that is not declared as `void`) should have a `@return` tag describing what the return value is. It should look like this:
  ```
  @return <description of return value>
  ```

- `@throws`: If a method throws an exception, then there should be a tag describing when that exception is thrown. If it can throw more than one exception, then put in one `@throws` tag for each exception. It should look like this:
  ```
  @throws <exception name> <description of when the exception is thrown>
  ```

- `@see`: This tag can be used to refer the reader to the documentation for a related method or class. To refer to another method of the same class, use a `#` sign, like this:
  `@see #otherMethod`
  To refer to another whole class, you don't need the `#`:
  `@see otherClass` or `@see package.otherClass`
  Finally, if you want to refer to a method in another class, then you combine both of these techniques:
  `@see otherClass#otherMethod()` or `@see package.otherClass#otherMethod()`

HTML tags

HTML (**H**yper**T**ext **M**arkup **L**anguage) is the language that Web pages are written in. It has commands for formatting a web document. You can include HTML in your documentation comments to make the web pages that JavaDoc creates look nicer. For example, in the detailed documentation of method `getName()` we wanted the word "args" to appear in a different font because it is the name of a parameter in the code. This font change was accomplished by including an HTML tag in the documentation comments for that method. (See "`<code>args</code>`" in figure 1.)

Here are some tags that you might want to use in your documentation comments.

- `<code>`: You should use `<code>` and `</code>` tags around any keywords or names that you mention in your description. For example:
  ```
  @throws NoNameException if <code>args</code> contains no element
  ```

- `<br>`: Creates a line break in the HTML output, *i.e.,* moves down to the beginning of the next line.

- `<p>`: Creates a paragraph break in the HTML output, by indenting the next line or skipping some spaces, for example.

- `<ul>` and `<li>`: These tags are used to make a "bulleted" list like the list that you are reading right now. "`ul`" is short for "unordered list" (as compared to an ordered list, in which the items would be numbered), and "`li`" is short for "list item". Here is an example using these HTML tags:

```
* Valid commands are:
* <ul>
* <li> "r": Reads and stores a new arithmetic expression. Any previous
* expression will be lost.
```

```
* <li> "e": Evaluates the current (integer) arithmetic expression and
* prints the result.
* <li> "p": Prints the current arithmetic expression.
* <li> "s": Prints a simplified version of the current arithmetic
* expression.
* <li> "q": Quits the driver program.
* </ul>
```

Although HTML tags can make your JavaDoc documentation look nice, it is not terribly attractive in the code, so don't overuse it.

3 Running JavaDoc

If you are using a development environment, then it is likely that it has its own way of running JavaDoc. If you are using the java development kit (JDK), then here is how to run JavaDoc:

```
javadoc -private *.java
```

With the `-private` option, JavaDoc generates documentation for the private members of your classes, as well as for the public members. This is appropriate when you are generating documentation for a programmer who needs to know the implementation details of your class(es), perhaps because he or she is going to modify the code. It is also the appropriate way to run JavaDoc for your assignments.

If you do not use not use `-private`, JavaDoc will only generate documentation on the public methods and members. This is appropriate when you are generating documentation for a programmer who will only use your code. This is how the Java API documentation was produced.

4 For more information

This document has given you enough information that you should be able to start using JavaDoc. If you are curious to learn more about JavaDoc or HTML, here are some Web pages with more information:

- For more on Javadoc, see the JavaDoc home page at `http://java.sun.com/products/jdk/javadoc/`. It also includes a file called *How to Write Doc Comments for Javadoc* which describes in much more detail what information should go into documentation comments and how it should be presented.
- For more information on HTML, see *HTML Help*, at `http://www.htmlhelp.com/`.

```java
/** This class prints a friendly welcome message. */

public class Hello {

    /** Prints a welcome message. The user must specify his or her name on
     * the command line.
     *
     * @param args the command-line arguments. Should contain at least one
     *             argument, the name of the user.
     * @throws NoNameException if the user does not enter a name.
     * @see #getName
     */
    public static void main( String[] args ) throws NoNameException {
        String name = getName( args );
        System.out.println( "Hello, " + name + "!" );
    }



    /** Extracts the user's name from the input arguments.
     * Precondition: <code>args</code> should contain at least one element,
     * the user's name.
     *
     * @param args the command-line arguments.
     * @return the user's name (the first command-line argument).
     * @throws NoNameException if <code>args</code> contains no element.
     */
    public static String getName( String[] args ) throws NoNameException {
        if( args.length == 0 ) {
            throw new NoNameException();
        } else {
            return args[0];
        }
    }
}
```

Figure 1: A source file with JavaDoc comments